Parallel Stochastic Discrete Event Simulation of Calcium Dynamics in NEURON

Mohammad Nazrul Ishlam Patoary

Doctor of Philosophy

School of Computer Science

McGill University

Montreal, Quebec

2017-12-01

A thesis submitted to McGill University in partial fulfillment of the requirements for the degree of Doctor of Philosophy

DEDICATION

This document is dedicated to my dear parents Nurjahan Akter and Mohammad Althaf Hossain Patoary

ACKNOWLEDGEMENTS

I express my profound gratitude to my thesis advisor, Prof. Carl Tropper, for his continual invaluable guidance and generosity throughout the entirety of my doctoral studies. Carl was more than an excellent supervisor. His care, moral support and encouragement were always very much appreciated. His financial assistance also made everything possible. I am indebted to my co-supervisor Robert A. McDougal for his support, feedback and invaluable suggestions throughout my whole research.

I'll remain very thankful to the fellows of the NEURON project members, especially William Lytton and Michael Hines. I came to learn about many interesting concepts, methods and tools during the weekly project meetings and during the friendly conversations with the members. I would also like to thank my lab colleague Pierre-Luc Bacon, my friends Misbahul Islam and Ibrahim Bousmah for translating the thesis abstract into French.

I would also like to thank you my wife, Rashida Akter, for all she has done ever since I started my Ph.D. Finally, I am ever grateful to my parents, Nurjahan Akter and Mohammad Althaf Hossain Patoary for encouraging me to pursue my doctoral studies.

PREFACE

In computational biology, cell signaling is a communication process between and within each cell in our body that governs basic activities of cells and coordinates all cell actions.

The ability of cells to perceive and correctly respond to their microenvironment is the basis of their development, tissue repair, immunity, and normal tissue homeostasis (the ability of a cell to seek and maintain a condition of equilibrium within its internal environment). It is possible that errors in signaling interactions and cellular information processing are responsible for diseases such as cancer, autoimmunity, and diabetes. Hence by understanding cell signaling, diseases may be treated more effectively.

Neurons use a number of signaling pathways to regulate their internal activity. One such pathway is calcium signaling. The calcium ion plays an important role in neuronal channel dynamics and ultimately in the behavior of the entire neuron. It participates in the transmission of the depolarizing signal and contributes to synaptic activity. Ca^{2+} signaling waves in neurons were discovered recently and are not yet fully understood. However they are thought to play an important role in synaptic transmission. Neuroscientists believe that neuronal transmission has a great impact on the process of learning and on the formation of memory. Intracellular Ca^{2+} signaling in a neuron is controlled by the neurons endoplasmic reticulum and plasma membrane.

Hence correct and efficient simulation of the calcium signaling pathways can help to understand the pathways and to treat neuronal diseases effectively. It is very important to realize that reaction-diffusion models are widely applicable in biology. Other examples include ecological invasions, the spread of epidemics, tumor growth and the healing of wounds. In this thesis I focus on the internal calcium dynamics of a neuron as described (and therefore modeled) by biochemical reactions and diffusions. Stochastic models of such systems provide a more realistic approximation of these systems than existing mathematical (partial differential equation) models.

Stochastic, spatial reaction-diffusion simulations have been widely used in systems biology and computational neuroscience. In this thesis, an optimized time warp synchronization approach was used, Neuron Time Warp (NTW), to simulate a 3 dimensional stochastic reaction-diffusion model of calcium signaling. NTW relies upon the next subvolume method (NSM), which in turn is an extension of Gillespie's Stochastic Simulation Algorithm (SSA). The increasing scale and complexity of simulated models and morphologies have exceeded the capacity of a serial implementation like Gillespie's SSA. It is therefore natural to employ parallel simulation for such complex problems. The main goal of this thesis is to develop a high performance simulator using parallel discrete event simulation (PDES). Newly developed NTW used next sub-volume method and optimistic PDES approach to overcome the computational bottleneck in Gillespie's SSA.

The detailed architecture and work flow of the NTW simulator is described in chapter 3. Prof. Carl Tropper's, supervisor, guidance and suggestions helped me to develop NTW in time. Connecting NTW with existing NEURON simulation environment was also a big challenge in my research. Robert A. McDougal (co-supervisor) helped me to connect

NTW with NEURON, described in chapter 4. To evaluate the correctness and performance of NTW, I employed one ecological model, the Lotka-Volterra model and two major biological computational models of intracellular calcium signaling pathways (calcium buffer model and calcium wave model). The calcium buffer model is employed in order to verify the correctness and performance of NTW by comparing it to a sequential deterministic simulation in NEURON. The derivation of a discrete event calcium wave model using the stochastic IP_3R structure is one of my main scientific contributions of this thesis. This newly derived stochastic discrete event calcium wave model is described in chapter-5. In high performance issue, a dynamic load balancing algorithm and a dynamic window control algorithm for NTW was also developed for the stochastic calcium wave model. We use Q-learning to determine the basic control parameters of the algorithm. Prof. Tropper motivated me to employ Q-learning to optimize the performance of NTW in calcium wave simulation. Developing such an intelligent dynamic load balancing algorithm with dynamic window control is another research contribution of this thesis. All algorithms are described in chapter 6.

ABSTRACT

The intra-cellular calcium signaling pathways of a neuron depends on both biochemical reactions and diffusions. Some quasi-isolated compartments (e.g. spines) are so small and the calcium concentrations are so low that one molecule diffusing into the compartment can make a nontrivial difference in calcium concentration. Such events can affect dynamics discretely in such a way that they cannot be evaluated by a deterministic simulation. Stochastic models of such a system provide a more detailed understanding of these systems than existing deterministic models because they capture their behavior at a molecular level. My research focuses on the development of a high performance parallel discrete event simulation environment, *NTW*, which is intended for use in the parallel simulation of stochastic reaction-diffusion systems such as intra-cellular calcium signaling. We make use of two models, a calcium buffer model and a calcium wave model. The calcium buffer model is employed in order to verify the correctness and performance of *NTW* by comparing it to a sequential deterministic simulation in NEURON. NEURON is a framework for simulating neurons which is used by neuroscientists world-wide. Our work is part of the NEURON project and has the long term goal of developing a multi-scale simulation for large scale neuronal networks. We derived a discrete event calcium wave model from a deterministic model using the stochastic inositol 1,4,5-trisphosphate receptors, IP_3R, structure. A dynamic load balancing algorithm and a dynamic window control algorithm for *NTW* was also developed for the stochastic calcium wave model. We make use of Q-learning to determine the basic control parameters of the algorithm.

Using this algorithm we obtained an improvement in the performance of our simulator of up to 30%.

ABRÉGÉ

Les voies de signalisation du calcium intracellulaire d'un neurone dpendent des ractions biochimiques et de la diffusion. Certains compartiments quasi isols (par exemple les pines) sont si petits et les concentrations de calcium sont si faibles qu'une molcule diffusant dans le compartiment peut faire une diffrence non triviale dans la concentration de calcium. De tels vnements peuvent affecter la dynamique de manire discrte de telle sorte qu'ils ne peuvent pas ltre valus par une simulation dterministe. Les modles stochastiques d'un tel systme fournissent une comprhension plus dtaille de ces systmes que les modles dterministes existants, car ils captent leur comportement au niveau molculaire. Ma recherche se concentre sur le dveloppement d'un environnement de simulation d'vnements discrets parallles de haute performance, *NTW*, qui est destin ltre utilis dans la simulation parallle de systmes de raction-diffusion stochastiques tels que la signalisation de calcium intracellulaire. Nous utilisons deux modles, un modle de tampon de calcium et un modle d'onde de calcium. Le modle de tampon de calcium est utilis afin de vrifier l'exactitude et la performance de *NTW* en le comparant une simulation dterministe squentielle dans NEURON. NEURON est un cadre logiciel pour simuler des neurones qui est utilis par les neuroscientifiques dans le monde entier. Notre travail fait partie du projet NEURON et comme but long terme de dvelopper une simulation multi-chelle pour les rseaux neuronaux grande chelle. Nous avons driv un modle discret d'ondes de calcium d'un modle dterministe l'aide de la structure stochastique IP_3R. Un algorithme d'quilibrage de charge dynamique et un algorithme dynamique de contrle de fenltre pour *NTW* ont galement t dvelopps pour le modle stochastique d'ondes de calcium. Nous utilisons Q-learning pour

dterminer les paramtres de contrle de base de l'algorithme. En utilisant cet algorithme, nous avons obtenu une amlioration de la performance de notre simulateur atteignant jusqu' 30%.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Introduction

A computer simulation is a computation that models the behavior of some real or imagined system over time [1]. Computer simulations have become a vital part in the design of large, complex systems. They are used to evaluate the performance of large systems when analytical solutions and prototyping are too difficult or costly to develop. In a discrete time simulation, the simulation model only considers a change of state at discrete points in simulation time and advances from one such point to another. The best known class of discrete simulation is event driven simulation [1], in which simulation time is advanced from the time stamp of an event to the time stamp of the next event. Processing the events of a sequential simulation is accomplished by using a centralized priority queue of events. Because of the size of these systems it is important to parallelize the simulations.

A parallel discrete event simulation (PDES) is composed of a set of processes which are executed on different processors and which model different parts of the simulated system. Each of these processes is referred as a Logical Process (LP) which communicate with one other via time stamped messages. In computational biology the biochemical systems (e.g. reaction-diffusion systems), can be modeled as discrete event systems [2] and can be simulated in parallel.

The human brain may be viewed as a densely connected network of approximately 86 billion neurons [3]. Biological 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 a neuron, the calcium ion plays a crucial role in neuronal channel dynamics and ultimately in the behavior of the entire neuron [4]. It also acts as a second messenger in the cell and triggers processes such as initiating the biochemical cascades that lead to the changes in receptor insertion in the membrane. This underlies synaptic plasticity, the ability of synapses to strengthen or weaken over time in response to increases or decreases in their activity, to muscle contraction, and the secretion of neurotransmitters at nerve terminals [5]. Because of the importance of calcium in neural transmission the accurate and efficient simulation of intra-cellular calcium dynamics has become an important research issue for neuroscientists.

Intra-cellular calcium signaling pathways make use of biochemical reactions as well as the diffusion of calcium ions. These pathways are an example of a reaction-diffusion system. Such a system describes the population dynamics of one or more species distributed in space which are propelled by two major events: reaction and diffusion. Partial differential equations are commonly used to model such system and employ continuous simulation. These models are not very accurate, however, for a small number of ions in a small compartment such as a neuronal spine [4]. These compartments are so small and calcium concentrations are so low that several calcium molecules diffusing into them can make a nontrivial difference in their concentration. The concentrations in continuous simulators are expressed by real numbers instead of integers, resulting in incorrect behaviors. As a result stochastic discrete-event simulation has emerged as a method to complement differential equations in biochemical simulations [6] [7].

A system consisting of a collection of chemical reactions can be modeled by a chemical master equation. Such an equation models the distribution of the chemical reactants in the system [4] probabilistically for each point in time. In general, it is very difficult to solve this equation. In [8], 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. Simulating a number of these trajectories then gives a picture of the system. The algorithm uses an exponential distribution to compute the time of the next event (i.e. reaction) and employs elementary combinatorics to compute the likelihood of a particular reaction occurring.

[9] describes the Next Reaction Method, which attempts to reduce the computation time of the propensities in Gillespie's algorithm. It makes use of a dependency graph among the reactions which is used to identify the reactions which are in need of an update. Gillespie's algorithm then updates the states of all of the reactants. [10] modifies the Next Reaction Method by re-sorting the event queue in the order of the probability of execution of the reactions. A number of other efforts aimed at improving the efficiency of the Gillespie algorithm have been made, including [11], [12].

A key assumption in the Gillespie algorithm is that the particles are distributed homogeneously in space. This, however, is not the case because particles in neurons are not distributed homogeneously in space because neurons are so large that diffusion cannot equilibrate them and the diffusion of ions in a neuron takes place. This diffusive behavior must be included in a realistic model. Molecular dynamic simulations [13] [14] [15] could be used to model individual particles but 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 (*NSM*) [16]. The *NSM* partitions space into cubes and represents the diffusion of ions between these cubes by events. Other events are used to characterize reactions within the cubes. *NSM* makes use of the Gillespie algorithm to compute next event times within the cubes and relies upon the use of a priority queue to determine the next event and diffusion times. Our parallel simulator, *NTW*, makes use of the *NSM* algorithm because particles are not distributed homogeneously in neurons.

NSM can be applied to PDES by discretizing space into sub-volumes and assigning a sub-volume to each LP. It is necessary to make sure that the events in a parallel simulation are executed in the same order as they would be in a sequential simulation [1], i.e. causality must be maintained. In order to do so, the LPs must be synchronized. There are two main approaches to this synchronization: conservative [17] and optimistic synchronization [18]. Conservative simulations rely on processes blocking, which by definition takes more time and results in deadlocks. At the other extreme, optimistic simulations process events in the order in which they arrive at an LP. No attempt is made to assure that events do not violate causality. Among the optimistic synchronization schemes Jefferson's Time Warp [18] is the most widely employed.

PDES makes us enable to execute discrete event simulation (DES) on a system with more than one processing node. The sequential stochastic discrete event simulation of large scale reaction-diffusion system is very slow whereas PDES is a promising approach to overcome this performance bottleneck [19]. This thesis employs real biological models which are of significance to neuroscientists. So the main focus of my thesis is to develop a PDES simulator which can be used to execute a 3D stochastic reaction-diffusion model.

Here I have used PDES not only to reduce the simulation time but also to find an efficient way to get the parallelism using Gillespies stochastic simulation algorithm in PDES paradigm.

1.2 Discrete Event Simulation

A simulation shows the change in the physical system over time. The properties of a system are represented by states and the changes within the system are modeled by changing these states. For example, in a biochemical reaction-diffusion system, the number of molecules of a chemical substance in a sub-volume at a given instant in time shows the state of that sub-volume at that instant. In general, in order to study the dynamics of a system, our simulation program must include the following elements:

1. A representation of the system's state.
2. Some means of changing the system's state.
3. A representation of simulation time.

The notion of time is central to a simulation. The *simulation time* represents the causal relationship between the events in a physical system. This should not be confused with the wall-clock time, or the execution time of the simulation-it might take an hour to simulate a hundred events in one system, while it takes a few seconds to simulate the same number of events in another system.

Simulations may be classified by their time flow mechanism [1] as being either continuous or discrete. In a continuous simulation the state of the simulation changes continuously over time. Simulations of weather and electrical currents are accomplished by continuous simulations. In a discrete simulation, changes in the systems occur at discrete points in time.

In an event-driven simulation, the simulation time advances with the time stamps of the events. Events are sorted according to their time stamp. In order to implement an event-driven simulation, we need three main data structures.

1. An event list which contains all of the events in increasing time stamp order. The event list is usually implemented with a priority queue [20]. The events in the list are processed until it is either empty or a termination conditions is satisfied. Processing an event may result in the creation of new events.
2. A global clock which shows the simulation time.
3. A set of state variables which represents the state of the simulation and model the physical system.

In general, an event-driven simulation program is comprised of a simulation application and a simulation kernel. The application part is specific to the model, e.g. the biochemical reaction-diffusion simulation model. State variables and the handlers for the events are implemented within the simulation application. The kernel's job is to manage simulation time and calling the functions associated with the events (event handlers) at the right times. Figure 1–1 depicts the main structure of an event-driven simulation program. In the following section, we describe the event driven simulation for stochastic reaction-diffusion system simulation.

1.3 Discrete Event Simulation of a Biochemical System

Consider a biochemical system in a cube (with a volume V) which contains a set of species and reactions involving these species. A reaction occurs when two species interact with each other. State changes correspond to increasing or decreasing the number of molecules of the reactants of the system. The simulation of such a biochemical system

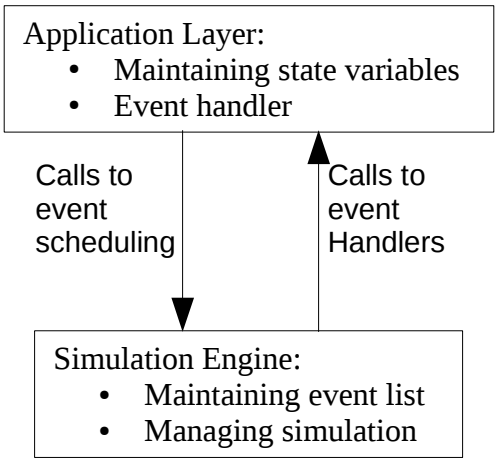


Figure 1–1: The Main Structure of an Event-Driven Simulation Program.

using Gillespie's stochastic simulation algorithm, SSA, [8] defines a discrete event simulation. In SSA, the time of next event i.e. event time is generated by using an exponential distribution.

1.4 Parallel Discrete Event Simulation (PDES)

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 and are referred to as Logical Processes (LPs). The LPs communicate with each other via time stamped messages. Events are stored in an event list and each LP processes all of 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 performed by using a centralized list, such an approach is not possible in parallel simulation as it would be highly inefficient.

It is possible that errors might occur if events are executed out of time-stamp order (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 synchronization and optimistic synchronization which will be discussed in the following two subsections.

1.4.1 Conservative Synchronization

In this scheme, the LPs avoid out of order execution of events. For example, assume that there is an event E_1 at LP_A with time stamp 3 and an event E_2 at LP_B with time stamp 10. If LP_A schedules an event E_3 in LP_B with time stamp 8, the execution of E_3 may affect the execution of E_2 at LP_B. By using a local causality constraint we can avoid the occurrence of such problems.

Such a constraint is obtained if each LP processes events in increasing time stamp order [1]. If each LP adheres to the local causality constraint the results obtained from parallel execution of the simulation program are exactly the same as the results of a sequential execution.

Assume that a simulation consists of N logical processes, LP_0, ... ,LP_{N-1}. Each LP has a current simulation time C_i. Whenever an event is processed, C_i is updated to the time stamp of that event. If LP_i can send a message to LP_j, an input link is established from LP_i to LP_j. The main goal in conservative synchronization is to determine when it is safe to process an event. An event E_i with time stamp T_i at LP_i is safe to process if LP_i can guarantee that it will not receive an event with a time stamp smaller than T_i.

Processes which do not have safe events are blocked and wait until their events become safe to process. One of the basic algorithms which determines which events are safe is the Chandy/Misra's Null message algorithm [21]. In this algorithm it is assumed that events are received in a time stamp order on a link and that there is a FIFO queue on each input link. The algorithm is as follows:

While(simulation is not over)

Wait until each FIFO contains at least one message

Remove smallest time stamped message M from its FIFO

Current simulation time, C:= time stamp of M

Process M

While this algorithm satisfies the local causality constraint, a cycle of empty queues may lead to a situation in which each process in the cycle is blocked and results in a deadlock. Deadlocks are avoided by the use of *lookahead*. If an LP with time stamp T can schedule events with time stamp of at least $T + L$ then L is referred as the lookahead of that LP.

In order to avoid deadlock, each LP_i with time stamp T_i sends a null message with time stamp $T_i + L_i$ to each LP_j which is waiting for a message from LP_i. This null message informs LP_j that LP_i will not send a message with a time stamp less than $T_i + L_i$. As a result some of the messages in LP_j may be executed and deadlock can be avoided [21].

1.4.2 Optimistic Synchronization

In optimistic synchronization LPs allow a violation of the local causality constraint and provide mechanisms to undo any damage caused by the violations. In this approach, LPs do not send messages in time stamp order. Time Warp [18] is the most widely used

of the optimistic synchronization mechanisms. It consists of two control mechanisms to guarantee correctness of the simulation, *Local Control* and *Global Control*. Local controls are implemented within processors while global controls execute a distributed computation performed by all of the processors.

Local Control Mechanism: Each LP in Time Warp has an event list which includes all of the events which are either processed or scheduled but not as yet processed. This event list is referred to as the input queue. The input queue contains messages sent by other LPs. The LP removes the event with the smallest time stamp from its event list and executes it without verifying the safety of the event. The LP may later receive a message with a time stamp smaller than the current time of the LP. This message is referred as *straggler* message. If a straggler message arrives, a rollback to the time stamp of the straggler is performed and all of the processed events with a greater time stamp than that of the straggler are re-executed.

Local control techniques (1) rollback all state variables modified at simulation times which were larger then that of the straggler message and (2) annihilate messages which have timestamps larger then that of the straggler and which were sent to other LPs. Two approaches for rolling back the state variables are *copy state saving*, and *incremental state saving*. Copy state saving saves all to the state variables in a state queue before executing an event. The messages which were sent to other LPs are annihilated via the use of *anti-messages* [18]. An anti-message is the exact copy of the original message along with a flag which indicates that it is an anti-message. The LP saves anti-messages in its output queue. Whenever a straggler message arrives at an LP, an anti-message for all the messages in the output queue which have a time stamp larger than that of the straggler message are sent.

The anti-message annihilates its corresponding message in the input queue of the receiver LP. This is continued until all incorrect processes are rolled back [1].

