BIOPHYSICALLY ACCURATE BRAIN MODELING AND SIMULATION USING HYBRID MPI/OPENMP PARALLEL PROCESSING

A Thesis

by

JINGZHEN HU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2012

Major Subject: Computer Engineering

Biophysically Accurate Brain Modeling and Simulation Using Hybrid MPI/OpenMP Parallel Processing

BIOPHYSICALLY ACCURATE BRAIN MODELING AND SIMULATION USING HYBRID MPI/OPENMP PARALLEL PROCESSING

A Thesis

by

JINGZHEN HU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Peng Li
Committee Members,	Eun Jung Kim
	Paul V. Gratz
Head of Department,	Costas Georghiades

May 2012

Major Subject: Computer Engineering

ABSTRACT

Biophysically Accurate Brain Modeling and Simulation Using Hybrid MPI/OpenMP Parallel Processing. (May 2012)

Jingzhen Hu, B.S., Beijing University of Posts and Telecommunications

Chair of Advisory Committee: Dr. Peng Li

In order to better understand the behavior of the human brain, it is very important to perform large scale neural network simulation which may reveal the relationship between the whole network activity and the biophysical dynamics of individual neurons. However, considering the complexity of the network and the large amount of variables, researchers choose to either simulate smaller neural networks or use simple spiking neuron models. Recently, supercomputing platforms have been employed to greatly speedup the simulation of large brain models. However, there are still limitations of these works such as the simplicity of the modeled network structures and lack of biophysical details in the neuron models. In this work, we propose a parallel simulator using biophysically realistic neural models for the simulation of large scale neural networks. In order to improve the performance of the simulator, we adopt several techniques such as merging linear synaptic receptors mathematically and using two level time steps, which significantly accelerate the simulation. In addition, we exploit the efficiency of parallel simulation through three parallel implementation strategies: MPI parallelization, MPI parallelization with dynamic load balancing schemes and Hybrid

MPI/OpenMP parallelization. Through experimental studies, we illustrate the limitation of MPI implementation due to the imbalanced workload among processors. It is shown that the two developed MPI load balancing schemes are not able to improve the simulation efficiency on the targeted parallel platform. Using 32 processors, the proposed hybrid approach, on the other hand, is more efficient than the MPI implementation and is about 31X faster than a serial implementation of the simulator for a network consisting of more than 100,000 neurons. Finally, it is shown that for large neural networks, the presented approach is able to simulate the transition from the 3Hz delta oscillation to epileptic behaviors due to the alterations of underlying cellular mechanisms.

DEDICATION

To my parents

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. Li, and committee members, Dr. Kim and Dr. Gratz, for their support and guidance on the research.

Thanks to my friends Boyuan Yan, Yong Zhang, Mingchao Wang and the ECE faculty and staff for helping on my studies at Texas A&M University. Finally, thanks to my parents and grandparents for their support and love.

TABLE OF CONTENTS

Page

ABSTRACT iii

DEDICATION v

ACKNOWLEDGEMENTS vi

TABLE OF CONTENTS vii

LIST OF FIGURES ix

LIST OF TABLES xi

1. INTRODUCTION 1

    1.1 Background and Motivation 1
    1.2 Contributions 2

2. FUNDAMENTALS OF BRAIN MODELS AND COMPUTATIONAL TECHNIQUES 5

    2.1 Introduction 5
    2.2 Models of Neurons 9
    2.3 Models of Synapses 15
    2.4 Model of the Network 16
    2.5 Numerical Integration Methods 18
    2.6 Parallel Computing 22

3. PROPOSED NETWORK SIMULATION TECHNIQUES 29

    3.1 Introduction 29
    3.2 Merging of Linear Synaptic Receptors... 30
    3.3 Simulation Speedup Considering the Data Locality 31

4. PARALLEL NEURAL NETWORK SIMULATION METHODS 33

    4.1 MPI Parallelization 33
    4.2 MPI Parallelization with Dynamic Load Balancing 37

4.3 Hybrid MPI/OpenMP Parallelization 41

5. EXPERIMENTAL RESULTS AND ANALYSIS 43

5.1 Experiments Setup 43
5.2 Simulation of Delta Oscillation and Epileptic Activities 44
5.3 MPI Parallelization 45
5.4 MPI Load Balancing Schemes 48
5.5 Hybrid MPI/OpenMP Parallelization 49