Global Control Mechanism While the simulation is in progress, memory is consumed by the creation of new messages and by saving the state of LPs. In order to avoid running out of memory, a mechanism is needed to reclaim it (*fossil collection* [22]). By finding a lower bound on the time stamp of future rollbacks, memory used to store events with a smaller timestamp then this lower bound can be deleted. This lower bound is called the *Global Virtual Time (GVT)*. The GVT at real time *(*t) is defined as the minimum timestamp of any unprocessed message or anti-message in the system at *(*t) [1].

In order to compute the GVT each processor calculates the minimum of all of its unprocessed messages and anti-messages, referred as *Local Virtual Time (LVT)*, and sends it to a controller. The controller then computes the global minimum among these values (GVT) and sends it to all to all of the processors. If one could take a snapshot of all unprocessed events and anti-messages in the system at time *(*t), computing the GVT would be trivial. However, there are two challenging problems regarding the computation-*transient messages* and *simultaneous reporting*.

A transit message is a message which has been sent but not received yet. Transit messages are unprocessed messages which disappear from the network and must be considered in the calculation of GVT. Assume that a controller could freeze all the processors of the system and have them report their local minimum among the unprocessed events and anti-messages. It could then compute the GVT. This algorithm is not correct because while all of the processors are frozen, there may be some messages in the network. These messages have been sent by the sender LP but have not been received at the destination.

Figure 1–2 illustrated an example of a transient message. The message with time stamp 15 is a transient message which is not accounted for in either sender or receiver processor.

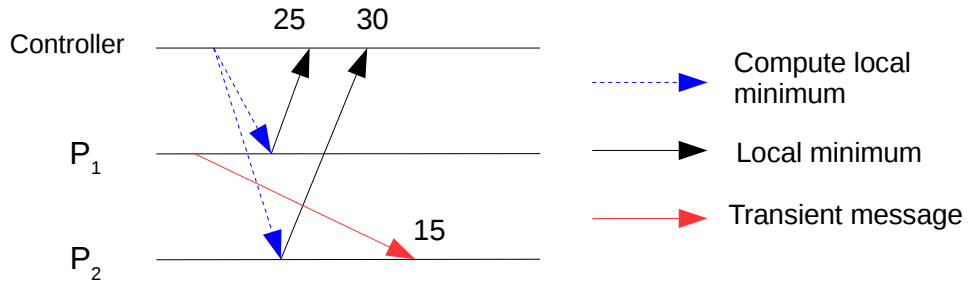


Figure 1–2: The transient message example.

A straightforward solution to the transient message problem is to use message acknowledgements. The idea is to make sure that each transient message is accounted for by either the sender or the receiver processor. A simple synchronous GVT algorithm which uses acknowledgements along with a central controller is shown in algorithm 1.

Algorithm 1 Synchronous GVT Computation Algorithm

Controller:

1. *The controller periodically broadcast a "Start-GVT" message to all the processors to initiate the GVT calculation.*
2. *Upon receive "Receive-Start" from all the processors:*
   - *Broadcast a "Compute-GVT" to all the processors.*
3. *Upon receive the local minimum from all the processors:*
   - *Compute the global minimum and broadcasts this value to all the processors.*

Processors:

1. *Upon receiving the "Start-GVT" message from the controller:*
   - *Stop processing the events.*
   - *Send the "Receive-start" message to the controller.*
   - *Wait to receive "Compute-GVT" from the controller.*
2. *Upon receiving the "Compute-GVT" message from the controller:*
   - *Compute the local minimum time stamp among:*
     - *The time stamp of unprocessed events and anti-messages within the processor.*
     - *The time stamp of the any message the processor has sent but has not received an acknowledgement.*
   - *Send the minimum value to the controller.*

The algorithm with two processors is illustrated in figure 1–3. The first two messages (Start-GVT and Receive-Start) are for synchronizing the processors. When each processor

receives the Compute-GVT message, the processor computes its LVT and sends this value to the controller. Finally the controller computes the GVT and broadcasts it to all the processors.

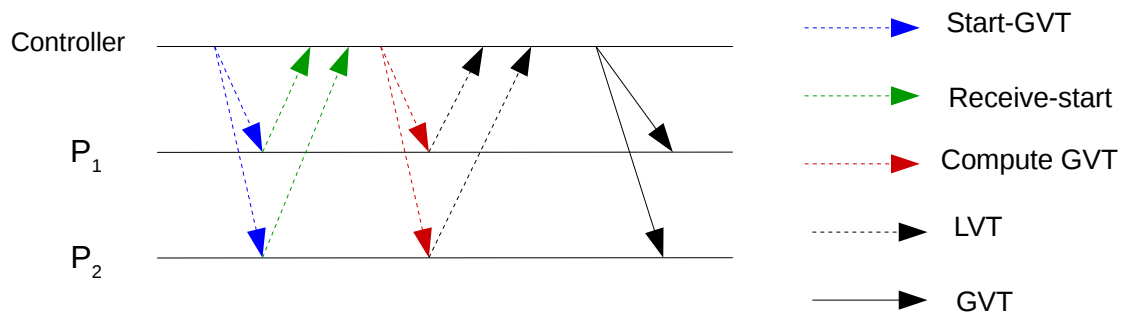


Figure 1–3: An example of synchronous GVT computation.

A drawback of this algorithm is the need to block every processor in the system in the interval between receiving the Start-GVT message and the Compute-GVT messages. An alternative approach is to let the processors keep processing the events while the GVT computation is in progress. This implies using an asynchronous GVT computation algorithm without the need for global synchronization points. Having such an algorithm may lead to the simultaneous reporting problem. The simultaneous reporting problem occurs because not all of the processors report their local minimum at the same point in real time. Hence one or more messages can slip between the cracks and not count towards the calculation of GVT.

Figure 1–4 illustrates an example of the simultaneous reporting problem. The controller broadcasts the Compute-GVT message. P_2 receives the message and reports its local minimum of 25. The Compute-GVT that is sent to P_1 is delayed in the network. Meanwhile, P_1 sends a message with time stamp 15 to P_2. At this time P_1 receives the

Compute-LVT message and reports that its local minimum is 30. Finally, the controller computes an incorrect GVT value of 25.

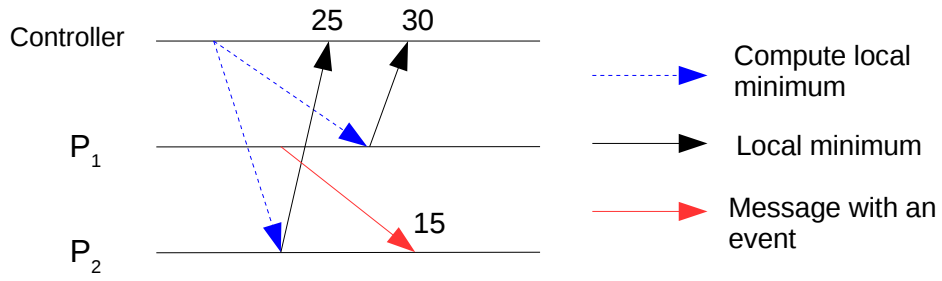


Figure 1–4: An Example of simultaneous reporting problem.

[23] solved the simultaneous reporting problem by making processors send acknowledgements and tagging these acknowledgements in the period starting from the time that the processor sends its local minimum until it receives the new GVT value. *NTW* uses this approach to calculate its GVT.

1.5 Stochastic Simulation Algorithm (SSA) and Next Sub-volume Method (NSM)

There exists many mathematical ways to describe the dynamics of chemical reactions. Deterministic reaction rate equation (RRE) is one of them. The RRE is accurate only when the number of reacting molecules is large enough to allow a continuum point of view. When the number of molecules of a given chemical species is very low, the randomness in the system usually cannot be ignored and stochastic description is essential there.

The chemical master equation (CME) [24] is such a stochastic description which is a mathematical formulation for the time evolution of the probability of the chemical system to occupy every possible discrete state. Basically, CME is derived from the Markov property of the underlying chemical kinetics. If the chemical system is determined by

specifying the number of molecules of each species, then the master equation governs the dynamics of the probability distribution for the system. However, the fact is that such a description suffers from the "curse of dimensionality", i.e., each new species adds one dimension to the problem, thus, the computational complexity grows exponentially.

Monte Carlo algorithms are used to analyze the CME without suffering from the curse which simulate a single trajectory of the chemical system at a time. Simulating a number of these trajectories then gives a picture of the chemical system.

Consider a chemical system with n different species in a volume, V. The time evolution of the state vector $x(t) = x_1(t), x_2(t), \ldots, x_n(t)$, where $x_i(t)$ is the number of molecules of species S_i at time t. Using the derivation in [8], for each reaction channel R_j there exists a constant c_j such that the average probability at which a particular combination of R_j reactants will react in the infinitesimal time interval $[t, t+dt]$ is $c_j dt$. Hence, given the time t and the state vector $x(t)$, the probability $a_j x(t) dt$ that the reaction R_j happens in $[t, t+dt]$ is $c_j dt$ times the total number of reactant combinations. $a_j x(t)$ is called the propensity function for reaction R_j.

The stochastic simulation algorithm (SSA) [8] offers an alternative way to solve the CME directly. SSA, defines a function $p(\tau, \mu)$ such that $p(\tau, \mu) d\tau$ is the probability that, given the state x at time t, the next reaction in the system will occur in the infinitesimal time interval $[t+\tau, t+\tau+d\tau]$, and it will be R. Again, the probability function $p(\tau, \mu)$ can be decomposed as the product of the probability function $p_0(\tau)$ times $a_\mu d\tau$. So the function $p(\tau, \mu)$ becomes:

$$p(\tau, \mu) d\tau = p_0(\tau) a_\mu d\tau \tag{1.1}$$

Here, $p_0(\tau)$ is the probability that, given the state x at time t, no reaction will occur in the time interval $[t, t+\tau]$ and $a_\mu d\tau$ is the probability that the reaction R will occur in the time interval $[t+\tau, t+\tau+d\tau]$.

The direct SSA is stated as follows:

1. Set the time variable t := 0 and the state variable x to the initial state.
2. Calculate the propensity functions $a_j(x)$, $1 \leq j \leq M$, for the current state $x(t) = x$ and the sum $a_0(x) = \sum_j a_j(x)$
3. Generate two uniform $U(0,1)$ random numbers r_1 and r_2 and then choose the next reaction time by

$$\tau = (1/a_0) ln(1/r_1) \tag{1.2}$$

and the reaction channel μ as the integer that satisfies the inequality

$$\sum_{j=1}^{\mu-1} a_j(x) \leq r_2 a_0 \leq \sum_{j=1}^{\mu} a_j(x) \tag{1.3}$$

4. Update the system state and $t := t + \tau$. Repeat from Step 2 until the final time t_f is reached.

In order to speed up the original SSA algorithm, improvements have been made by adopting different approximation techniques. [9] describes the Next Reaction Method, which attempts to reduce the computation time of the propensities in Gillespie's algorithm. [10] modifies the Next Reaction Method by re-sorting the event queue in the order of the probability of execution of the reactions. More efforts also aimed at improving the efficiency of the Gillespie algorithm have been made, including [11], [12]. Gillespie's SSA assumes the homogeneous distribution of particles in space however the particles in neurons are not distributed homogeneously in space. To include diffusive behavior of

neuron particles, molecular dynamic simulations [13] [14] [15] could be used to model individual particles but take into account inter-particle forces, rendering them computationally expensive. Instead, we can use a less expensive Monte Carlo algorithm, the next sub-volume method (*NSM*) [16].

NSM is a discrete-event approach towards the simulation of reactions and diffusion of species inside of a volume containing an inhomogeneous distribution of particles [16]. It is a generalization of the SSA [11]. In NSM the simulation is divided into sub-volumes each of which is assumed to be well-stirred. Figure 1–5 contains such a sub-volume.

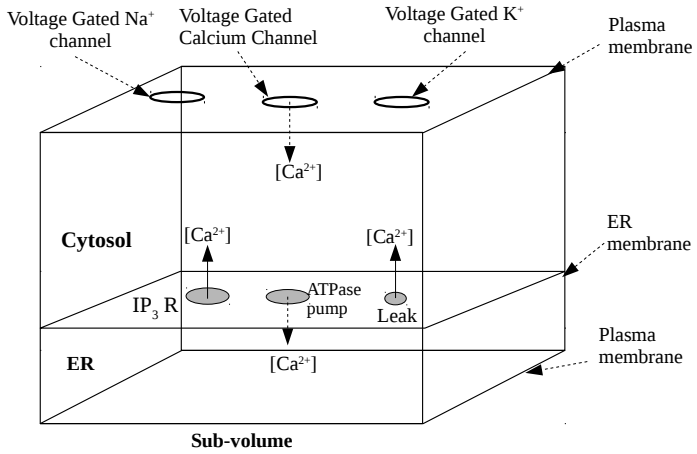


Figure 1–5: Computational sub-volume

The state of the system is updated by performing an appropriate reaction within a single sub-volume or by allowing a molecule to move to a randomly selected neighboring sub-volume. There are two types of events used in this approach: 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.

1.6 Time Warp for Stochastic Reaction Diffusion Simulation

[2] points out that a conservative synchronization is not suitable for use with the *NSM* due to the zero lookahead property (in which a simulator at logical time T can only schedule a new event with time stamp T) of the exponential distribution. This leads Time Warp as the only alternative.

The natural outcome of using Time Warp is that space is divided into sub-volumes and a collection of sub-volumes is assigned to each LP. Interactions between LPs consist of diffusion events between neighboring sub-volumes. [25] [2] and [19] evaluate several Time Warp based approaches. *NTW* also uses *NSM* and inherits features from previously recognized simulators, Clustered Time Warp (CTW) [26] and XTW [27].

In CTW, LPs are grouped together to form a cluster. The main idea behind this is that large simulation, LPs belonging to the same functional units can be grouped together. These groups are referred to as a cluster. There is no limitation on the number of clusters in a system. The only restriction is that each cluster can be mapped onto one processor - it cannot be divided between processors.

XTW uses a multi-level queue (XEQ) along a single rb-message. XEQ has an O(1) cost bounded by the number of simulated entities (as opposed to the number of events). An rb-message not only reduces the cost of annihilating previously sent messages, but also reduces the memory cost by eliminating the output queue for each LP. *NTW* inherits both

the clustering approach and the multilevel queuing scheduling (i.e. XEQ) with a single rb-message in. Chapter 3 describes NTW in some detail.

1.7 NEURON

NEURON [28] is a simulation environment primarily developed by Michael Hines at Yale. In this environment, complex nerve models can be created by connecting multiple one-dimensional sections together to form arbitrary neuron morphologies. NEURON also allows for the insertion of membrane properties in these sections (including channels, synapses, and ionic concentrations). In NEURON, the ion channels of axons and soma [4] are typically of the Hodgkin-Huxley type [29]. NEURON is very flexible and supports a wide class of models. Recently, NEURON was extended to incorporate reaction-diffusion models so that it would be able to simulate chemical reactions of diffusive ions (example Ca^{2+}) as well as electric signals and allow these two types of information to be combined to understand diseases [30]. NEURON as released currently only support deterministic, although there is infrastructure to support stochastic using NTW.

[30] focus how extended NEURON supports in research on the role of cell biological principles (genomics, proteomics, signaling cascades and reaction dynamics) on the dynamics of neuronal response in health and disease. [31] is the example of the multiscale simulation using extended NEURON for Pharmacological Treatment of Dystonia in Motor Cortex.

The main goals of my thesis are to develop a high performance parallel simulation environment for the stochastic reaction-diffusion systems by using PDES and to integrate

this simulator with existing deterministic models of reaction-diffusion systems. This research is a part of NEURON the project. More precisely, it will be integrated within NEURON (www.neuron.yale.edu). A sequential version of *NTW* is already integrated with NEURON. The experimental integration of parallel *NTW* with NEURON is described in chapter 4. Developing parallel reaction diffusion models for the simulation of neurons is just beginning here. The work described in this thesis should be viewed as a first step in this direction.

1.8 Intracellular Signaling Pathways of a Neuron

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 biochemical chain of reactions and/or diffusions inside the cell and then creates a response.

In a neuron, the calcium ion plays an important role in neuronal channel dynamics and ultimately in the behavior of the entire neuron. It behaves like a second messenger in the cell and triggers processes such as initiating the biochemical cascades that lead to the changes in receptor insertion in the membrane. This underlies synaptic plasticity, the ability of synapses to strengthen or weaken over time in response to increases or decreases in their activity, to muscle contraction, and the secretion of neurotransmitters at nerve terminals. Because of the importance of calcium in neural transmission the accurate and efficient simulation of intra-cellular calcium dynamics has become an important research issue for neuroscientists.

In this thesis we simulate two major models of calcium signaling pathways, a calcium buffer and a calcium wave model. The calcium buffer model is employed in order to verify

the correctness and performance of *NTW* by comparing it to a sequential deterministic simulation in NEURON. We also derived a discrete event calcium wave model from a deterministic model using the stochastic IP_3R channel structure so that we can use it on our developed *NTW*. Details about the derived models are given in chapter 5.

1.9 Thesis Motivations and Structure

Stochastic, spatial reaction-diffusion simulations have been widely used in systems biology and computational neuroscience. Specifically, in computational neuroscience, much work involves the use of computational models for the electrical response of neurons. In these models, it is assumed that the concentrations of sodium and potassium ions are constant during the course of electrical activity. However, this is not the case for calcium. Indeed, the calcium concentration within a cell is highly dynamic and is determined by a number of factors (1) the influx of calcium through VOCC (2) the release of calcium from calcium activated internal stores (e.g. Endoplasmic Reticulum, ER) (3) buffering by mobile and fixed buffers and (4) the extrusion of calcium by calcium pumps. The release of calcium from internal stores, e.g. endoplasmic reticulum, depends on reactions and diffusions which occur stochastically [32]. If the number of molecules within cell compartment is small, a deterministic simulation is not realistic. So depending upon the concentration of calcium ions an adaptive stochastic-deterministic solver can be used. *NTW* connected with NEURON could be a such adaptive solver.

The increasing scale and complexity of simulated models and morphologies have exceeded the capacity of a serial implementation. It is therefore natural to employ parallel simulation for such complex problems. In this thesis, we have developed an optimistic

MPI-based, parallel stochastic spatial reaction-diffusion simulator, Neuron Time Warp (*NTW*), to simulate models of calcium signaling pathways in a neuron.

Most existing calcium wave models are governed by differential equations. In this thesis we derived a stochastic discrete event calcium wave model from a mathematical model (which we used as a benchmark model for our experiments). The propagation of a calcium wave through a neuron leads to a very computational imbalance between the areas covered and uncovered by the wave at any given instant in time. Hence designing and developing a load balancing algorithm became an important part of this thesis. We used dynamic load balancing algorithms based on techniques from artificial intelligence. The introduction of such algorithms into *NTW* proved to enhance its performance.

The rest of the thesis is organized as follows: Chapter 2 describes the background and related work of this thesis. A detailed description of *NTW* is contained in chapter 3. The connection between *NTW* and NEURON is described in chapter 4. The derivation of a discrete event calcium wave model is described in chapter 5. A dynamic load balancing algorithms using reinforcement learning is described in chapter 6. Finally the last chapter contains the conclusion and thoughts on future research in this area.

CHAPTER 2
Background and Related Work

There are two types of algorithms which have been used for the stochastic simulation of neurons, particle-based and lattice-based [7]. In particle based methods, the state of the system is the number and location of particles in the sub-cellular space. The location of a particle and the system time are governed by probability distributions (e.g. [7]). Particles engage in a reaction when they are close enough. MCell [14], Smoldyn [15] and CDS [33] are particle based simulators. In lattice-based methods, the sub-cellular space is subdivided into lattice points or voxels using a mesh generation algorithm, and molecules are represented within each voxel. *NSM* is a lattice-based algorithm, in which space is partitioned into mesh grids called sub-volumes (i.e. voxels). Reactions can happen between molecules in the same grid and molecules can diffuse to adjacent grids. As we employed the next sub-volume method in which reactions between adjacent grids are ignored, we also ignored reactions between adjacent grids in *NTW*. NeuroRD [7] is a spatial extension of the Gillespie's tau-leap algorithm. This tool develops sequential version, while our intention is to produce a parallel algorithm capable of large scale simulations.

2.1 PDES for a Stochastic Reaction Diffusion System

The sequential stochastic discrete event simulation of large scale reaction-diffusion system is very slow. However in the era of high performance computation, parallel discrete event simulation has the ability to overcome this performance bottleneck [19].

In order to evaluate the parallel execution of the *NSM* algorithm, [25] uses two Time Warp based approaches. Both represent sub-volumes by *LP*s and messages between sub-volumes contain the diffusion events. One of the simulators makes use of grid computing. Preliminary results on small models were encouraging. As pointed out by the authors, a number of areas including window management and state saving remain to be investigated.

[2] points out 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 obtained for a predator-prey model (Lotka-Volterra). The performance of the simulation was hampered by a lack of control over the window size.