6. SUMMARY AND CONCLUSIONS 53

REFERENCES 55

VITA 59

LIST OF FIGURES

FIGURE		Page
1	Organization of human brain	5
2	Structure of a neuron	6
3	HH model compartment with ion channels and synaptic receptors	10
4	Structure of cerebral cortex and thalamus	17
5	Forward Euler	18
6	Backward Euler	19
7	Trapezoidal rule	20
8	Comparison between serial processing and parallel processing	22
9	Point to point routine	26
10	Collective scatter routine	27
11	Merging of linear synaptic receptors.	30
12	Two level time steps.	32
13	Message vector stored in each processor	34
14	Gathering the entire firing message.	35
15	MPI parallel algorithm.	36
16	Load balancing problem.	37
17	2D Torus	38
18	Dynamic load balancing scheme 1	39
19	Dynamic load balancing scheme 2	40
20	Structure of hybrid MPI/OpenMP parallelization.	41

FIGURE Page

21 Architecture of one node in the supercomputer 43

22 Delta wave and epilepsy..... 45

23 Speedup of the MPI implementation..... 46

24 Efficiency of the MPI implementation. 47

25 Imbalance ratio of the MPI implementation 48

26 Comparison of speed up using different load balancing schemes. 49

27 Comparison between different scheduling schemes 50

28 Comparison of simulation based on different methods..... 52

29 Comparison of speed up based on different methods 52

LIST OF TABLES

TABLE Page

1 22 types of neurons 8

2 Comparison between MPI and OpenMP..... 24

3 Comparison of runtime between different methods on 105k network..... 51

1. INTRODUCTION

1.1 Background and Motivation

In past several years, neural network simulation based on different models has drawn great attention. By conducting computer simulations, researchers have examined the close relationship between certain brain disorders (such as epilepsy and Parkinson) and the brain dynamics. For example, in the study of epilepsy, the work from (Van Drongelen et al, 2005) shows that by weakening excitatory synapses, seizure like activity may happen. To better understand and explore how our brain works, it is pivotal to use more complex and biophysical neuron and network models. With the increasing advancements on anatomy and neuroscience, a large number of works reveal more in-depth understanding on the brain organization including the cerebral cortex and thalamus. Also a large number of neuron models have been studied. By taking biophysical mechanisms of neurons into consideration, the Hodgkin-Huxley (HH) model (Hodgkin & Huxley, 1952) explicitly includes models of synaptic receptors and ion channels. However, considering the large number of parameters in the HH model, most existing works are limited by the size and complexity of the neuron network. Later, some phenomenological models such like spiking model (Izhikevich, Gally, & Edelman, 2004) and integrate and fire model (Odabasioglu, 2004) were proposed. Though those simpler models may consume less computation power, they cannot directly connect with the biological dynamic behavior of ion channels (Abbott & Van Vreeswijk, 1993).

This thesis follows the style of *Neural Computation*.

Moreover, with the emergence of supercomputing platforms, large scale neural network simulations have been conducted which can be completed within a reasonable time. For instance, recently researchers at IBM implemented a Brain-scale simulation of the neocortex on an IBM supercomputer (Blue Gene), comprising up to 11 billion synapses and 22 million neurons (Djurfeldt & Lundqvist, 2008). But those works are based on either spiking neural models or a simple network without considering the realistic cerebral structure. In addition, graphics processing units or GPUs are used in neural network simulations (Nageswaran et al, 2009) (Wang, Yan, Hu, & Li, 2011). Considering the limitation of the GPU architecture, the neuron and synapse models they used are not biologically realistic. Therefore, in order to get meaningful results from the neural network simulation, both biophysically realistic models and a large scale neural network are needed.

1.2 Contributions

In this work, parallel neural network simulations are proposed based on biophysically realistic models and more detailed structure of cerebral cortex and thalamus. The network is divided into multiple regions in both brain hemispheres and consists of six layers. Also both local connections inside a region and long range global connection between different regions are constructed according to the data from (Peters & Payne, 1993) (Binzegger, Douglas, & Martin, 2004) and (Izhikevich & Edelman, 2008). There are 22 types of neurons in the network and each type of neuron which is composed by different numbers of compartments is modeled by the multi-compartment

HH model. Various ion channels are explicitly included to reflect the roles played by biophysical mechanisms. There are four kinds of synapse receptors in the modeled including AMPA, NMDA, $GABA_A$ and $GABA_B$. Once a presynaptic neuron fires, the dynamics of corresponding synaptic receptors will gradually change according to different differential equations (Izhikevich & Edelman, 2008) (Rubchinsky, Kopell, & Sigvardt, 2003).

To speed up the simulation, we adopt several techniques in the simulation. Since most synaptic receptor models are linear, we merge all these linear models mathematically which effectively reduces the number of differential equations and improves the simulation efficiency. Also by using two level time steps, we can continuously simulate one neuron for multiple small steps inside one outer macro step. As a result the cache hit rate is significantly increased.