[19], [34] investigate the performance of optimistic synchronization algorithms in simulations of a reaction-diffusion system based on Gillespie's SSA [8]. They present a variant of the Next Sub-volume Method called the Abstract Next Sub-volume Method (ANSM). Three optimistic synchronization algorithms were employed - Time Warp(TW) [18], an optimistic approach with risk-free message sending called Breathing Time Bucket (BTB) [35] and a hybrid approach combined these approaches. [19] and [34] were also used for the simulation of a predator-prey model. None of the approaches employed a 3D grid geometry.

[36] developed the Lattice Microbes software to efficiently sample trajectories from either the Chemical Master Equation (CME) and Reaction-Diffusion Master Equation (RDME) on a high performance computing infrastructure (workstation containing a Supermicro X9DRG-QF motherboard with dual Xeon E5-2640 CPUs and four GTX680 GPUs using NVIDIA drive), taking advantage of GPUs to increase performance. They employed

a multiple thread approach to get efficient shared-memory communication between host threads, and avoided high GPU context switching overheads that would otherwise occur for multi-process access approach (i.e. with MPI). They considered uniform sub-volumes with some spacing and employed a multi-particle diffusion method. They used a dynamic load balancing algorithm to deal with inhomogeneous workloads. Employing a very small model (four reactions and 4 species), two and four GPUs provide a speedup over the single device, however for smaller volumes the benefit of four GPUs is correspondingly smaller. Eight GPUs did not provide any speedup when the sub-volume size was smaller than 64 cubic microns.

None of those above mentioned approaches employed a real neuronal geometry. They did not employ a real model (e.g. calcium buffering, calcium wave model etc.) as a benchmark to verify their simulators. We use models derived from a NEURON model to verify *NTW*'s correctness and to examine its performance.

2.2 Discrete Event Calcium Wave Model

In this thesis, a discrete event calcium wave model is derived from a deterministic calcium wave model [37] using the stochastic IP_3R channel structure. [38] uses a stochastic calcium wave model in which a reaction-diffusion system is modeled using partial differential equations. The channel dynamics of endoplasmic reticulum involving the inositol 1,4,5-trisphosphate receptors (IP_3R), are modeled stochastically.

Models of IP_3R play a central role not only for understanding channel kinetics, but also as building blocks for constructing larger scale models of cellular Ca^{2+} signaling. [39] developed a stochastic model of IP_3R in Xenopus oocytes and used stochastic simulation to predict the behavior of individual channel. They assumed that each IP_3R channel

consists of four identical and independent subunits and that each subunit consists of two independent Ca^{2+} binding sites. One site is an activating/inhibitory binding site while one is an IP_3 binding site. The channel is open when at least three subunits are in active state. We follow the same IP_3R channel dynamics specified in [39] to derive a discrete event calcium wave model.

2.3 Dynamic Load Balancing and Time Window

Load balancing algorithms could be static or dynamic, depending upon the rules they follow. In static algorithms [40], [41], [42] the tasks are assigned to a processor according to pre-specified rules and the assigned work load or tasks do not change during the simulation. The decision regarding the transfer of the task does not depend on the current state of the simulation. On the other hand, dynamic algorithms [43], [44] distribute tasks using the current state information of the simulation. If the workload of a processor becomes heavy then the tasks received at the processor are transferred to another processor which is not loaded so heavily.

[45] presents two algorithms for dynamic load balancing in a distributed computer system to reduce the difference of workload among the processors. Both algorithms are based on the load information of neighbor processors and they need considerable memory space for maintenance of the routes and load table. Those approaches more efficient in local area networks.

[46] presents a dynamic load balancing algorithm for PDES of spatially explicit problems. They used a virtual ring of processes with a constraint of one round of communications and nearest-neighbor communication. This approach is efficient when the problem

size is large (i.e. large space LP). The cost of load movement with small space LPs exceeds the benefits of an even load distribution.

[47] describes a load balancing algorithm for Clustered Time Warp [26]. The objective of the algorithm is to balance the computational load among the processors while minimizing the communications load. The computational load of a cluster is defined as the number of processed events since the previous load balance. The computational load of a processor is the sum of the loads of all of the clusters within that processor. Load balancing is performed by transferring clusters from overloaded processors to under-loaded ones. The algorithm iteratively transfers half of the difference in load from the most loaded processor to the least loaded one. The change in inter-processor communication is estimated and the cluster with the lowest inter-processor communication cost is selected for the actual transfer. This algorithm is repeated iteratively until all the processors have approximately the same load [47].

Rollbacks have a very strong influence on Time Warp's performance. An approach to controlling the number of rollbacks is to limit the optimism by allowing only those events whose timestamps are within a certain time window to be executed optimistically [48]. The time window is defined by the interval $[GVT, GVT + W]$, where W is the size of the window. Events within this interval can be executed, but those which have a time stamp beyond $(GVT + W)$ are not allowed to be executed, i.e. the LP is blocked. A blocked LP can still receive messages, but cannot send messages except for messages involved in the GVT computation. After a GVT update, the window itself is updated. Previously blocked LPs are unblocked if their next scheduled event falls within the newly updated window.

A load balancing/window control algorithm is described in [49]. In *NTW*, we made use of techniques from artificial intelligence to build these algorithms similar to the ones described in [49]. We combined the load-balancing algorithms with a window control algorithm.

CHAPTER 3
Neuron Time Warp

In *NTW* we divide the portion of the neuron being simulated into sub-volumes (*SV*s), each of which is assigned to an LP. Interactions between the *SV*s consist of events being sent between neighboring sub-volumes.

3.1 Introduction

As previously mentioned, [27] describes an event scheduling (XEQ) algorithm and a rollback message mechanism (rb-message). XEQ has an O(1) cost, while rb-message eliminates the computing cost of anti-messages and reduce the memory cost by eliminating the output queue at each SV. These two ideas are incorporated in NTW.

NTW [50] is an object oriented simulation environment which utilizes the *NSM* algorithm and makes the following assumptions:

1. SVs are grouped together to form a cluster. There is no limitation on the number of clusters in a system. However the only restriction is that each cluster can be mapped onto one processor - it cannot be divided between processors.
2. In each cluster, the reaction and diffusion events are arranged in chronological order.
3. Within a cluster, *SV*s receive messages in chronological order.
4. Each *SV* is a cube and is therefore connected to maximum of six neighbor *SV*s.
5. The topology of the *SV*s is static during the simulation.

In NTW, clusters are distributed among computational units (e.g. logical processors or cores). We currently allocate one cluster per core and each core runs an MPI process.

Each cluster processes the events belonging to its LPs in increasing time-stamped order. There is a special cluster, called a controller which is in charge of distributing the simulation workload, computing the GVT, and collecting the simulation results. We employ Mattern's algorithm [51] to calculate the GVT. *NTW* employs local rollback and local checkpointing [26] upon the arrival of a diffusion event at a cluster. A history queue, used to record scheduling history has been added to minimize the cost of scheduling rb-messages.

3.1.1 Architecture

The architecture and communication overview of *NTW* is displayed in figure 3–1.

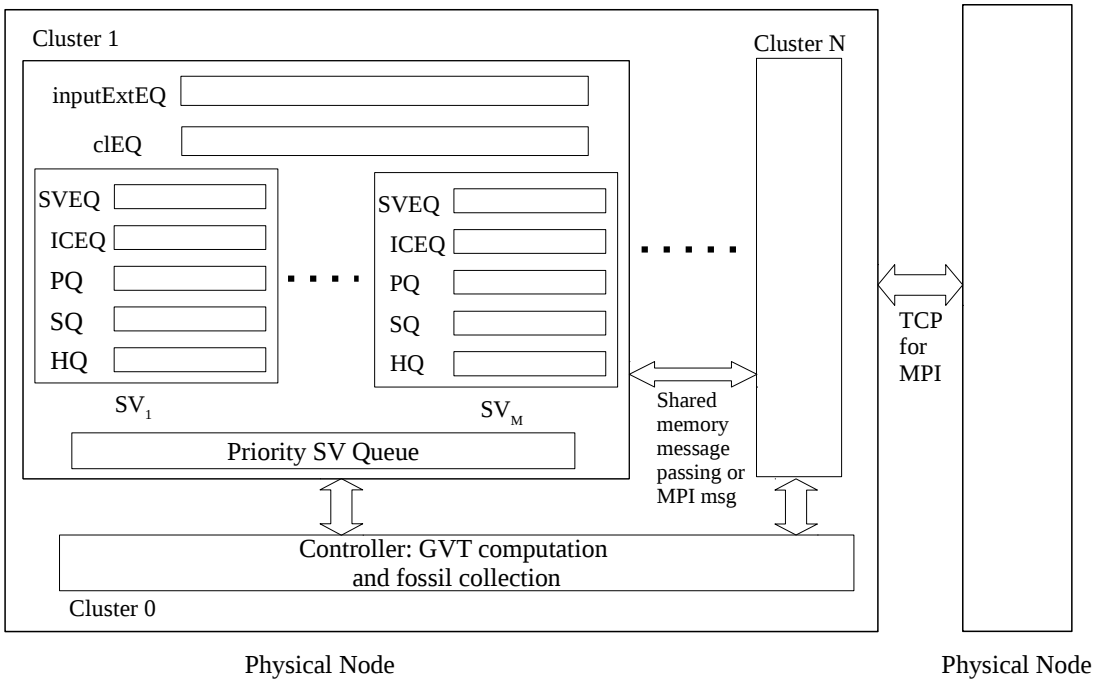


Figure 3–1: Architecture and communication overview of *NTW*.

In every cluster there are two event queues, the *clEQ* and the *inputExtEQ*. The *clEQ* is used to sort the events generated by the LPs in the same cluster. Its top event is the

lowest time stamped event of the cluster. All events from different clusters are put into $inputExtEQ$ and then forwarded to a destination input channel event queue, $ICEQ$. The priority SV queue is a container of SVs. The head of the SV queue contains the SV with the minimal local time, Lt, of next event in the cluster.

In a three-dimensional geometry, an SV can have at most six adjacent SVs. Each SV has an $SVEQ$ which is used to find the smallest time-stamped event of the SV, several input channels $ICEQ$'s which are used to hold diffusion events from neighbors, a processed event queue PEQ, a state saving queue SQ and a history queue (HQ).

The multi-level event queue structure is composed of three parts:

1. Each input channel can have one unique incoming source i.e. there are six $ICEQ$s for six neighbors of an SV.
2. At the SV level, $SVEQ$ which is a priority event queue. The top is the lowest event of the $ICEQ$s.
3. At the cluster level, the priority event queue is $clEQ$. The top of $clEQ$ contains the lowest time stamped event of the cluster.

There is a pointer at each input channel called current input event (CIE) which points to the event which is de-queued from its $ICEQ$ and is stored in the $SVEQ$. There is also a pointer at each SV current sub-volume event (CSE) which points to the event which is de-queued from its $SVEQ$ and is stored in the $clEQ$. These two pointers are used when rollback happens. Note that :

1. An $ICEQ$ can submit one event to its corresponding SV's, $SVEQ$, if and only if $ICEQ$ is not empty. The pointer value of the event is assigned to CIE.

2. An SV can submit only one event to its corresponding cluster's $clEQ$ if and only if $SVEQ$ is not empty. The pointer value of the event is assigned to CSE.

After an event is generated it will be sent to the corresponding input channel queue, $ICEQ$ and the event is inserted into the tail of the $ICEQ$. If the $ICEQ$ is not empty, the smallest timestamped event is submitted to $SVEQ$. The cost of insertion into the $SVEQ$ is constant i.e. log(6). If $SVEQ$ is not empty, the top of $SVEQ$ that is the smallest time stamp event of the SV will be submitted to the $clEQ$. Here the insertion cost is log(n). Here n is the number of SVs belongs to the cluster that is fixed up at the beginning of the simulation. Thus the cost of event scheduling is constant for the whole simulation. Using rb-messages instead of anti-messages in XTW removes the need to employ an output queue. Whenever a straggler arrives, an rb-message is send to other LPs. This message will cancel all the messages which have a time stamp larger than the time stamp of itself.

The HQ can be viewed as an array of its neighbors, in which each element stores the time stamp of the last event sent to the corresponding neighbor. The HQ is used when the SV needs to be rolled back. For example, suppose that the local virtual time of SV_i is 100, its $HQ[0]$ is 80 and $HQ[1]$ is 90. This indicates that it did not schedule any event to its first neighbor after 80 and second neighbor after 90. Assume that SV_i receives a straggler and needs to rollback to a checkpoint at 88. After retrieving the local virtual time and recovering the state, it needs to send rb-messages to the neighbor SVs to nullify the events which time stamps are greater than 88. It is easy to assert that a rollback-message i.e rb-message should be sent to the neighbor defined by $HQ[1]$ whereas we do not need to send an rb-message to the neighbor defined by $HQ[0]$. In this way, both the communication during the simulation and the cost for rolling back can be reduced.

3.1.2 Event Flow

In *NTW*, the event set consists of *rd-event*, *diffuse event* and *rb-message*. When an LP receives an event message, it checks either it is normal message (i.e. rd-event or diffuse event) or not (i.e. rb-event message). If the time stamp of any arrival event message is less than LP's local time i.e. straggler, then the LP needs to go through rollback process otherwise put into input channel queue (for normal message) or remove events from input channel queue which time stamps are greater than the arrival rb-message's time stamp. The steps are depicted in figure 3–2. When a cluster receives a diffuse event from an LP located in another cluster, it is buffered into inputExtEQ and then forwarded into the *ICEQ* of its destination LP.

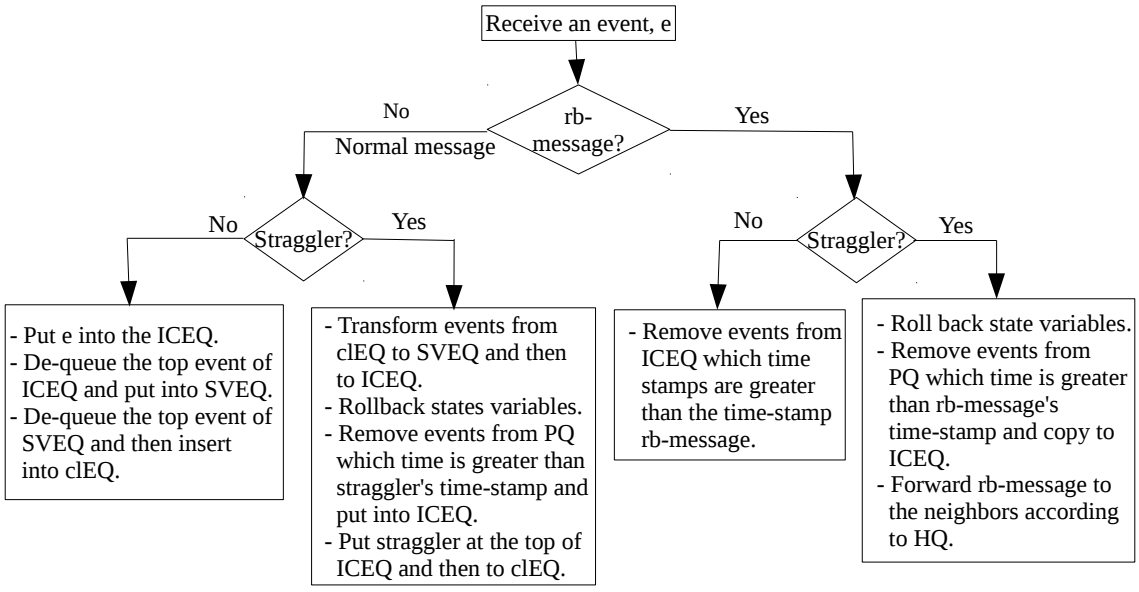


Figure 3–2: LP level event receiving.

The steps for processing events in *SV*s are determined by *NSM*. In every cluster, the *SV*'s are kept sorted in a binary heap such that the *SV* for which the first event occurs is at

the top. The rd-event is scheduled by the top *SV* of priority event queue and the diffusion event is scheduled by a neighboring *SV* which is responsible for the diffusion event, see figure 3–3.

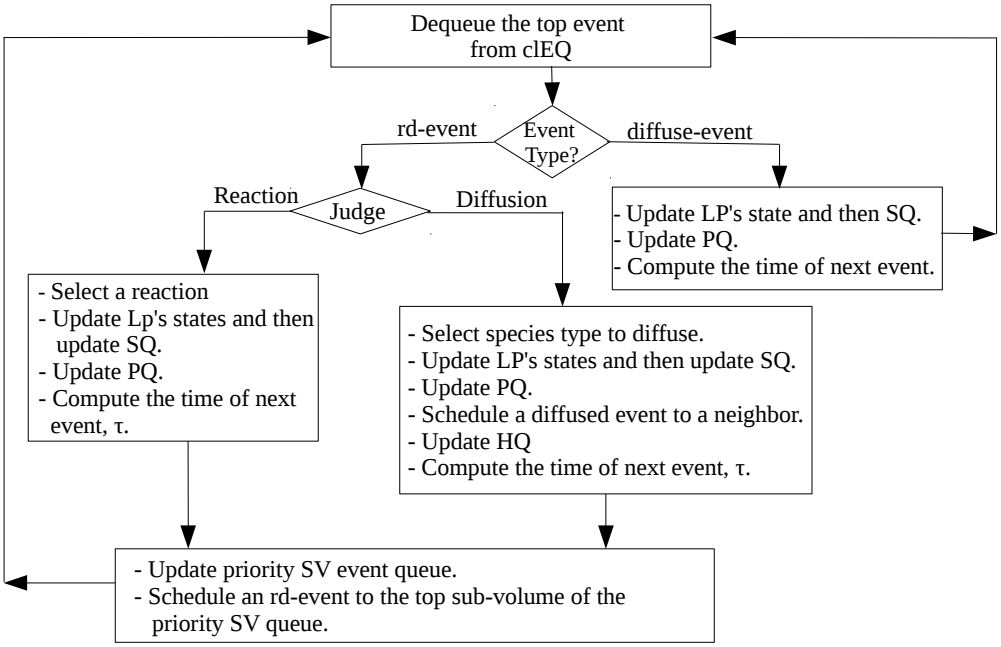


Figure 3–3: Cluster level event processing.

3.2 Shared Memory Communication

We call other *SV*s located in the same machine a family and give every *SV* a buffer to receive messages from its family. A semaphore controls access to the buffer. For example, if SV_0 is about to send data to its (family) SV_2, SV_0 needs to hold the semaphore for SV_2, then find room to write the data, after which it releases the semaphore when writing is finished. When receiving data, SV_0 must first hold the semaphore and then receive the data, after which it releases the semaphore.

3.3 Experimental Work and Analysis

The front end for *NTW* makes it possible for a group of chemical reactions to be input by the user and to be used in the simulation. This enables the user to experiment with different reactions, different concentrations of the molecules and ions involved in the reactions and their associated reaction rates. It is also possible to experiment with different diffusion rates into between adjacent cells.

Our experiments made use of the Lotka-Volterra model [52] to represent chemical reactions and the diffusion of ions taking place on a dendritic branch. The reasons we selected the LV model are that (1) it is simple and we can therefore focus on the simulation and that (2) the model is well known so we can verify our experimental results easily.

Spatial Lotka-Volterra model: We model a neurite branch as a 3D geometry of 2766 cubical sub-volumes. N_{prey} and $N_{predator}$ are the populations of the prey and the predators in a single sub-volume. Initially they are set to 100 in each sub-volume of the domain. C_i stands for the reaction constants.

$$r_0 : Prey \xrightarrow{C_1} 2 \times Prey$$
$$r_1 : Predator + Prey \xrightarrow{C_2} 2 \times Predator$$
$$r_2 : Predator \xrightarrow{C_3} Null$$

$C_1 = 10$, $C_2 = 0.09$ and $C_3 = 10$ are the reaction rate constants for reactions r_1, r_2 and r_3, respectively. $D_{prey} = 7.5\ \mu m^2/ms$ and $D_{predator} = 5\ \mu m^2/ms$ are the diffusion rate constants for the prey and the predators. The reaction rate equations for all of the reactions are: $r_1 = C_1 * N_{prey}$, $r_2 = C_2 * N_{prey} * N_{predator}$ and $r_3 = C_3 * N_{predator}$. The diffusion rates for any sub-volume i are

$$S_i = n_i * \sum_{j=1}^{M} D_j * N_j$$

Here, n_i = number of directions to which molecules can diffuse, D_j = diffusion rate constant for any species(Prey or Predator) j and N_j = number of molecules of species j at any sub-volume i.

Simulation Domain: Because the cylinder and branches are basic shapes for modeling neurites in NEURON, we use a Y-shaped geometry to determine the connections of neighboring *SV*s. The Y-Shape geometry consists of three cylinders, each 10 microns long and 1 micron in diameter. Each sub-volume is 0.25x0.25x0.25 cubic microns in size, and there are 2766 sub-volumes in total. Each cluster involved in the simulation has the same number of *SV*s. A conceptual neuronal branch is shown in figure 3–4.

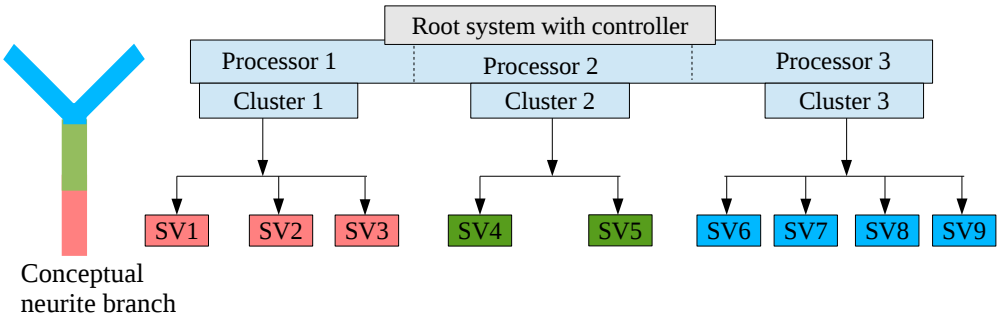


Figure 3–4: Conceptual Y-Shaped geometry is distributed to three clusters.

Hardware and Software Platform: The simulation runs have been executed on a Intel(R) Core(TM) i7-3770S CPU physical machine which is equipped with 4 physical cores (i.e. 8 logical processors) and 16GB RAM (Each physical core can execute 2 thread in parallel. So there are 8 logical processors in this machine.) The operating system on this machine is Ubuntu 12.04, with kernel 3.2.0-29-generic-pae. The MPI version is: (Open

MPI) 1.4.3 and gcc version is 4.6.4. *NTW* is implemented by using C++ programming language.

3.3.1 Oscillatory behavior of Lotka-Volterra System

The number of species is the state of the Lotka-volterra model. A portrayal of the interaction of the two species is contained in figure 3–5. In this figure we plot the population of the prey and the predator for a period of time during which 24,000,000 events were processed. We employ one controller and three workers and use the same initial conditions for all of the sub-volumes. We recorded the state of each sub-volume every 1.0 virtual time units. [19] observed the pronounced oscillatory behavior of the Lotka-Volterra System using the same spatial Lotka-Volterra model which we used here. We obtained the same behavior as mentioned in [19] and we can conclude our experimental results are faithful to Lotka-volterra model.

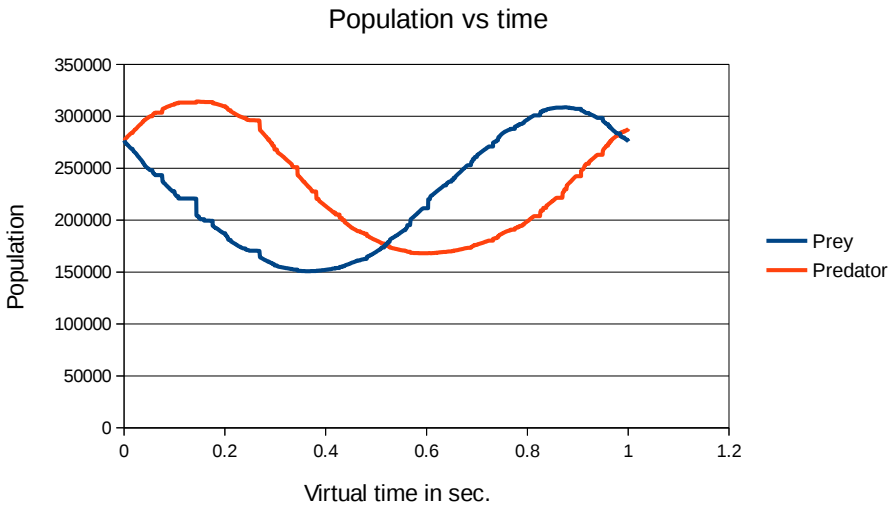


Figure 3–5: Population of predator and prey in time.

3.3.2 Performance

Lotka-volterra Model on Y-shape and Cuboid Geometry: We have two versions of our code, one of which employs MPI alone, while the other makes use of shared memory. The execution time and the number of rollbacks for both of these versions are portrayed in figures 3–6 and 3–7 on the Y-shaped geometry.

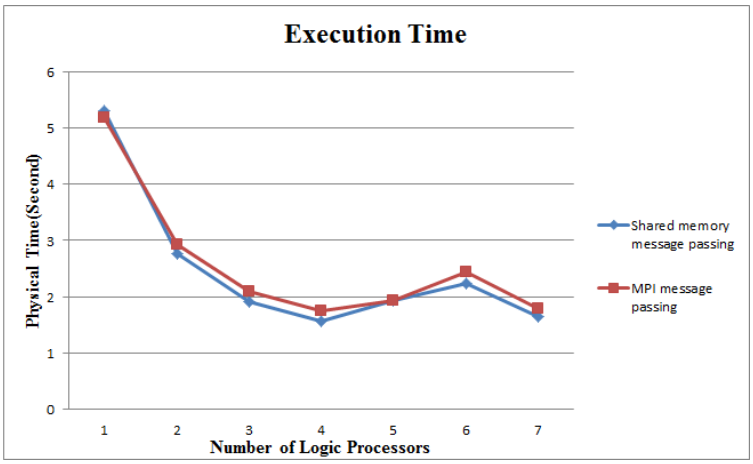


Figure 3–6: Execution time for Y-shaped model.

From these two figures we see that the shared memory version of the simulation time has a lower execution time then the MPI version (as expected). A maximum difference of 11% occurs at 6 processors. We also see that the MPI version has more rollbacks. The maximum percentage difference occurs with 6 processors, approximately 30%.

In order to determine if shared memory would further improve performance in a larger model we made use of a cuboid of dimension $8\times8\times100$ (thus 6400 *SV*s in total). The results are depicted in figures 3–8 and 3–9 in which we see the expected improvement. There is a 12% maximum difference in the execution time using 4 processors and a maximum difference of 79% in the number of rollbacks using 4 processors.

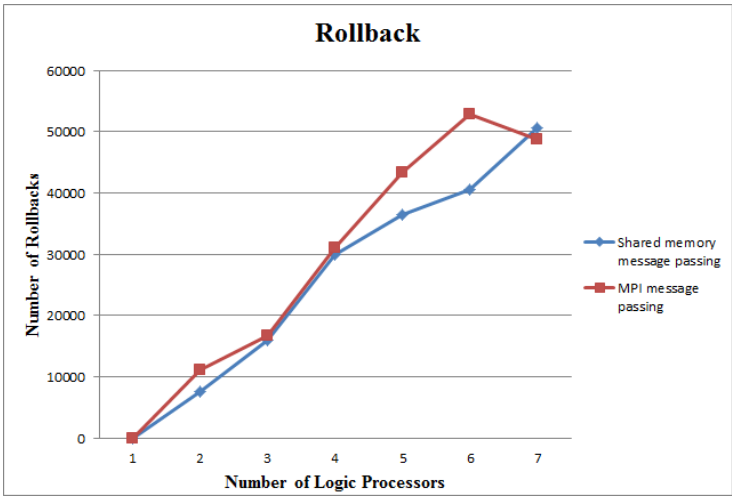


Figure 3–7: Rollbacks for Y-Shaped Model.

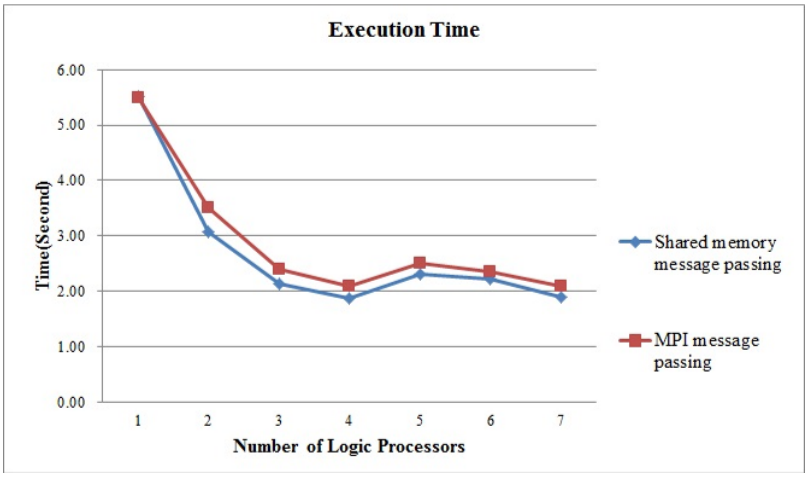


Figure 3–8: Execution time for cuboid model.

Diffusion rates affect the probability of a particular type of chemical reaction occurring, thereby altering the state of the system. This motivated us to experiment with

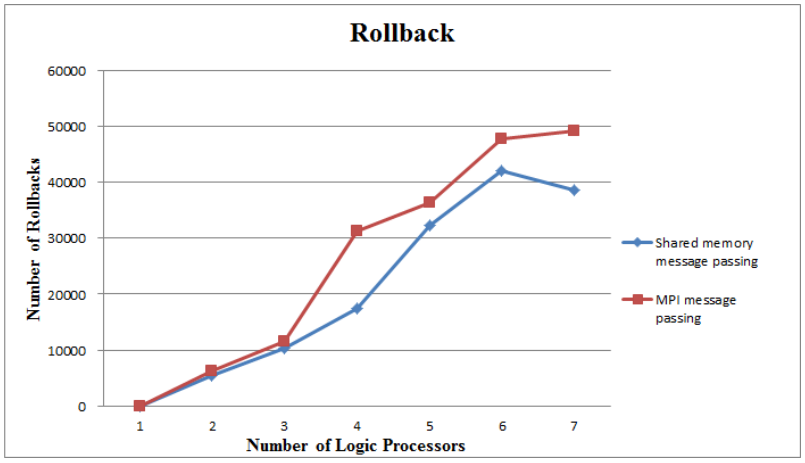


Figure 3–9: Rollbacks for cuboid model.

different values of the diffusion rate in the Y-Shaped geometry - we doubled the diffusion rate of both of the prey and the predator (Figures 3–10 and 3–11).

Figure 3–10: Execution time with doubled diffusion rates for Y-shape model.

A similar story plays out in these figures, with the maximum differences for the execution time and the number of rollbacks being 12% and 44% respectively.

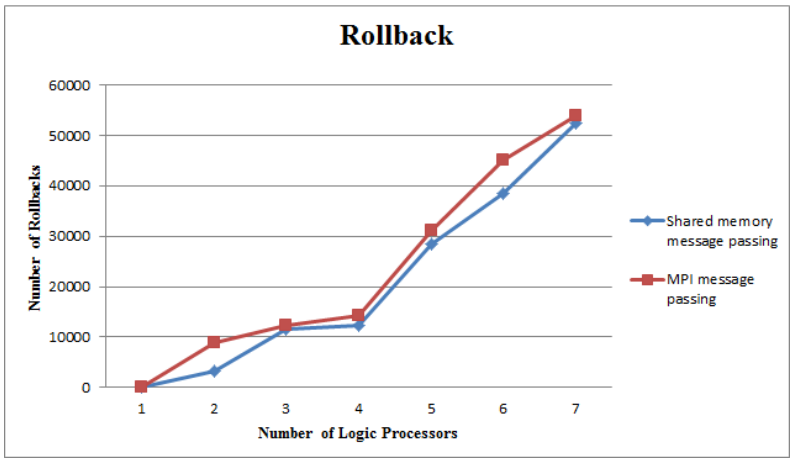


Figure 3–11: Rollback with doubled diffusion rates for Y-shape model.

These results indicate that better performance is achieved by utilizing shared memory. However, shared memory is not a panacea - the number of rollbacks increases for both approaches after achieving a minimal execution time. This in turn indicates the need for dynamic load balancing and dynamic window management.

CHAPTER 4
NEURON and NTW

Detailed simulation of chemical reactions and the diffusion of ions through a neuronal membrane presents challenges due to the multiple scales at which this occurs, scales that require development and consolidation of a number of different simulation methodologies. In the last chapter we described the *NTW*. In this chapter we describe NEURON simulator and our attempts to integrate *NTW* with NEURON.

4.1 NEURON

NEURON [28] is a simulation environment which was primarily developed by Michael Hines at Yale. In NEURON complex nerve models can be created by connecting multiple one-dimensional sections of neurons together to form arbitrary neuron morphologies. NEURON also allows for the insertion of membrane properties in these sections (including channels, synapses, and ionic concentrations). In NEURON, the ion channels of axons and soma are typically of the Hodgkin-Huxley type [29]. NEURON is very flexible and supports a wide class of models.

The computation executed by the nervous system involves the spread and interaction of electrical and chemical signals within and between neurons and glia cells (a type of cell which supports neuronal communication and also plays a vital role in the development of human intelligence). These signals can be modeled by the diffusion equation and the cable equation, partial differential equations in which the potential (voltage, concentration) and the flux (current, movement of a solute) are smooth functions of time and space. In

NEURON, these partial differential equations are solved numerically by converting them into difference equations.

A deterministic reaction-diffusion model is not accurate [4] when a small number of Ca^{2+} ions are involved (e.g. a neuronal spine, which is a small membranous bump on a dendrite). A sequential version of *NTW* is already integrated with NEURON. The experimental integration of parallel *NTW* with NEURON is also done as to check its compatibility with NEURON.

4.2 Connecting NTW to NEURON

Python has direct access to NEURON and can import NEURON's different computational modules by using the *import* statement. On the other hand, *NTW*'s code is in C++ and Python does not have direct access to C++ code. However, Python can use ctypes, a foreign function interface module (included in Python 2.5 and above) which allows for the loading of dynamic libraries and calls C functions. Hence Python can provide information such as the geometry (i.e. neighbor information for each sub-volume, species and reactions, sub-volume size, etc.) from the NEURON simulator to *NTW* through the C-C++ interface.

4.2.1 Sequential NTW integration

We developed a serial version of *NTW* with only one process (sequential *NTW*) and incorporated it with NEURON. The interaction between *NTW* and NEURON is shown in figure 4–1.

NTW is an individual module which interacts with NEURON using Python's ctypes interface. Prior to the simulation, pointers to all necessary NEURON's data structures are sent to *NTW*. As a result *NTW* can access NEURON's data structures and can update

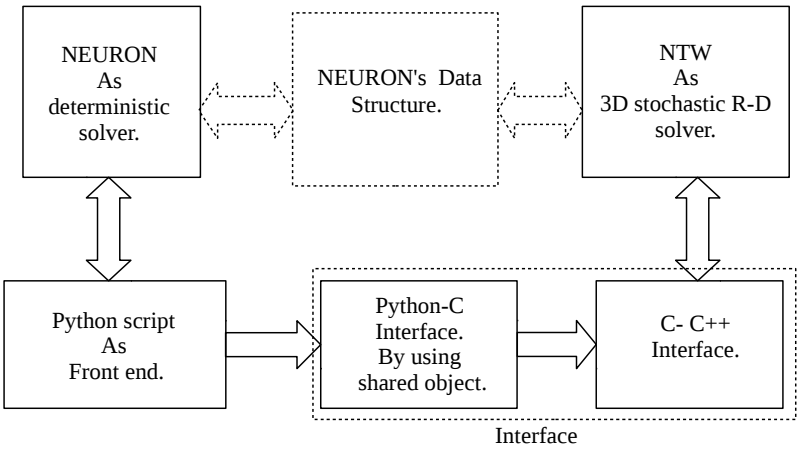


Figure 4–1: Typical interaction between NEURON and *NTW*.

NEURON's state. *NTW* can update NEURON states immediately when *NTW* processes an event during the simulation.

4.2.2 Parallel NTW integration

We have integrated the parallel version of *NTW* with NEURON in order to to check its compatibility with NEURON. We used a python script as a front end to the simulation and used ctypes as shown in figure 4–1. From this experiment we noticed that during the simulation *NTW*'s worker processes do not have any access to NEURON's states (i.e NEURON's data structures) directly and they can not update NEURON's state instantly when an event is processed. Only *NTW*'s controller has access to NEURON's states, but the controller does not process any events. A typical block diagram of parallel *NTW* with NEURON is also shown in figure 4–2.

In this experiment, we used a cylinder (10 microns in length and 2 microns in diameter) model with 2700 voxels (i.e. 2700 SVs each which is 0.25x0.25x0.25 cubic microns in size) and we also considered only one type of molecule, Ca^{2+} ions. Every voxel has 20

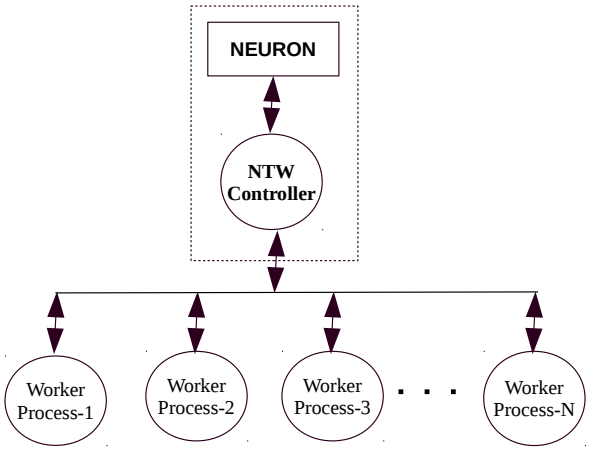
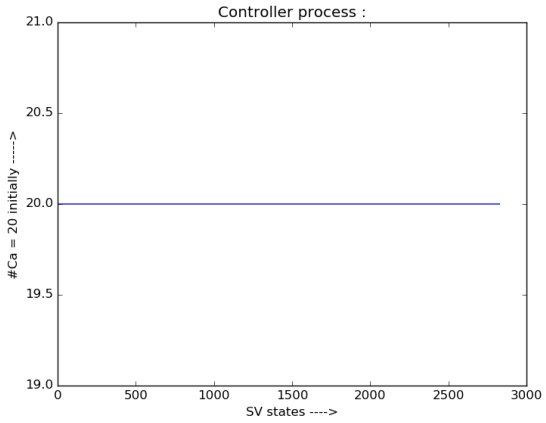


Figure 4–2: Typical block diagram of NEURON-NTW connection.

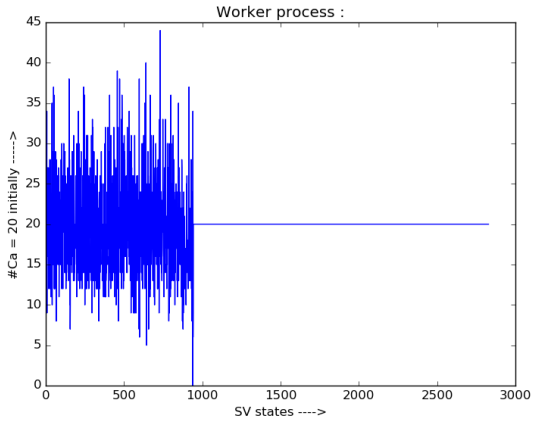
Ca^{2+} ions. In this case, the size of the NEURON's data structure is 2700. For simplicity we only consider diffusion events. *NTW* uses 3 worker process and one controller.

At the beginning of the simulation, the controller distributes 2700 sub-volumes among the worker processes evenly. It also provides a pointer to NEURON's states to all of worker processes. This pointer is copied to all worker processes. As a result, every worker process individually updates only its portion of copied states. After execution of 20,000 diffusion events the state of NEURON and *NTW* shown in figure 4–3.

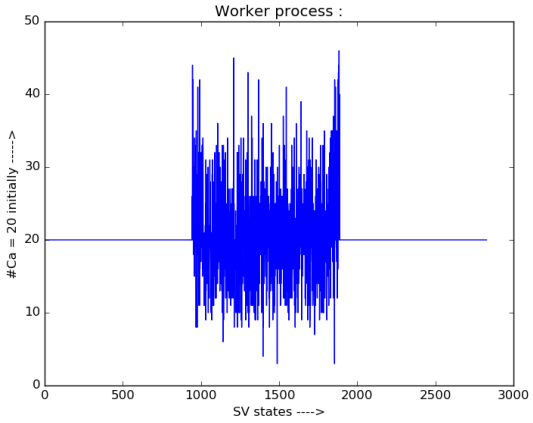
Transforming Safe States from NTW to NEURON: After every certain period of time (a cycle), NEURON's state is updated by the controller. In every cycle, all worker processes of *NTW* send their safe states to the controller. After receiving current safe states from all of the worker processes, the controller updates NEURON's states. Safe states means that the current updated states due to the execution of events which have smaller time stamp than the current GVT. In this experiment, after the execution of 20,000



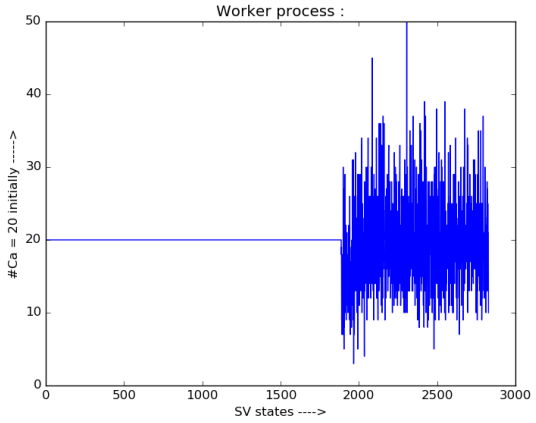
(a) Controller shows the NEURON's state (NEURON's state not updated yet here).