Finally, several parallel strategies have been proposed to improve the efficiency of the neural network simulator. First, a MPI parallelization is implemented. As the result of different firing frequencies of neurons in different processors, our results indicate the workload among processors is imbalanced. Two dynamic load balancing schemes are developed which effectively make the workload on different processors evenly but the simulation efficiency is not improved. By taking the advantage of dynamic scheduling schemes of OpenMP, a Hybrid MPI/OpenMP Parallel simulator is developed. The results show that the new strategy not only leads to noticeable efficiency improvement (10 % faster than MPI implementation), but also show an excellent load

balancing. Using 32 processors, the simulator achieves 31X faster than the basic serial implementation for a network consisting more than 100,000 neurons.

2. FUNDAMENTALS OF BRAIN MODELS AND COMPUTATIONAL TECHNIQUES

2.1 Introduction

The human brain, located in the center of nervous system, is a highly complex organ. Figure 1 shows the functional part of the brain including: Cerebral cortex, thalamus, hippocampus, corpus callosum, cerebellum and spinal cord. The cerebral cortex, which is often referred to as the gray matter, plays an important role in listening, thinking, memory, understanding of languages and movement; while the thalamus acts

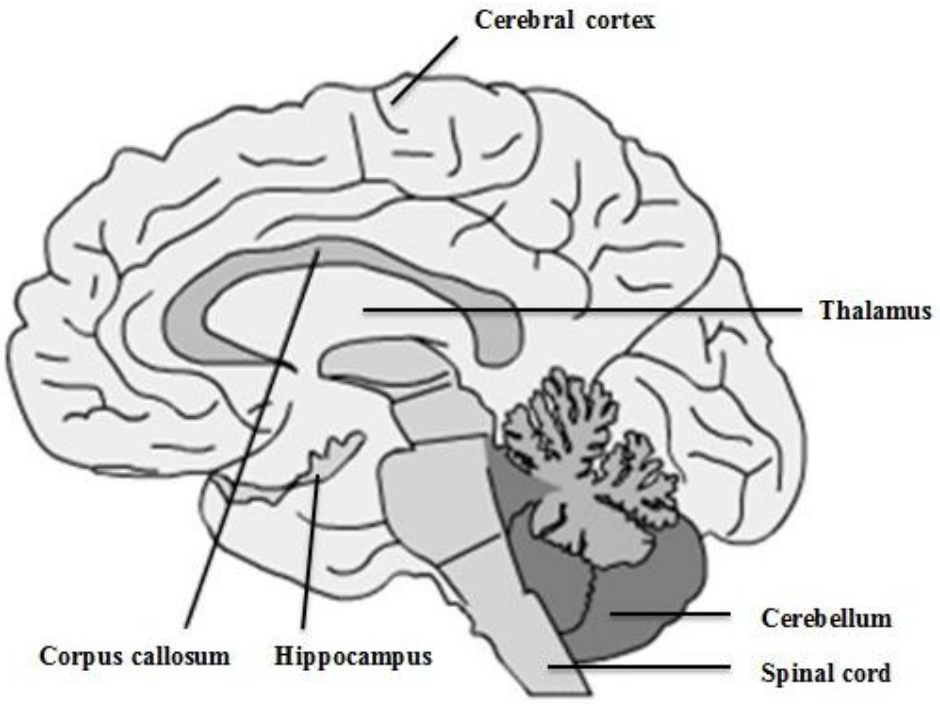

Figure 1 Organization of human brain.

as a relay and projects signals to the corresponding areas of the cerebral cortex (Kailasanath & Fu, 2003). In this work, cerebral cortex and thalamus are constructed in the targeted neural network.

There are around 20 billion neurons in our cerebral cortex and thousands of connections for each neuron. A neuron consists of a soma, an axon and dendrites (Figure 2). The soma is the body of the neuron for processing received signals and generates signals to be sent. The axon is a long filament attached to the soma; its functionality is to transmit signals sent from the soma to all postsynaptic neurons. There are a number of dendrites attached to each soma; dendrites have tree-like structures. The soma and these dendrites attached to it can receive signals from the axons of other neurons.