(b) Worker process 1 updates only first 1 to 900 sub-volumes (*SV*s).



(c) Worker process 2 updates 901 to 1800 *SV*s.



(d) Worker process 3 updates 1801 *SV*s to 2700 *SV*s.

Figure 4–3: State of NEURON and *NTW* workers.

events by each worker process, the controller collects safe states and updates NEURON's states as shown in figure 4–4.

NEURON state is updated by the controller after every C GVT cycle of the simulation. This might have a serious effect on the performance of a large simulation.

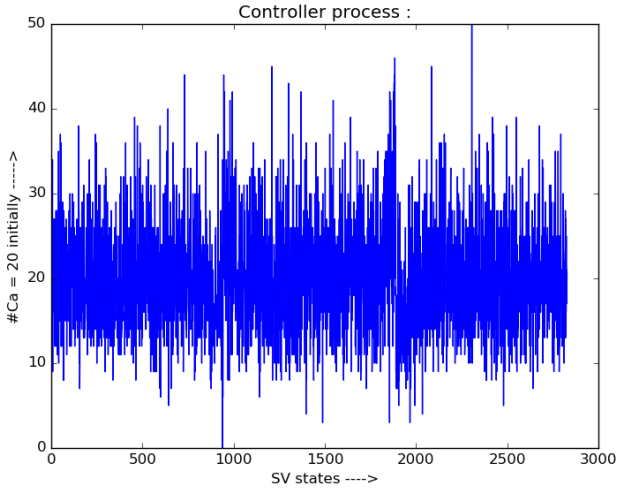


Figure 4–4: NEURON's state after completion of the simulation.

CHAPTER 5
Intracellular Ca^{2+} Signaling Pathways

As we mentioned before, the calcium ion, Ca^{2+}, plays a vital role in neuronal channel dynamics and ultimately in the behavior of the entire neuron. It is an important second messenger signal in many cell types, with diverse roles, from fertilization to regulating gene expression [37]. It is very important to neurons as it participates in the transmission of the depolarizing signal and contributes to synaptic activity [54]. Ca^{2+} signaling waves in neurons were discovered more recently and are not yet fully understood however they are thought to play an important role in synaptic transmission which is the form of secretion that leads to the release of neurotransmitters [54]. Nowadays, neuroscientists believe neuronal transmission has great impact in the process of learning and the formation and consolidation of memory [54].

Intracellular Ca^{2+} dynamics is controlled by the endoplasmic reticulum and the plasma membrane operating as a binary membrane system. These two membrane system interact with each other to control neuronal processes such as neurotransmitter release, associativity, plasticity and gene transcription. My research focus on the internal calcium dynamics of a neuron which depends on both biochemical reactions and diffusions.

In this chapter we describe the intracellular calcium signaling dynamics of a neuron. We simulate two models of calcium signaling pathways, a calcium buffer and a calcium wave model. The calcium buffer model is employed in order to verify the correctness and

performance of *NTW* by comparing it to a sequential deterministic simulation in NEURON. We also describe the newly derived discrete event calcium wave model (from a deterministic model) using the stochastic IP_3R channel model.

5.1 Neuron Basics

The neuron is the computational building block of the human brain. Each neuron receives inputs from thousands of other neurons via its dendrites and, in turn connects to thousands of other neurons via its axon. The point of contact between the axon and the dendrite of another neuron is called the synapse. The inter-cellular space between the presynaptic and postsynaptic neurons is called the synaptic space or synaptic cleft. When a synapse is activated by an electrical impulse from a presynaptic neuron, it releases chemical substances called neurotransmitters into the synaptic cleft which diffuse across the synaptic space to the postsynaptic neuron. The neurotransmitter molecules then bind to special receptors located on the membrane of the postsynaptic neuron. Receptors are membrane proteins which are able to bind to a chemical substance such as a neurotransmitter.

The membrane of a neuron is semi-permeable and has ion channels within it which control ion flow (including sodium, potassium, and calcium) between the exterior and the interior of the cell body. Movements of ions through these channels result from the diffusion of the ions down concentration gradients and the voltage difference created by different concentrations of these ions on the exterior and the interior of the membrane. Some of the channels are voltage gated; they are opened or closed by the electrical membrane potential.

5.2 Ca^{2+} Signaling Dynamics of Neuron

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 biochemical chain of reactions and/or diffusions inside the cell which in turn leads to a response.

These signaling pathways fall into two main groups depending on how they are activated. Many of them are activated by external stimuli. The cell responds to both intra- and extra-cellular cues, and can detect these through various signaling cascades wherein molecules react, diffuse, and/or are transported. Others respond to information generated from within the cell, usually in the form of metabolic messengers. In these signaling pathways information is conveyed either through protein-protein interactions or is transmitted via diffusing molecules which are referred to as second messengers. A second messenger is an intra-cellular substance, e.g. a calcium ion, that mediates cell activity by relaying a signal from an extracellular molecule, (e.g. a neurotransmitter) which is bound to the cell's surface. In this case the neurotransmitter is the first messenger. Neurotransmitters transmit signals across a synapse between two neurons. When a receptor on a post-synaptic neuron receives a neurotransmitter, it initiates intra-cellular signaling pathways. The role of Ca^{2+} signaling is very important to a neuron's response.

The basic mechanism of calcium signaling depends upon increases in the intra-cellular concentration of calcium ions. In most cells the concentration of intra-cellular calcium oscillates with a period ranging from a few seconds to a few minutes [57]. 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, as shown in figure 5–1. The ER acts as an internal warehouse of calcium ions from which calcium can exit to the cytosol through channels such as inositol triphosphate receptors, IP_3R, (e.g. it can also exit via RyR or leak). The IP_3Rs are located on the surface of the ER. Calcium ions, Ca^{2+}, can be pumped back from the cytosol to the ER by ATPase pumps.

When a postsynaptic neuron is electrically excited by receiving neurotransmitters at receptors, its voltage gated calcium channels (VGCC) located in the plasma membrane are opened and some calcium ions enter into the cytosol. This excitation also activates transmembrane proteins (a type of membrane protein spanning the width of the biological membrane to which it is permanently attached) to facilitate communication between cells by interacting with chemical messengers and G-proteins. G-proteins are a family of proteins involved in transmitting signals from stimuli coming from outside a cell to the inside of the cell. They function as molecular switches. The G-protein activates a Phospholipase C (PLC) enzyme and produces two second messengers, di-glyceride (DAG) which remain in the membrane and IP_3, which diffuses through the cytoplasm of the cell and binds to IP_3 receptors (IP_3R). When an IP_3R channel is triggered by both Ca^{2+} and IP_3, it is opened and allows the fast release of calcium from the ER to the cytoplasm, as shown in figure 5–1.

Typical calcium signaling pathways involve both binding and enzymatic reactions while molecules move through the intra-cellular space randomly. Hence binding and enzymatic reactions in combination with diffusion are the basic building blocks for modeling

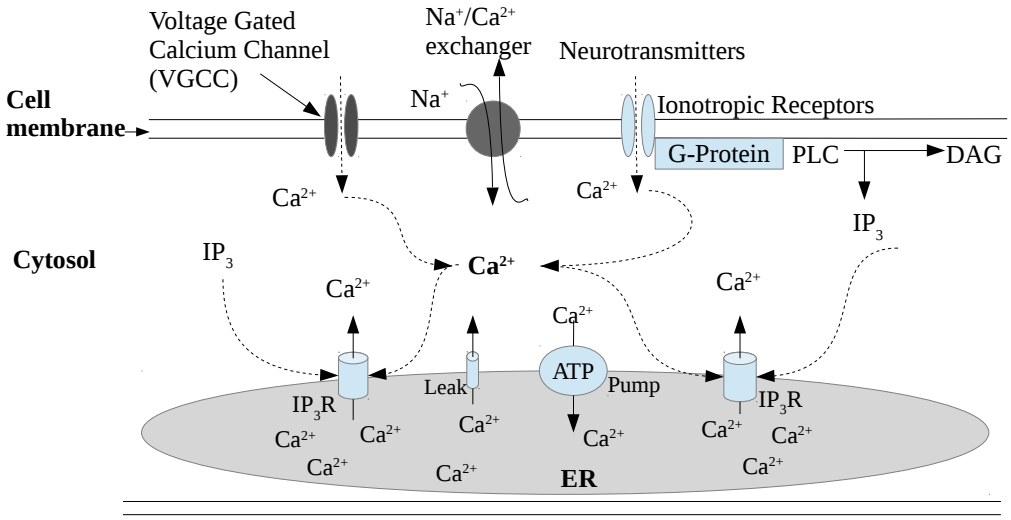


Figure 5–1: Calcium wave dynamics in neuron.

intra-cellular signaling pathways [4]. Recent intra-cellular calcium model consists of deterministic reaction-diffusion equations coupled to stochastic transitions of the calcium channels. The calcium and buffer concentrations in the cytosol are represented by partial differential equations (PDE). Stochastic quantities are the discrete states of channel units which determine the open/close state of a channel. The intra-cellular calcium concentration is determined by calcium diffusion, the transport of calcium ions through the ER membrane and the binding and unbinding of buffer molecules, mitochondria etc. The reaction terms, buffer binding and unbinding of calcium are modelled by the mass action kinetics in which the rate of a chemical reaction is proportional to the product of the concentrations of the reacting chemical species.

In a neuron calcium releases do not occur with a regular period, but are strongly influenced by stochastic process [53] [38]. The exact nature of these stochastic processes is not clear, but the most plausible explanation is that stochastic opening and closing of

the IP_3R and RyR are the most important influence causing irregularity in the calcium release. In [32], the calcium dynamics of a cell is formulated as a hybrid model in which the reaction-diffusion equations are used (a high concentration of calcium and buffers in the cytosol is assumed) and the opening/closing of channels is stochastic. In this paper we develop an IP_3R model and a stochastic simulation to predict the behavior of individual channels.

5.3 Experimental Work and Analysis

Our experiments made use of two models - a calcium buffer [4] on a dendritic branch derived from NEURON simulation environment and a newly derived discrete event calcium wave model on a cylinder of 3D grid of sub-volumes cubes (derived from NEURON simulation environment) and on an one dimensional geometry.

We make a comparison between a deterministic simulation in NEURON and a stochastic parallel simulation in NTW on a calcium buffer model to verify NTW's accuracy with respect to deterministic computation. We use the same reaction and diffusion rates, the same geometry (dendritic branch). We also employed the calcium buffer model to ascertain NTWs performance. We use a discrete event calcium wave model derived from a deterministic model [37] [38].

5.3.1 Calcium Buffer Model

Free calcium, Ca^{2+}, is buffered by intra-cellular buffers (e.g. calmodulin or parvalbumin). It can escape from these buffers, resulting in an almost constant concentrations of cytosolic calcium. This observation can be used to verify our simulator. The buffer model includes two reactions as follows:

$$r_0 : Ca + Buf \xrightarrow{C_1} CaBuf$$

$$r_1 : CaBuf \xrightarrow{C_2} Ca + Buf$$

Here, the reaction constant, C_1, for reaction r_0 is 0.01 $/ms$ and for reaction r_1 the reaction constant, C_2, is 0.01 $/ms$. The diffusion constant of Ca is 0.0001 $\mu m^2/ms$. Buf and $CaBuf$ are not mobile species i.e. diffusion constant for those are 0.

The cylinder and branch are basic shapes for modeling neurites in NEURON. We use a dendritic branch geometry, also referred to as Y shaped, which is taken from a NEURON model. The Y-Shape geometry consists of three cylinders, each 10 microns long and 1 micron in diameter. Total volume of 3 connected cylinders is about 23.56 μm^3 and whole volume is sub-divided into 2766 sub-volumes. All sub-volumes are considered same in size and it is $0.25 \times 0.25 \times 0.25$ cubic microns. Each cluster has near to the same number of SV's.

Hardware and Software Platform: The simulation platform was an Intel(R) Core(TM) i7-3770S CPU with 4 physical cores (8 logical processors) and 16GB RAM Each core can execute 2 threads in parallel. Hence there are 8 logical processors. The operating system is Ubuntu 12.04, with kernel 3.2.0-29-generic-pae. The MPI version is (Open MPI) 1.4.3, gcc version is 4.6.4. NTW is implemented by C++.

Concentrations were recorded for both the deterministic and parallel stochastic algorithms - see figure 5–2 for different concentrations. We employed a Y-shape geometry with the same reaction and diffusion rates and obtained almost the same behavior for both simulations. In the high concentration experiment, the initial concentrations in NEURON model for Ca, Buf and CaBuf were 8.0 μM, 4.0 μM and 0.0001 μM respectively. In order to verify the accuracy of the parallel simulation, we kept the same configurations. The

parallel version was run on four cores (one controller and three workers). The same experiment was done for low concentrations- 0.8 μM, 0.4 μM and 0.0001 μM for Ca, Buf and CaBuf respectively. The standard deviation plots for both the high and low concentration experiments are shown in figure 5–3.

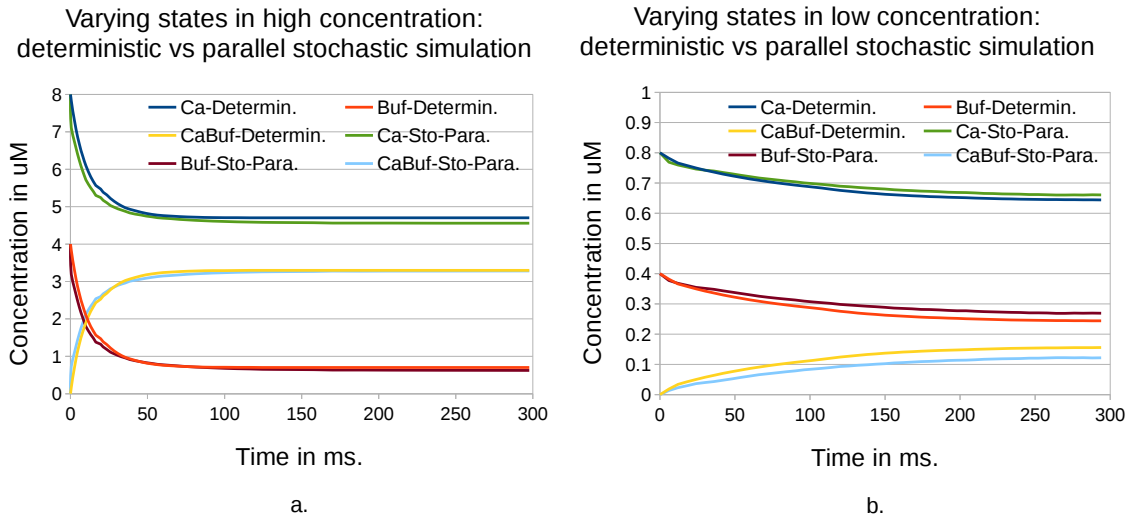


Figure 5–2: Varying concentration in deterministic and parallel stochastic simulation for a. high concentration b. low concentration.

To compare the results obtained by the *NTW* sequential (one controller with one worker) and parallel (one controller with several workers) stochastic simulations, we employed the calcium buffer model on the same Y-shape geometry. The *NTW* sequential and parallel (one controller with three workers) simulation with a standard deviation is shown in figure 5–4.

For all of the stochastic simulation experiments, the average of three parallel stochastic runs is considered. The standard deviation of three stochastic runs is shown in figure

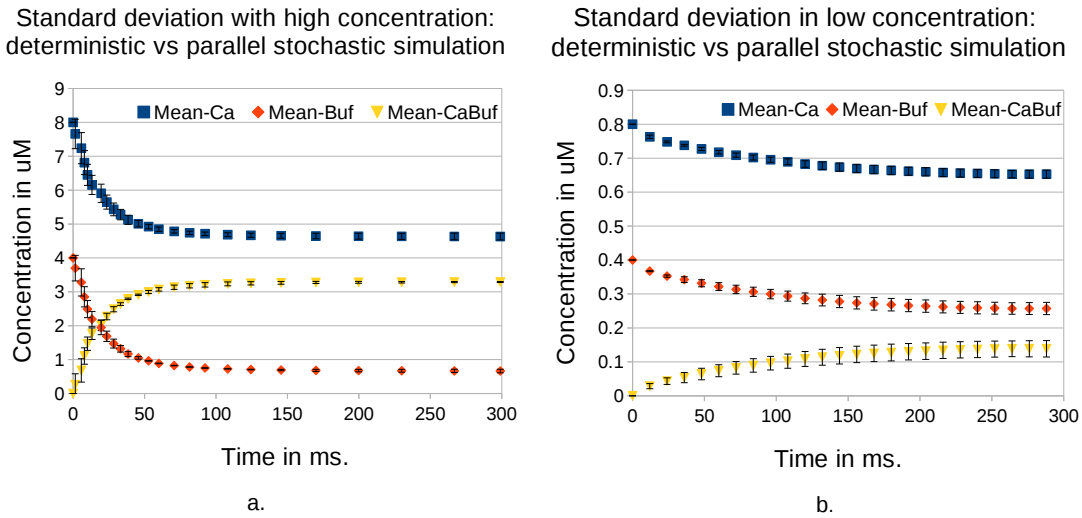


Figure 5–3: Standard deviation between deterministic and parallel stochastic for a. high concentration b. low concentration.

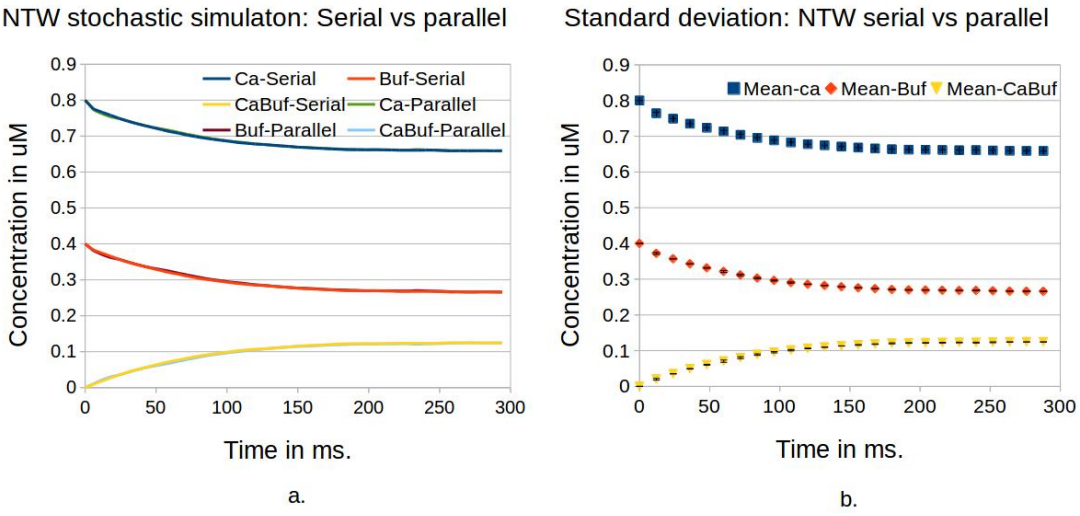


Figure 5–4: a. Varying concentration in *NTW* stochastic sequential and parallel simulation. b. Corresponding standard deviation plot.

5–5. The average standard deviation of the three stochastic runs (on 25 sample points) for Ca, Buf, and CaBuf are 0.01 μM, 0.006 μM and 0.095 μM respectively.

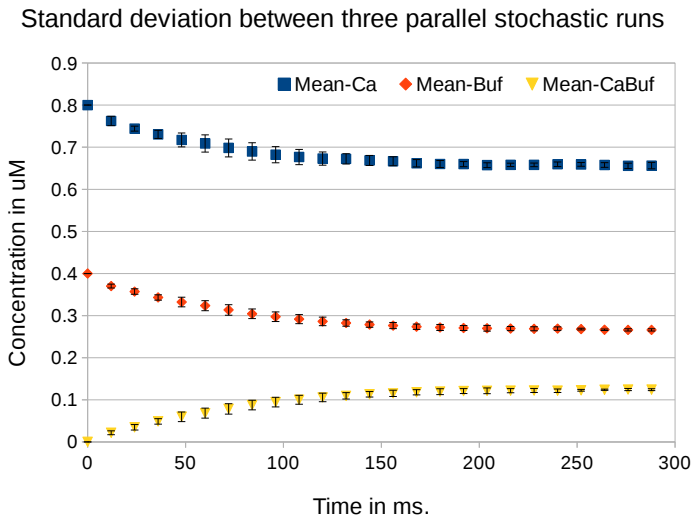


Figure 5–5: Standard deviation of three parallel stochastic runs.

Table 5–1: Performance table of calcium buffer model in parallel simulation.

Number of workers, N	Average physical time, T(N), in sec.	Speedup, S(N) = T(1)/T(N)
1	2.53	1
2	1.27	1.993
3	0.96	2.66
4	0.68	3.7
5	0.57	4.44
6	0.48	5.27
7	0.46	5.5

Performance: To measure the speed-up of the parallel simulator, we employed a calcium buffer model on a neuronal branch and ran the parallel simulation for 5.7 virtual time units during which 1.2 million events were processed. The simulation took 2.53 seconds for one worker and for two workers it required 1.27 sec, as shown in the table 5–1.

The execution time and speedup plot for the calcium buffer model on the Y-shape geometry are portrayed in figure 5–6 and figure 5–7 respectively. When a dendrite branch is distributed over multiple logical processors, communication latency is created and the execution time flattens out at 7 logical processors.

The number of rollbacks increases linearly when the number of worker process increases, the result of increased communication between the workers. With proper load balancing we can distribute the communication workload over the processors and decrease the number of rollbacks.

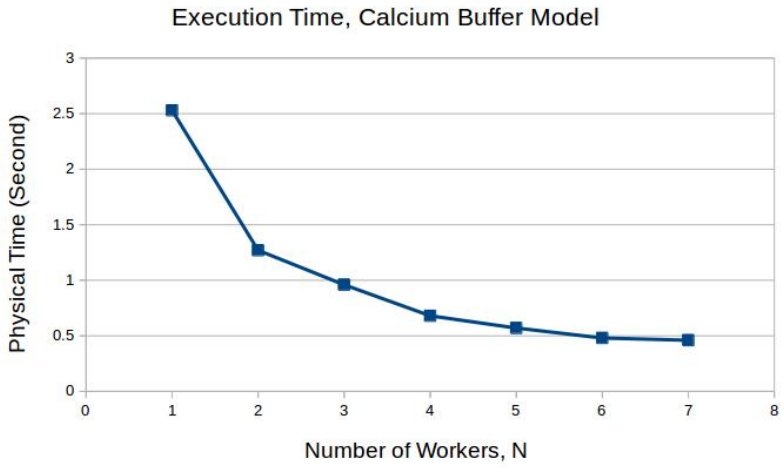


Figure 5–6: Execution time of calcium buffer model in parallel simulation.

5.3.2 Stochastic Discrete Event Calcium wave model: including Ca^{2+} activating and Ca^{2+} inhibiting sites dynamics

In the deterministic approach, the calcium concentration in the cytosol, $[Ca\_C]$, is calculated at every time step from different (in and out) fluxes- J_{IP_3R} , J_{SERCA}, and J_{LEAK} [37]. In the stochastic approach, different events (Channel Open, Release, Pump back and

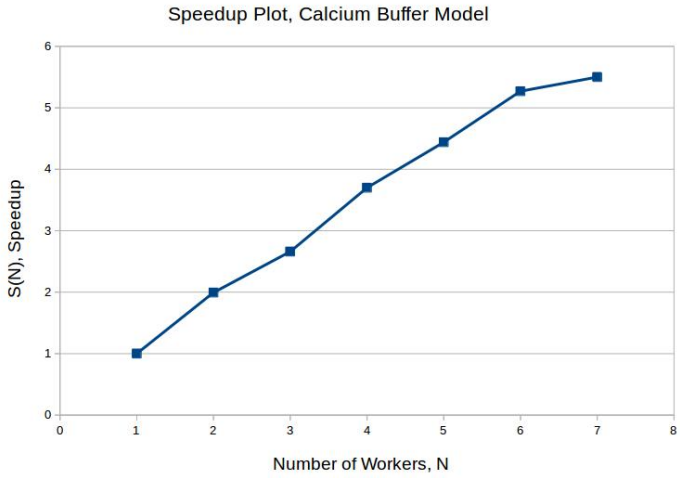


Figure 5–7: Speedup of NTW as a function of number of worker process.

Leak) at discrete points of time are considered. The event frequency can be controlled by their (in/out) flux rates, as derived in a mathematical model [37]. We have derived a stochastic reaction-diffusion discrete event calcium wave model from the deterministic model by considering only two bindings sites (activating and inhibiting Calcium sites). Because of the high occurrence of IP_3 binding and unbinding [38] ignored the IP_3 binding site in their IP_3R model and we do the same. Parameter values were obtained from [37] to produce a calcium wave propagation in an unbranched one dimensional geometry (length of 200 micron and diameter of 1 micron). We also observe Ca^{2+} wave propagation in a cylinder with a diameter of 1 micron and a length of 20 micron which consists of 3D grid of 1701 sub-volumes.

The stochastic channel dynamics of IP_3R: The IP_3R releases Ca^{2+} from the ER to the cytosol. It consists of four identical sub-units, each of which is composed of three binding sites [38]: an activating Ca^{2+} site, an inhibiting Ca site and an IP_3 binding site.

The three binding sites allow for eight different states X_{ijk} for each subunit. The index 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 activating site and 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 [38] ignored the IP_3 binding site. We also consider the same approach and assumed that the channel has a possibility of opening if the subunits is X_{10}.

The binding and dissociation of calcium Ca^{2+} at the activating and inhibition sites are stochastic events rendering the opening and closing of the channel a stochastic process. This stochastic process is coupled to the concentration of cytosolic calcium because the binding probabilities per unit time depend on it and the number of open channels determines the concentration.

The transitions corresponding to reactions follow:

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

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

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

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

$R_4 : Ca\_E \xrightarrow{rc_5} Ca\_C$

$R_5 : Ca\_C \xrightarrow{rc_6} Ca\_E$

$R_6 : Ca\_E \xrightarrow{rc_7} Ca\_C$

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 calcium binding and unbinding at the calcium activation site. Reaction R_4 is for calcium release from ER to cytosol. R_5 and R_6 are for the Sarco/Endoplasmic Reticulum Ca^{2+}-ATPase (SERCA) pump from cytosol to ER and leaking Ca^{2+} to cytosol, respectively. Calcium in the cytosol is Ca_C and calcium 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 Ca_E is not mobile, so its diffusion constant is, $D_{CE} = 0$.

In *NSM*, at every iteration, the reaction rates for all of the reactions are evaluated. In order to calculate a reaction rate, we follow the flux rate calculation of the deterministic model [37]. For 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 $rc_7 = 1.5$.

The calculation of *NSM* reaction rates are given below:

Reaction rate for reaction R_0 (to not inhibit IP_3R) and R_1 (to inhibit IP_3R Channel):

The rate of R_0 reaction i.e. Ca not inhibiting rate = $rc_1 * IP_3R_{Inhibit} * h_{inf}$.

The rate of R_1 reaction i.e. Ca inhibiting rate = $rc_2 * IP_3R_{Not\_Inhibit} * (1 - h_{inf})$.

Here, h_{inf} represents the fraction of inactivated IP_3R receptors by cytoplasmic calcium i.e Ca_C and it is defined here as follows: $h_{inf} = K_{inh}/(K_{inh} + [Ca\_C])$. Here, K_{inh} is a constant.

$[Ca\_C]$ is the local concentration of calcium in the cytosol. Initially, $IP_3R_{Inhibit} = 0$ and $IP_3R_{Not\_Inhibit} = 4$ (as for 4 sub-units) which causes R_1's reaction rate to be greater

than the R_0's reaction rate. That means initially that all of the channels are in an inhibited state i.e. X_{J1}. When the R_0's reaction rate is greater than R_1's rate the channel is in not in an inhibited state i.e. X_{J0}.

Reaction rate for reaction R_2 (to activate IP_3R) and R_3 (to inactivate IP_3R Channel):

The rate of R_2 reaction i.e. Ca activating rate = $rc_3 * IP_3R_{Not\_Active} * (1 - n_{inf})$.

The rate of R_3 reaction i.e. Ca inactivating rate = $rc_4 * IP_3R_{Active} * n_{inf}$.

Here, n_{inf} represents the fraction of activated IP_3R receptors by cytoplasmic calcium i.e Ca_C and it is defined here as follows: $n_{inf} = [Ca\_C]/(K_{Act} + [Ca\_C])$. Here, K_{Act} is a constant.

Initially, $IP_3R_{Not\_active} = 4$ and $IP_3R_{Active} = 0$ which causes R_2's reaction rate to be greater than R_0's reaction rate. So all of the channels are in inactivate state i.e. X_{0K}. When R_3's reaction rate becomes greater than R_2's reaction rate, the channel becomes active i.e. X_{1K}.

Reaction rate for reaction R_4 (to release Ca^{2+} from ER to cytosol):

The reaction rate of R_4, i.e. the Release rate = $rc_5 * (J_{IP_3R})$

Here, $J_{IP_3R} = V_{IP_3R} * x * n * m * h * ([Ca\_E] - [Ca\_C])$

Where,

$n = IP_3R_{Active}/(IP_3R_{Active} + IP_3R_{Not\_Active})$

$m = IP_3/(IP_3 + K_{IP_3})$

$h = IP_3R_{Not\_Inhibit}/(IP_3R_{Inhibit} + IP_3R_{Not\_Inhibit})$

$x = number\ of\ IP_3R\ channels\ per\ cluster.$

$Here\ x = 1\ i.e.\ one\ IP_3R\ channel\ per\ cluster.$

V_{IP_3R} and K_{IP_3R} are constants.

It is notable that release depends on m, n, and h.

- If IP_3 is not available i.e. $m = 0$ and release rate will be 0. This means release event will never happen.
- Similarly, h depends on $IP_3R_{Not\_Inhibit}$. When, $IP_3R_{Not\_Inhibit}$ is 0 then h is also 0, which means that the channel is in an inhibited state and the release event cannot occur. The state of $IP_3R_{Not\_Inhibit}$ or $IP_3R_{Inhibit}$ is managed by reaction R_0 and R_1.
- n depends on IP_3R_{Active}. When, IP_3R_{Active} is 0 then n is also 0. This means IP_3R is in an Ca inactivated state. The value of IP_3R_{Active} and $IP_3R_{Not\_Active}$ is always updated by reaction R_2 and R_3.

Reaction rate for reaction R_5 (to pump back Ca^2+ from cytosol to ER):

The reaction rate or pumping rate = $rc_4 * J_{SERCA}$

Here, $J_{SERCA} = V_{SERCA} * ([Ca\_C]^2/(K^2_{SERCA} + [Ca\_C]^2)$.

and V_{SERCA} and K_{SERCA} are constants.

Reaction rate for reaction R_6 (leak Ca from ER to cytosol):

The reaction rate i.e., the leak rate = $rc_5 * J_{LEAK}$.

Here, $J_{LEAK} = V_{Leak} * ([Ca\_E] - [Ca\_C])$.

and V_{Leak} is a constant.

All constants are taken from deterministic model [37] are shown in table 5–2.

The basics of Ca^{2+} wave propagation

Figure 5–8 shows a calcium wave through an array of sub-volumes, each of which has only one IP_3R receptor. When (1) there is a small number of Ca^{2+} and IP_3 ions are available (2) the calcium activated site being activated and no calcium inhibition, the IP_3R

Table 5–2: All constants for calcium wave model

V_{IP_3R}	0.0002
K_{Act}	0.0004
K_{IP_3}	0.0013
K_{inh}	0.0019
K_{SERCA}	0.0001
V_{SERCA}	.00003249
V_{Leak}	0.00003

channel releases Ca^{2+} from the ER to the cytosol, creating a small calcium wave called a 'blip'. Several blips in a IP_3R cluster creates a 'puff'. Because each cluster has only one IP_3R, we do not distinguish between blips and puffs in our experiments. Due to the puffs, the cytoplasmic calcium concentration is raised as shown in figure 5–8. The released calcium then diffuses to neighboring sub-volumes and activates nearby IP_3R via "calcium induced calcium release" (CICR). Thus a global propagation of a calcium wave is evoked, figure 5–8. In short, Ca^{2+} release at one IP_3R can trigger Ca^{2+} release at adjacent IP_3R via CICR, leading to the generation of Ca^{2+} waves.

Model verification and wave propagation In order to to observe wave propagation experimentally, we employed an array of two hundred 1x1x1 cubical micron *SV*s connected linearly as shown in figure 5–9. In every sub-volume we modeled the cytosolic and endoplasmic reticulum (ER) compartments by using a fractional volume for each, as shown in figure 1–5.

Initial concentration for both Ca^{2+} and IP_3 are 0.0033 μM (i.e. 2 Ca^{2+} molecules per *SV*) and 0.0 μM respectively. After a period of time, after one iteration of main loop, the concentration of IP_3 is increased to 0.332 μM (i.e. 200 IP_3 molecules per *SV*) in the middle six *SV*s (i.e. SV_{97} to SV_{102}) and observed the spreading of a Ca wave as shown in figure

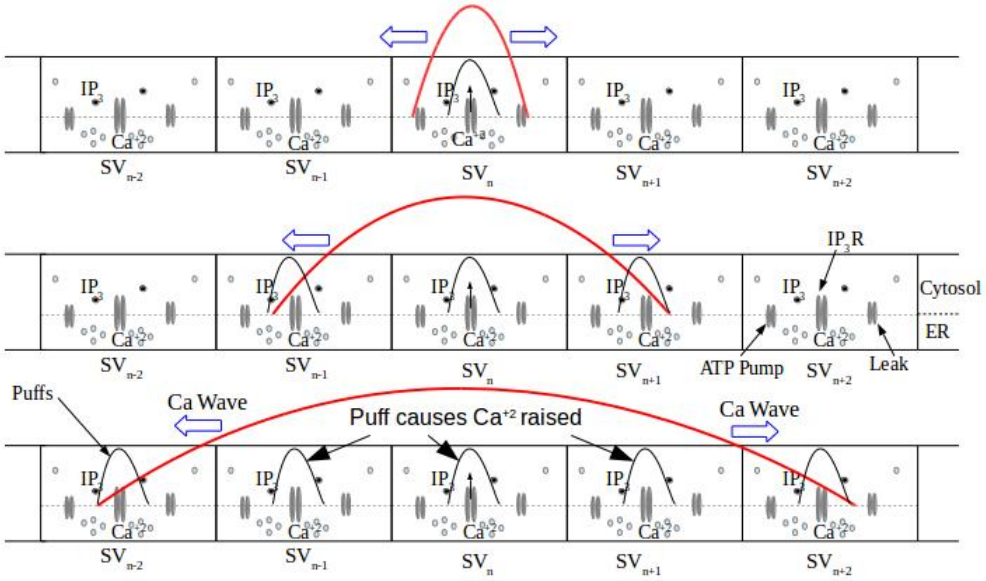


Figure 5–8: Calcium wave propagation through a linear geometry with one IP_3R channel per micron.

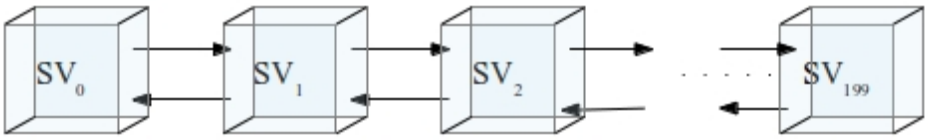


Figure 5–9: Linear connection of computational sub-volumes, SVs.

5–10. Similarly to observe the spreading of calcium wave through a 3D grid, we employed a cylinder with a diameter of 1 micron and a length of 20 micron which consists of 3D grid of 1701 0.25x0.25x0.25 cubical micron sub-volumes. Initial concentration for both Ca^{2+} and IP_3 are 0.0033 μM (i.e. 2 Ca^{2+} molecules per SV) and 0.0 μM respectively. After a period of time, after one iteration of main loop, the concentration of IP_3 is increased to 106.27 μM (i.e. 1000 IP_3 molecules per SV) in the middle nine SVs (i.e. SV_{845} to SV_{853}) and observed the spreading of calcium through a 3D grid cylinder as shown in figure 5–11.

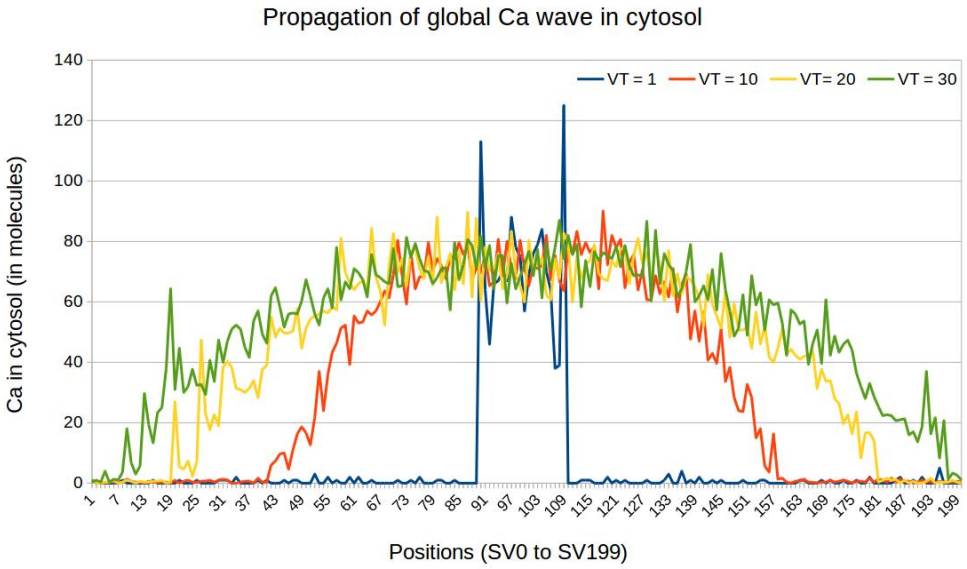


Figure 5–10: Observation of calcium wave propagation in linear geometry.

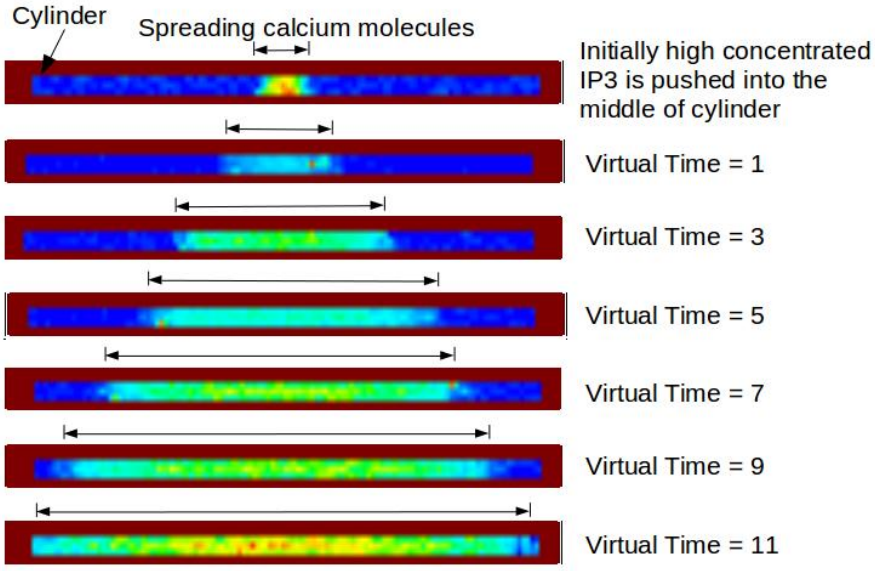


Figure 5–11: Observation of calcium wave propagation in a cylinder with a diameter of 1 micron and a length of 20 micron which consists of 3D grid of 1701 sub-volumes.

Because there is high concentration of IP_3 in SV_{97} to SV_{102}, the release event can only occur at those SVs. IP_3, and then spread gradually along the dendrite, resulting in a Ca^{2+} activation of the IP_3 receptors (IP_3Rs). Activation of IP_3Rs permit release of Ca^{2+} from the endoplasmic reticulum stores into the cytosol. This causes the concentration of calcium in cytosol, $[Ca\_C]$ to increase and to start to diffuse in both directions (left and right) from the middle and to affect the neighbor SVs as well. In addition the IP_3 also diffuses to neighbors and causes neighboring SVs to start to release events. Thus a global wave of calcium in the cytosol, Ca_C, starts to propagate in both directions with respect to time (1 ms to 30 ms) as shown in figure 5–10. The calcium wave is travelling very fast (about 3.3 micron per ms), which is far from a real biological neuron (1 micron every 20 ms). In fact we reproduced the qualitative dynamics and moved on because we are interested in the performance of NTW with our discrete model.

Steady state IP_3R channel dynamics:

We developed an IP_3R model and a stochastic simulation to predict the behavior of individual channels. The IP_3R model plays a central role not only for the understanding of channel kinetics but also as a building block for constructing larger scale models of cellular calcium signaling. Figure 5–12 describes the dependence of IP_3R channel open probability (P_o) as a function of cytosolic Ca^{2+} for different concentration of IP_3.