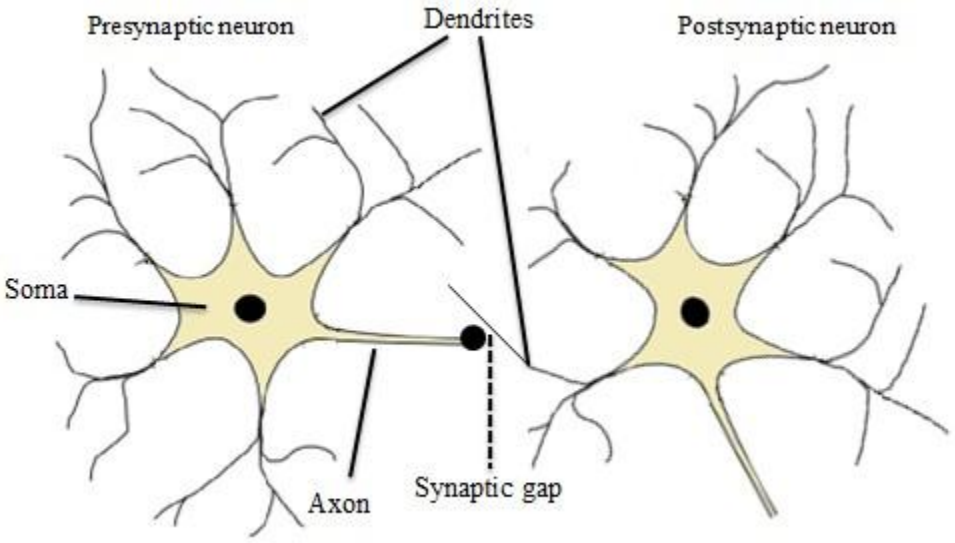

Figure 2 Structure of a neuron.

In neural networks, a signal is transmitted from one neuron to another through a junction called synapse. Correspondingly, the two neurons are called presynaptic neuron and postsynaptic neuron. Signals can only be transmitted from the former to the latter through a synapse. Depending on the presynaptic neuron, signals are classified into two types: excitatory and inhibitory. The former tends to enlarge the membrane voltage of postsynaptic neurons, while the latter may have the opposite effect and tends to decrease the membrane potential. In reality, each neuron has typically thousands of synapses. Since we track the dynamics of each synapse in our model, to reduce the computational effort without sacrificing much accuracy of the model, we scale down the amount of synapses of a neuron by a factor of 0.05. This way, the typical number of synapses on each neuron model is several hundreds.

To keep the structural information in the multi-compartment model of neurons, each compartment is confined to a specific layer. This way, for a neuron that spans several layers, the dendrite and soma part in certain layer is modeled by different number of interconnected compartments. The number of compartments is chosen such that the number of synapses per compartment is closest to 40 (Izhikevich & Edelman, 2008). Since the functionality of axons is to transmit signals, we only consider the delay in the model. Based on the morphology and the functionality, neurons are classified into 22 basic types, as shown in Table 1 from Appendix of (Izhikevich & Edelman, 2008).

In the following three sections, the models for the neuron, synapse and the network structure will be illustrated in detail. Then the basic numerical integration methods are covered.

Table 1 22 types of neurons.

Neuron Type	Excitability	description
p2/3	Excitatory	Pyramidal in L2/3
ss4(L2/3)	Excitatory	Spiny stellate in L4 (project to L2/3)
ss4(L4)	Excitatory	Spiny stellate in L4
p4	p4 Excitatory	Pyramidal in L4 (project to L4)
p5(L2/3)	Excitatory	Pyramidal in L5 (project to L2/3)
p5(L5/6)	Excitatory	Pyramidal in L5 (project to L5/6)
p6(L4)	Excitatory	Pyramidal in L6 (project to L4)
p6(L5/6)	Excitatory	p6(L5/6) Excitatory Pyramidal in L6 (project to L5/6)
b2/3,b4,b5,b6	Inhibitory	Inhibitory Basket interneurons in L2/3/4/5
nb1,nb2/3,nb4,nb5,nb6	Inhibitory	Non-basket interneurons in all layers
TCs/TCn	Excitatory	Thalamocortical interneurons in specific/nonspecific nucleus
TIs/TIn	Inhibitory	Thalamocortical interneurons in specific/nonspecific nucleus
RE	Inhibitory	Thalamic reticular cells

2.2 Models of Neurons

2.2.1 Spiking models

In neural network simulation, what we need to do is to track the action potential (spike) of a neuron. So Izhikevich (Izhikevich, Gally, & Edelman, 2004) proposed a spiking model to describe the spiking dynamics of a neuron and dendrites. By using only two differential equations and four variables, it successfully reproduces the spiking activity.