Experiment: Initially all channels of the domain (sub-volume SV_0 to SV_{199} each of which has only one IP_3R channel) are closed. The concentration of Ca^{2+} in cytosol is .0033 uM and the concentration of the IP_3 in cytosol is 0 μM. After a very short period of time, the concentration of IP_3 (for all SVs) rises to .0083 μM. This causes all of the IP_3R channels to open and increases the level of Ca^{2+} concentration in the cytosol very sharply.

When Ca^{2+} concentration in cytosol reaches into a certain point, here it is .035 μM, all channels starts to close and reach into a steady state.

Open probability (P_o) calculation: The *NSM* algorithm first calculates the total reaction and diffusion rates at every iteration. Then it decides whether the event is reaction or diffusion event randomly. If the event is a reaction, *NSM* selects a reaction out of the 7 reactions. The model consists of 7 reactions, R_0 to R_6, where R_4 is the open reaction i.e. release Ca^{2+} from ER to cytosol. The open probability P_o is defined as the ratio of the total number of R_4 reactions occurring (release events), N_{R4} to the total number of reaction events, N. $P_o = N_{R4}$/N. Figure 5–12 is the plot of P_o for different concentration levels of cytosolic calcium with 3 different concentrations of IP_3.

[39] experimentally observed that the channel open probability is a bell-shaped function of the concentrations of calcium in cytosol, $[Ca\_C]$. The open probability varies with respect to IP_3 concentrations. We also obtained the same behavior, validating our stochastic discrete event calcium wave model. It is notable that when calcium concentration is about .073 μM the open probability is very low (less than .02). In a stochastic simulation, the channel open probability P_o never goes to zero as in steady state some releases do occur due to the constant concentration of calcium in cytosol and IP_3.

Performance: The execution time of our model in the neuronal branch geometry is displayed below in figure 5–13. We first note that it takes 5 worker processes for the execution time of the model to be divided in half. In the Calcium buffer model it took 2 workers to achieve this.

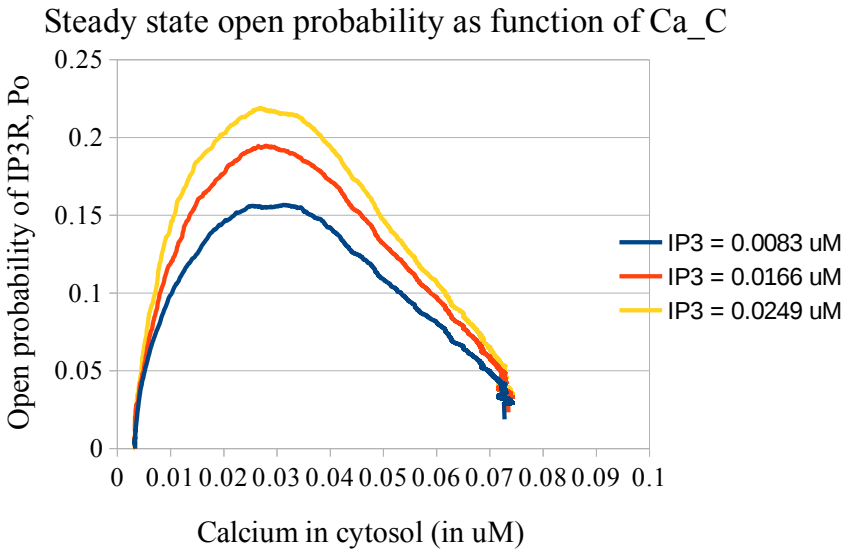


Figure 5–12: Steady state open probability as function of calcium in cytosol, *Ca_C*.

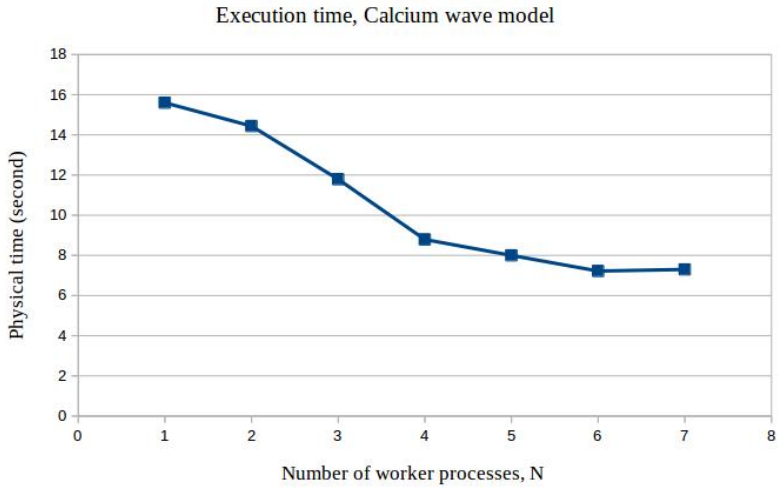


Figure 5–13: Execution time of calcium wave model in parallel simulation.

The reason for this has to do with the large computational imbalance between the areas in the model with high calcium (covered areas) and those with low calcium (uncovered areas). This imbalance results in a large number of rollbacks-they increase almost

Table 5–3: Experimental data of workloads of the covered and uncovered areas.

	Virtual Time = 4		Virtual Time = 8		Virtual Time = 12	
Description	Covered area	Uncovered area	Covered area	Uncovered area	Covered area	Uncovered area
Average number of processed events, NOE	5017.13	196.04	8668.65	380.4	12100.58	564.5
Percentage of event processing	96.10%	3.90%	95.62%	4.38%	95.34%	4.66%
Number of Sub-volumes	46	154	88	112	100	100

linearly with the number of process-see figure 5–14 and degrade the performance of the simulation.

Our experiments clearly revealed this computational imbalance. Table 5–3 shows experimental data comparing the workloads of the covered and uncovered areas.

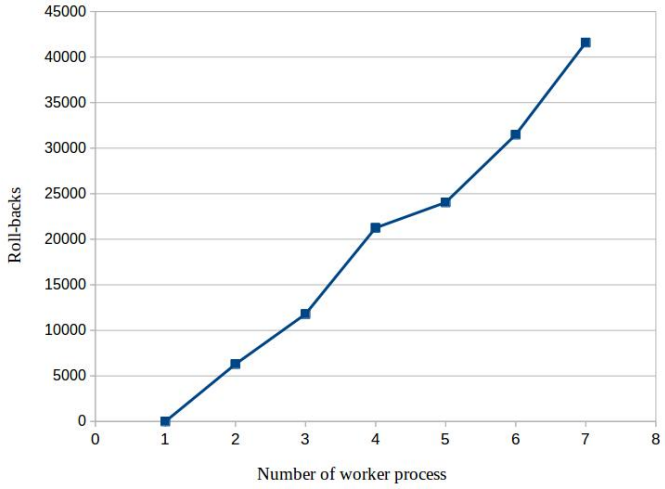


Figure 5–14: Rollback with multiple worker processes in calcium wave model simulation.

Some take-aways from these experiments

- 95% more events are processed in the covered area than in the uncovered area. Among the processed events in the covered area about 15% are reaction events and about 85% are diffusion events. We conclude that it is important to detect the covered area in order to determine the proper distribution of sub-volumes among the worker processes.
- The size of the covered area increases with time, so dynamic load balancing is worthwhile investigating.
- The rate of spreading of the covered area depends on the speed of calcium wave which in turn depends on the concentration of IP_3. If the concentration of IP_3 is high, the Ca wave spreads quickly, otherwise slowly [37]. Hence we can detect the covered area via the concentration of IP_3 in the SVs.

We intend to explore the utility of a load balancing algorithm for this problem.

CHAPTER 6
Dynamic Load Balancing using Reinforcement Learning

In the last chapter we noticed that a large computational imbalance occurs between the areas in the model with high calcium (covered areas) and those with low calcium (uncovered areas) in our calcium wave model. This imbalance results in a large number of rollbacks and degrades the performance of the simulation. In this chapter we introduce a dynamic load balance algorithm along with a dynamic time window in order to enhance the performance of calcium wave simulation.

6.1 Dynamic Load-balancing

The distribution of load among the processors has an important effect on the performance of a parallel program. In order to achieve the best performance all of the processors should have approximately the same load. In the calcium wave simulation when the calcium wave starts to spread from one processor to neighboring processors a load imbalance starts to occur.

A centralized algorithm is employed for dynamic load balancing in NTW. In this approach each processor sends its load and communication information to a central node called a controller. The controller node utilizes this information to select (and inform) those processors which participate in the algorithm.

6.1.1 General Structure of the Algorithm

We introduce two algorithms, the *computation* and *communication* algorithms, used to balance the computational and communication loads respectively. Recall that in our

algorithm each SV is represented by an LP. 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 events processed since the last load balance.

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 has 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 The final load of the processor. Its value is between 0 to 1.

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

$$PLoad = \lambda * PComp + (1-\lambda) * PComm \tag{6.1}$$

The load-balancing algorithm is initiated at every C GVT cycles, where C is a control parameter whose value is determined by a Q-learning algorithm. After C GVT computations we estimate the workload for each worker process and detect a load imbalance using runtime statistics. The idea of the algorithm is as follows: each processor (i.e. NTW's worker process) sends the values of *PComp* and *PComm* to the controller (i.e. NTW's controller process). The controller checks the loads of the worker processes to detect an imbalance. If there is an imbalance, the controller selects the over and under-loaded worker processes and sends a message to the over-loaded process. The maximum loaded process then selects up to L of its LPs and sends them to the minimum loaded process. To select the value of L the controller uses a q-learning algorithm.

The next subsections describe the details of all algorithms used in our dynamic load-balancing approach.

6.1.2 Load Imbalance Detection

Every C GVT cycles controller detects the imbalance of workload among the worker processes as specified in algorithm 2.

When load imbalance is detected then the maximum and minimum workload are computed by using algorithm 3.

6.1.3 The Computation and Communication Load-balancing Algorithm

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 $PLoad_{min}$.

Algorithm 2 Load imbalance detection algorithm

Input: $cltLt, Lt, tau, cltLt_{min}, cltLt_{max}$

Output: TRUE OR FALSE

Initialisation : $cltLt$ is updated by the SV's local time, Lt, which is calculated as $Lt = Lt + tau$ (tau is the time of next event). tau 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 tau to decrease. When tau decreases, Lp level (SV's) local simulation time (Lt) progresses very slowly. In our experiments we observed that least valued Lt is always overloaded.

1: Controller sends $loadDetectionSignal$ to all worker processes.
2: All processes send back their cluster level local time, $cltLt$, to the controller.
3: 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 $differenceLt = max(cltLt_i - cltLt_{min})$ where $i = 1....n$ processes.
4: **if** $(differenceLt > cltLt_{threshold})$ **then**
5: Return TRUE
6: **else**
7: Return FALSE
8: **end if**

Algorithm 3 Max-min workload calculation

Input: $CollectLoadSignal, PComp, PComm$

Output: $PLoad_{max}, PLoad_{min}$

1: Controller sends $CollectLoadSignal$ to all worker processes.
2: 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:

$$\lambda_i = PComp/(PComp + PComm) \tag{6.2}$$

3: Computes $PLoad_i$ of a worker process i using equation (6.1).
4: Controller determines the highest loaded process, $PLoad_{max}$ and least loaded process, $PLoad_{min}$.

Algorithm 4 The computation load-balancing algorithm

Input: calling algorithm 2

Output: $P_{maxLoad}$, $P_{minLoad}$, $doMigration$

When load imbalance detection algorithm returns TRUE the controller process P_0:

1: Send $collectLoad$ signal to all worker processes.

Upon the receipt of $collectLoad$ signal, each worker process P_i:

2: **for** each LP_j which P_i hosts **do**

3: $PComp_i = PComp_i + LpLoad_j$

4: **end for**

5: calculate $PLoad$ by using equation 1 and send the $PLoad_i$ and $PComm_i[]$ to the Controller processes.

Upon the receiving of $PLoad_i$ and $PComm_i[]$ from all worker process the controller process P_0:

6: Finds the highest loaded process P_j:
$PLoad_{max,j}$ = Max($PLoad_1, PLoad_2, ...PLoad_N$) where j^{th} process has maximum load and N is the number of worker processes

7: Finds least loaded process P_j:
$PLoad_{min,j}$ = Min($PLoad_1, PLoad_2, ...PLoad_N$) where j^{th} process has the minimum load and N is the number of worker processes

8: Sends the $doMigration$ message to $P_{maxLoad}$ to inform about minimum loaded process $P_{minLoad}$.

Upon the receipt of $doMigration$ signal from controller, the highest loaded process:

9: Finds the top n LPs which have the maximum value of $LpComm[Destination]$

10: Sends the n LPs to the destination processor which is the least loaded process $P_{minLoad}$.

Algorithm 5 The communication load-balancing algorithm

Input: calling algorithm 2

Output: $P_{maxCommLoad}$, $P_{minLoad}$, $doMigration$

When load imbalance detection algorithm returns TRUE the controller process P_0:

Send $collectLoad$ signal to all worker processes.

Upon the receipt of $collectLoad$ signal, each worker process P_i:

for each LP_j which P_i hosts **do**

  $PComp_i = PComp_i + LpLoad_j$

end for

calculate $PLoad$ by using equation 1 and send the $PLoad_i$ and $PComm_i[]$ to the Controller processes.

Upon the receiving of $PLoad_i$ and $PComm_i[]$ from all worker process the controller process P_0:

Finds the maximum value of $PComm_i[j]$ i.e. $P_{maxCommLoad}$:

for i = $1ton-1$ **do**

  for j = $1ton-1$ **do**

    if $PComm_i[j] > MaxComm$ **then**

      $Selected1 = i$

      $Selected2 = j$

    end if

  end for

end for

Find the sender process:

if $PComp_{selected1} > PComp_{selected2}$ **then**

  $Sender = P_{selected1}$

else

  $Sender = P_{selected2}$

end if

Finds least loaded process P_j:

$PLoad_{min,j}$ = Min($PLoad_1, PLoad_2, ...PLoad_N$) where j^{th} process has the minimum load and N is the number of worker process.

Sends the $doMigration$ message to Sender process which is highest communication loaded $P_{maxCommLoad}$.

Upon the receipt of $doMigration$ signal from controller, the highest loaded process:

Finds the top n LPs which have the maximum value of $LpComm[Destination]$

Sends the n LPs to the destination processor which is the least loaded process $P_{minLoad}$.

6.2 Reinforcement Learning

Reinforcement learning (RL) is a computational approach to understanding and automating goal-directed learning and decision-making [56]. In reinforcement learning, an agent interacts with an environment. The agent takes actions which cause changes in the environment and the environment, in turn, sends numerical responses to the agent indicating the effectiveness of its actions.

A Markov decision processes (MDP) is a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. The agent and its environment can be represented by a finite MDP (the MDP has with finite number of states and action sets). The MDP consists of:

1. S: a set of environment states.
2. A: a set of actions that the agent can take.
3. π: a policy which is a mapping from the environment to the action and that the agent takes.
4. Reward Function (RF): A reward function maps the state or state-action pair of the environment to a set of scalar rewards R.
5. Value Function (VF): A value function defines the expected return an RL agent can receive for a given policy.

At some point in time t, the agent selects an action a $\in$ (As_t) on the basis of its current state $s_t \in$ S. The goal of the agent is to develop a policy π which maximizes the long-term reward. The reward function indicates the desirability of different states or state-action pairs. It is very important to remember that reward function must reflect the main goal of the system. For example, in the dynamic load-balancing algorithm for NTW, if we assign

a reward for balancing the computational load instead of decreasing the simulation time, the simulation might wind up with a balanced computational load between processors and a bigger simulation time.

6.2.1 Q-Learning

In general, Q-learning [55] is used to find an optimal action-selection policy for any given finite Markov decision process. An agent can utilize Q-learning to acquire an optimal policy using delayed rewards. The agent can find an optimal policy even when there is no prior knowledge of the effects of its actions on the environment [55]. In Q-learning, the reward and the best value of the current state is used to improve the estimate of the previous state-action pair. The Q-learning algorithm estimates the state-action value function as follows:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha[R_t + \gamma Max_a Q(s_{t+1}, a)] \tag{6.3}$$

where α and γ are the learning step and the discount rate, respectively. The discount rate is a measure of how far ahead in time the algorithm looks. It affects how much weight it gives to future rewards in the value function. s_{t+1} is the state reached from state s_t when performing action a_t at time t. The best value at the next state (i.e. $Max_a Q(s_{t+1}, a)$) is the maximum over all actions a that could be executed at the next state s_{t+1}. In order to select an action in a state, we follow a straightforward approach - the highest rewarded action is selected first. Algorithm 6 displays the basic Q-learning algorithm.

6.3 The Control Parameters and Q-Learning for NTW

Basically, we have used a single agent with two-state approach to formulate the dynamic load balancing approach. Here, a central node, the controller process, gathers information from all of the worker processes, runs the Q-learning algorithm, finds new values

Algorithm 6 The Q-learning

for Repeat for each episode **do**
  Initialize s
  for Repeat for each step **do**
    Select and carry out an action a at any time t
    observe reward r and next new state s_{t+1}
    $Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha[R_t + \gamma Max_a Q(s_{t+1}, a)]$
    $s \leftarrow s_{t+1}$
  end for
end for

for the control parameters and informs all of the worker processes 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 at each execution of the dynamic load balancing algorithm. 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. 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 6–1

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

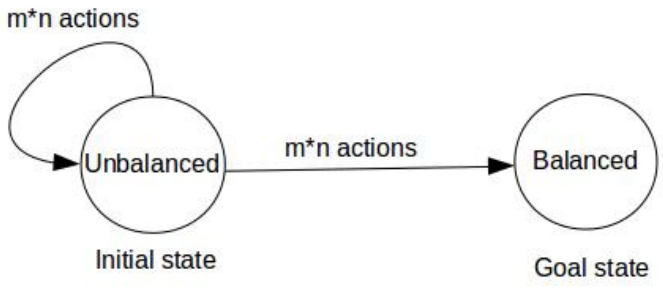


Figure 6–1: States and actions in Q-learning.

The Reward Function: 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:

$$ECR_i = NC_i/(t_i - t_{i-1}) \tag{6.4}$$

where NC_i denotes the number of committed events as GVT_i. Committed events are the processed events whose times stamps are less than 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:

$$ECR_{ref} = (\sum_{i=0}^{D} ECR_i)/(t_D - t_0) \tag{6.5}$$

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:

$$R_i = ECR_i - ECR_{ref} \tag{6.6}$$

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.

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 7 presents the general structure of the Q-learning dynamic load balancing algorithm.

Algorithm 7 Q-learning and Dynamic load balancing

1: In every GVT cycle the controller checks for a load imbalance by running algorithm 2.
2: **if** Algorithm 2 returns FALSE **then**
3: Controller does nothing
4: **else**
5: Controller identifies processes with maximum and minimum workload using algorithm 3.
6: Controller calculates λ using equation (6.2).
7: **if** $\lambda > 0.6$ **then**
8: Run the communication algorithm in process with maximum workload
9: **else**
10: Run the computation algorithm in process with maximum workload
11: **end if**
12: 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, $PLoad_{min}$)
13: Compute the reward R_i by using equation (6.6).
14: Run the Q-learning algorithm with R_i as input.
15: **end if**

Algorithm 8 presents the basic Q-learning algorithm used in dynamic load-balancing approach.

Algorithm 8 Q-learning for NTW

Input: reward, R_i, of the latest execution of the last action.
Output: L and C

1: **for** Repeat for each episode **do**
2: Update the reward of the last action via equation (6.6).
3: Update the Q matrix via equation (6.3)
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.

6.3.1 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 (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. The basic steps of controlling time window is given in algorithm 9.

Algorithm 9 Time Window

Input: L and C of the latest execution of the last action.
Output: W_i

1: Compute the window size by using C and L obtained from algorithm 8.
2: $W_i = C\text{x}L\text{x}U$. Here i = 1 to mxn actions. We use U = 1000.

6.4 Simulation Result

In this section we study the performance of the dynamic load balancing algorithms. We use our discrete event calcium wave model as specified in chapter 5. In the learning algorithm the values of α and γ are 0.1 and 0.9 respectively. Each experimental result is the average of three simulation runs.

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 6–2. 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.

Performance We employed the Ca^{2+} wave model in a hippocampal pyramidal neuron (from NeuroMorpho.Org, C91662, ModelDB:87284, 50 micron distance from soma and consisting of 14749 SVs). We executed about 72 Million events. The execution time

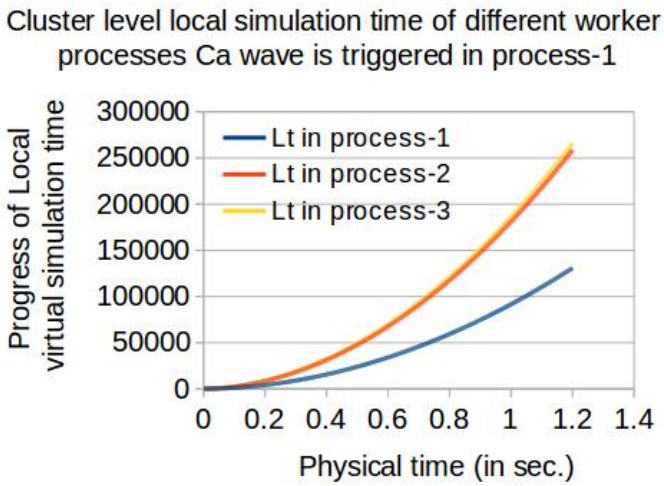


Figure 6–2: Cluster level local simulation time when Ca^{2+} wave is initiated in process 1.

and the number of rollbacks both decrease when load balancing along with a time window is used, as shown in figure 6–3 and figure 6–4 respectively. 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.

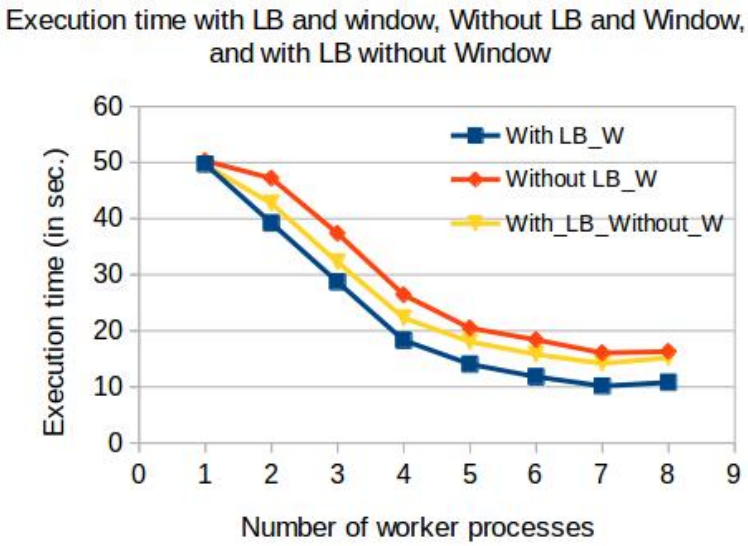


Figure 6–3: Execution time 1) with load balancing and window, 2) with load balancing and without window, and 3) without load balancing and window.

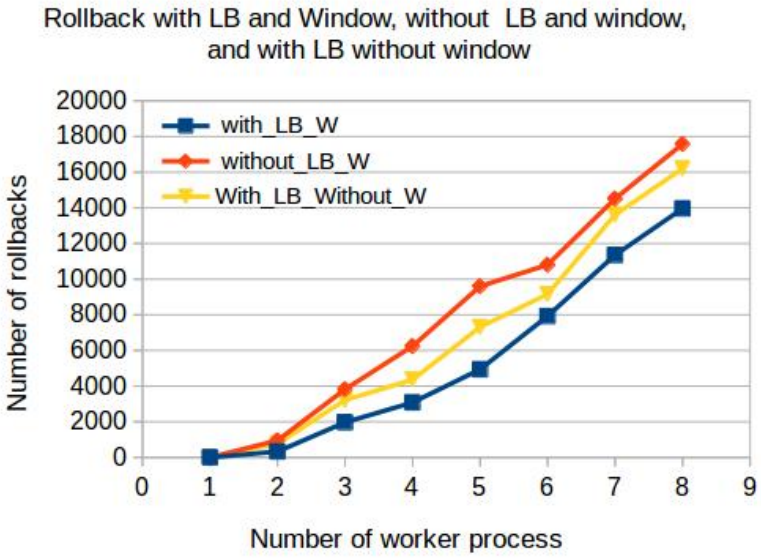


Figure 6–4: Number of rollbacks 1) with load balancing and window, 2) with load balancing and without window, and 3) without load balancing and window.

CHAPTER 7
Conclusion

In this chapter, we first briefly summarize the research contained in this thesis and discuss our thoughts for our future research.

Stochastic, spatial reaction-diffusion simulations have been widely used in systems biology and computational neuroscience. The increasing scale and complexity of simulated models and morphologies have exceeded the capacity of a serial simulator. It is therefore natural to employ parallel simulation for such complex problems. In this thesis, we have developed an optimistic MPI-based, parallel stochastic spatial reaction-diffusion simulator, Neuron Time Warp (*NTW*), to simulate stochastic reaction diffusion systems. Our research is a part of the NEURON project at Yale University.

NTW makes use of a multi-leveled priority queue in order to minimize the cost for scheduling events. A history queue (HQ) is used to minimize unnecessary forwarding of rb-messages to neighboring sub-volumes. Initial experiments made use of the Lotka-Volterra (LV) model to represent chemical reactions and the diffusion of ions taking place on a dendritic branch of a neuron. We observed that NTW scaled well with the number of processes used. The execution time using shared memory improved more than 12% for 4 processors and more than 30% for 6 processors.

The intra-cellular calcium signaling pathways of a neuron depend on both biochemical reactions and diffusions. Some structures (e.g. spines) are so small and chemical concentrations are so low that a small number of molecules diffusing into such a structure

can make a large change in the concentration of the molecules. These events can affect the cellular dynamics in such way that they cannot be evaluated by a deterministic simulation. Stochastic models of such a system provide a more detailed understanding of these systems then deterministic models because they capture their behavior at a molecular level. Chapter 5 describes experiments using models of calcium signaling pathways.

We simulated two models of calcium signaling pathways in a neuron-a calcium buffer and a calcium wave model. The calcium buffer model was employed in order to verify the correctness and performance of NTW by comparing it to a sequential deterministic simulation in NEURON. We found almost same behavior between the serial and parallel runs in high concentration experiment whereas we observed some deviation in case of low concentration experiment. The average (25 sample points) standard deviation of calcium, Buf, and CaBuf dynamics between sequential deterministic and stochastic parallel runs were about 0.098 μM, 0.062 μM and 0.064 μM respectively. To measure the speedup of the parallel simulator, we employed calcium buffer model on the same neuronal branch and ran the parallel simulation for 5.7 virtual time units during which 1.2 million events were processed. We obtained speedup more than 5 for 7 processes.

We developed a discrete event calcium wave model (from a deterministic model) using a stochastic IP_3R model. We employed it in NTW for both 1-D and 3-D geometries. We observed that the propagation of a calcium wave through a neuron led to a large computational imbalance between those areas which were covered and those which were not covered by the wave at a given instant in time. Hence developing a load balancing algorithm became an important undertaking. We developed a distributed dynamic load balancing algorithm to balance the computational and communication loads during the

simulation. In order to tune the parameters of the dynamic load balancing algorithm and also to find a good value for the time window in our Time Warp implementation, we made use of reinforcement learning. We achieved up to a 30% improvement in performance using this algorithm.

A sequential version of *NTW* was integrated with NEURON. The architecture for a connection of parallel *NTW* with NEURON is described in chapter 4. This leads us to a description of our future work.

Open door for future research in biology: This work applies an optimized time warp synchronization approach using next sub-volume method algorithm, named NTW, to study intracellular neuronal dynamics, but reaction-diffusion kinetics arise across a large range of spatial/temporal scales in biology.

- Waves of elevated intracellular calcium using the same underlying ER kinetics considered here arise in the Xenopus immediately following fertilization and increase inositol-1,4,5-trisphosphate (IP_3). This fertilization calcium wave in Xenopus laevis is originated at the point of spermegg fusion and traverses the entire diameter of the egg. [58] proposed a fertilization calcium wave which is a reaction diffusion model.
- In the brain, spreading depression – which is linked to migraines [59] and ischemic stroke [60] – is driven by regenerative potassium waves where the potassium is released from cells and diffusion is across the extracellular space. The details of the extracellular space create small "compartments" that have low levels of molecules that need to be addressed stochastically.
- In ecology, a fox eating a hare is a "reaction", as is their reproduction. Random movement of animals is a random walk (aka diffusion).

7.1 Future Work-Short Term and Long Term

Our short term objectives revolve around the improvement of our load balancing algorithms. In this thesis, we applied a single agent Q-learning technique to tune the parameters of the load balancing algorithm. In this approach a learning agent tries to find the optimal values for the load balancing parameters by interacting with an environment. For the distributed algorithm, a multi-agent learning approach can be employed to tune the dynamic load balancing algorithm's parameters at each node. In this approach agents have to cooperate with one other in order to find optimal values for the control parameters.

Simulated annealing might be another option to optimize the performance of dynamic load balancing algorithm [49]. I plan to investigate the performance of load balancing algorithms using simulated annealing approach to tune the parameters of the load balancing algorithm.

The long term objectives for our research are to:

1. develop a combined deterministic and stochastic discrete event model for reaction diffusion system so that connected NTW-NEURON can be used as an adaptive stochastic-deterministic solver.
2. develop a combined electrical excitation (i.e. extracellular) and intracellular calcium dynamics models for neurons.

These objectives are quite ambitious. We leave you with an intriguing thought-given the complexity of combining continuous and discrete event simulations it is possible that the right approach is to not combine the two models/simulations but instead to develop one discrete event model for both of the models/simulators. So we end the thesis with a question.

References

[1] Richard M. Fujimoto. *Parallel and Distribution Simulation Systems*, John Wiley and Sons, Inc., New York, USA, 2000.

[2] L. Dematte, T. Mazza *On parallel stochastic simulation of diffusive systems*, Proceeding of the 6th International Conference on Systems Biology, Pages: 191-200, Germany: Springer 2008.

[3] Azevedo FA, Carvalho LR, Grinberg LT, Farfel JM, Ferretti RE, Leite RE, Jacob Filho W, Lent R, Herculano-Houzel S. *Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaledup primate brain*, Journal of Comparative Neurology, 513(5), 532-541, 2009.

[4] D. Sterratt, B.Graham, A. Gillies, and D.Willshaw *Principles of Computational modeling in Neuroscience*, UK: Cambridge University Press, 2011.

[5] Michael J. Berridge *Neuronal calcium signaling*, Neuron, Vol. 21, 13-26, Cell Press: 1998.

[6] Iain Hepburn, Weiliang Chen, Stefan Wils, and Erik De Schutter *STEPS: efficient simulation of stochastic reaction-diffusion models in realistic morphologies*, BMC systems biology, 6:36, 10 May 2012.

[7] Blackwell KT *Approaches and tools for modeling signaling pathways and calcium dynamics in neurons*, Journal of neuroscience methods, 220(2):131-140, 15 November 2013.

[8] D. T. Gillespie *Exact stochastic simulation of coupled chemical reactions*, The Journal of Physical Chemistry, Vol.81(25) pp. 23402361, USA: 1977.

[9] M. A. Gibson and G. Bruck. *Efficient exact stochastic simulation of chemical systems with many species and many channels*, The Journal of Physical Chemistry A, 104(9):18761889, USA: 2000.

[10] Y. Cao, H. Li, and L. Petzold *Efficient formulation of the stochastic simulation algorithm for chemically reacting systems*, The Journal of Chemical Physics,121(9):4059 - 4067 USA: AIP Publishing, 2004.

[11] D. T. Gillespie *Approximate accelerated stochastic simulation of chemically reacting systems*, Journal of Chemical Physics, Vol.115, Number:4, USA: AIP Publishing, 2001.

[12] K. Takahashi, K. Kaizu, B. Hu, and M. Tomita *A multi-algorithm, multi-timescale method for cell simulation*, Bionformatics, 6:538 - 546, UK:Oxford University Press 2004

[13] A. R. Leach, *Molecular modeling:Principles and Applications*, USA: Prentice Hall, 2001.

[14] Kerr R, Bartol TM, Kaminsky B, Dittrich M, Chang JCJ, Baden S, Sejnowski TJ, Stiles JR *Fast Monte Carlo Simulation Methods for Biological Reaction-Diffusion Systems in Solution and on Surfaces*, SIAM J. Sci. Comput., 30(6):3126-3149.

[15] Steven S Andrews, Nathan J Addy, Roger Brent, and Adam P Arkin *Detailed simulations of cell biology with Smoldyn 2.1*, PLoS Comput Biol, 6(3):e1000705, 12 March 2010.

[16] J. Elf and M. Ehrenberg, *Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases*, System Biology (Stevenage), IEEE Proceeding, 1(2):230 - 236, IET 2004.

[17] Yi bing Lin and Paul A. Fishwick, *Asynchronous parallel discrete event simulation.*, IEEE transactions on Systems, Man and Cybernetics, 26:397-412, 1996.

[18] David R. Jefferson *Virtual time*, ACM Trans. Program. Lang. Syst., Vol. 7, No. 3, Pages: 404 - 425, 1985.

[19] B. Wang, Y. Yao, B. Hou, S. Peng *Experimental analysis of optimistic synchronization algorithms for parallel simulation of reaction-diffusion systems*, International Workshop on High Performance Computational Systems Biology, Pages: 91 - 1001, Trento, Italy : 2009.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. *Introduction to Algorithms*, MIT Press, Cambridge, Massachusetts London, England, 2001.

[21] K. M. Chandy and J. Misra. *Distributed Simulation: A case study in design and verification of distributed programs.*, Software Engineering, IEEE Transactions on SE-5(5):440-452, 1979.

[22] Christopher H. Young and Philip A. Wilsey. *Optimistic fossil collection for time warp simulation.*, In Proceedings of the 29th Hawaii International Conference on System Sciences, pp. 364, Washington, DC, USA, IEEE Computer Society, 1996.

[23] K. S. Panesar and R. M. Fujimoto *Adaptive flow control in time warp.*, In Parallel and Distributed Simulation-1997, Proceedings 11th Workshop on Parallel and Distributed Simulation, page108-115, 1997

[24] D. T. Gillespie *A rigorous derivation of the chemical master equation*, Physica A, 188: 404425 USA: 1977.

[25] M. Jeschke, R. Ewald, A.Park, R. Fujimoto and A. M. Uhrmacher *Parallel and Distributed Spatial Simulation of Chemical Reactions*, 22nd Workshop on Principles of Advanced and Distributed Simulation, Pages: 51-59, IEEE 2008.

[26] Herve Avril, C. Tropper *Clustered Time Warp and logic simulation*, SIGSIM Simul. Dig., 25(1:112-119), 1995.

[27] Q. Xu, C. Tropper *XTW, a parallel and distributed logic simulator*, 26th Workshop on Principles of Advanced and Distributed Simulation, pp. 181-188, ACM/IEEE/SCS, 2005.

[28] M. L. Hines and N. T. Carnevale *The NEURON Book*, UK: Cambridge University Press, 2006.

[29] 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

[30] Robert McDougal, Hines ML and William W. Lytton *Reaction-diffusion in the NEURON simulator*, Front Neuroinform, 7:28 2013.

[31] Neymotin SA, Dura-Bernal S, Lakatos P, Sanger TD and William W. Lytton *Multitarget Multiscale Simulation for Pharmacological Treatment of Dystonia in Motor Cortex*, Front Pharmacol, 7:157 2016.

[32] N. Chamakuri, R. Sten *Whole-cell simulations of hybrid stochastic and deterministic calcium dynamics in 3D geometry*, Journal of Computational Interdisciplinary

Sciences, 3(1-2):3 - 18, Pan-American Association of Computational Interdisciplinary Sciences: 2012.

[33] Michael J Byrne, M Neal Waxham, and Yoshihisa Kubota *Cellular dynamic simulator: an event driven molecular simulation environment for cellular physiology*, Neuroinformatics, 8(2):63-82, June 2010.

[34] B. Wang, Y. Yao, B. Hou, S. Peng *Abstract Next Subvolume Method: A logical process-based approach for spatial stochastic simulation of chemical reactions.*, Computational Biology and Chemistry vol. 35, pp. 5193 - 198, Washington, DC, USA: IEEE Computer Society, 2011.

[35] J. S. Steinman *SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation*, SCS Western Multiconference on Advances in Parallel and Distributed Simulation (PADS91), vol. 23, pp. 95 - 103, 1991.

[36] Michael J. Hallock, John E. Stone, Elijah Roberts, Corey Fry, Zaida Luthey-Schulten *Simulation of reaction-diffusion processes over biologically relevant size and time scales using multi-GPU workstations*, Journal of Parallel Computing, vol. 40 pp. 86 - 99, USA: Elsevier B.V., 2014

[37] Samuel A. Neymotin, Robert A. McDougal, Mohamed A.Sherif, Christopher P. Fall, Michael L. Hines and William W. Lytton *Neuronal calcium wave propagation varies with changes in endoplasmic reticulum parameters: a computer model*, Neural Computation, Vol. 27, No. 4, Pages 898-924, MIT press, 2015.

[38] Martin Falcke *On the role of stochastic channel behavior in intracellular Ca^{2+} dynamics*, Biophysical Journal, Volume 84, pp. 4256, Biophysical Society, 2003.

[39] Jianwei Shuai, John E. Pearson, J. Kevin Foskett, Don-On Daniel Mak, and Ian Parker *A Kinetic Model of Single and Clustered IP_3 Receptors in the Absence of Ca^{2+} Feedback*, Biophysical Journal, Volume 93, pp. 1151 - 1162, Biophysical Society, 2007.

[40] W. Chu et al., *Task allocation in distributed data processing*, IEEE Comput. 13: 15-69 1980.

[41] E. DeSouza e Silva and M. Gerla, *TLoad balancing in distributed system with multiple classes and site constraints*, in Proc. Performance 84 (1984) 17-33 1984.

[42] A.N. Tantawi and D. Towsley, *Optimal load balancing in distributed computing system*, J. Assoc. Comput. Machinery, 32(2): 445-465 1985.

[43] A. Barak and A. Shiloh *A distributed load balancing policy for a multicomputer*, Software Practice Experience, 15(9): 901-913 1985.

[44] T.C.K. Chou and J.A. Abraham *Distributed control of computer systems*, IEEE Trans. Comput. C-35 (6): 564-567 1986.

[45] C. Barmon, M.N. Faruqui and G.P. Battacharjee *Dynamic load balancing algorithm in a distributed system*, Microprocessing and Microprogramming, 29:273-285 North Holland: 1991.

[46] Ewa Deelman and Boleslaw K. Szymanski *Dynamic load balancing in parallel discrete event simulation for spatially explicit problems*, Proc. 12th workshop on parallel and distributed simulation-PADS98, IEEE computer society press, pp. 46-53 Calgary, Canada: 1998.

[47] Herve Avril and Carl Tropper, *The dynamic load balancing of clustered time warp for logic simulation.*, SIGSIM Simul. Dig., 25(1):112-119, 1995.

[48] A. C. Palaniswamy and P. A. Wilsey, *Adaptive bounded time windows in an optimistically synchronized simulator.*, In VLSI, 1993, Design Automation of High Performance VLSI Systems, Proceedings, Third Great Lakes Symposium on, pp. 114-118, 1993.

[49] Sina Meraji, Wei Zhang, Carl Tropper *On the Scalability and Dynamic Load-Balancing of Optimistic Gate Level Simulation*, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 29, NO. 9, pages: 1368-1380, 2010.

[50] Mohammand Nazrul Ishlam Patoary, Carl Tropper, Zhongwei Lin, Robert McDougal and William W. Lytton *Neuron Time Warp*, Proceedings of the 2014 Winter Simulation Conference, pages:3447-3458, NJ, USA: IEEE Press, December 2014.

[51] Friedemann Mattern *Efficient Algorithms for Distributed Snapshots and Global Virtual Time Approximation*, Journal of Parallel and Distributed Computing, Vol. 18, No. 4, 1993.

[52] Rinaldo B. Schinazi *Predator-prey and host-parasite spatial stochastic models.* The Annals of Applied Probability, Volume 7(1):1-9, 1997.

[53] J. Marchant and I. Parker *Role of elementary Ca^{2+} puffs in generating repetitive Ca^{2+} oscillations*, EMBO J., 20, pp. 65 - 71, 2001

[54] Marisa Brini, Tito Cali, Denis Ottolini, Ernesto Carafoli *Neuronal calcium signaling: function and dysfunction*, Cellular and Molecular Life Sciences, 71:27872814, August 2014.

[55] C. J. C. H. Watkins, P. Dayan *Q-Learning*, Machine Learning, 8:279292, 1992.

[56] Shai Shalev-Shwartz and Shai Ben-David *UNDERSTANDING MACHINE LEARNING:From Theory to Algorithms.* Cambridge University Press, 2014.

[57] Prof. James Sneyd *Models of calcium dynamics.* U. Auckland, New Zealand, James Sneyd (2007), Scholarpedia, 2(3):1576, doi:10.4249/scholarpedia.1576.

[58] Christopher P. Fall, John M. Wagner, Leslie M. Loew, Richard Nuccitelli *Cortically restricted production of IP 3 leads to propagation of the fertilization* Ca^{2+} *wave along the cell surface in a model of the Xenopus egg.* Journal of Theoretical Biology 231 (2004) 487496.

[59] Hall GC, Brown MM, Mo J, MacRae KD. *Triptans in migraine: the risks of stroke, cardiovascular disease, and death in practice.* Neurology. 2004;62:5638

[60] Sochurkova D, Moreau T, Lemesle M, Menassa M, Giroud M, Dumas R. *Migraine history and migraine-induced stroke in the Dijon stroke registry.* Neuroepidemiology. 1999;18:8591.