$$C\dot{v} = k \cdot (v - v_r) \cdot (v - v_{th}) - u + I,$$

$$\dot{u} = a \cdot \{b \cdot (v - v_r) - u\}, \tag{2.1}$$

$$I(t) = -I_{syn} - I_{dendr}, \tag{2.2}$$

In the equation, C denotes the membrane capacitance of the neuron, v denotes the membrane voltage, v_{th} is the threshold voltage, v_r is the resting voltage, u is the recovery voltage and I is the summation of the synaptic input currents and dendritic currents. a and b are variables. When a neuron fires, the membrane voltage will be larger than the peak voltage. So if $v > v_{peak}$ and a spike will be generated, and we will do a reset operation: reset v to c, u to $u+d$. Here we need to notice that unlike the threshold voltage , v_{peak} denotes the peak voltage of a spike, and the value is a constant (50 mV); while the firing threshold voltage in the spiking model depends on the activities of a neuron. As we know, there are many types of neurons with different sizes and firing patterns. In the spiking model, different neuron will have different values of parameters. Then the spiking model will reproduce the neural dynamics in the cortex and thalamus.

2.2.2 HH models

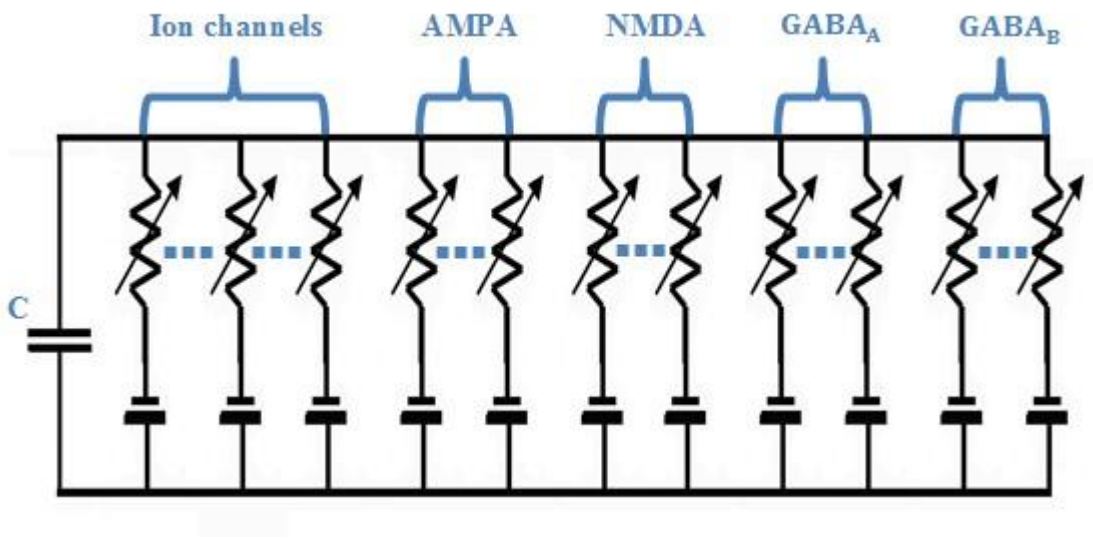

Figure 3 HH model compartment with ion channels and synaptic receptors.

Hodgekin and Huxley (1952) first presented the HH model to generate the action potential of the squid axon. Originally, only three types of channels were contained in the model: leakage channel, sodium channel and potassium channel. Later, more channels were studied to describe different firing patterns of neurons.

As shown in Figure 3, a neuron compartment is modeled by an equivalent circuit with several components. The behavior of the circuit can be illustrated by the following differential equation

$$C_m \cdot \frac{dV}{dt} = -g_{Leak} \cdot (V - E_{Leak}) - I_{ion} - I_{com} - I_{syn}, \qquad (2.3)$$

C_m denotes the membrane capacitance of the compartment, V denotes the intercellular potential, g_{Leak} denotes the resting membrane conductance, E_{Leak} denotes the reversal potential, I_{ion} is the current caused by all ion channels, I_{syn} is the synaptic input current, I_{com} is caused by the conductance between this compartment and all neighboring. The current between two compartments is proportional to the difference between two membrane potentials.

There are five kinds of ion channels in our model such as I_{Na}, I_K, I_M, I_L and I_T. I_{Na} and I_K are the currents of sodium and potassium channels respectively and they are used for generating normal action potentials, I_M is the current of the slow voltage dependent potassium channel which is used for spike frequency adaptation, I_T is the current of low threshold calcium channel and I_L is high threshold calcium channel current. The following equations and data are from (Destexhe, Bal, McCormick, & Sejnowski, 1996a) (Huguenard & Prince, 1992) and (Pospischil et al, 2008).

a) Sodium and potassium currents

$$I_{Na} = g_{Na} \cdot (V - E_{Na}) \cdot m \cdot h^3,$$

$$\dot{h} = -h \cdot \alpha_h(V) + (1 - h) \cdot \beta_h(V),$$

$$\dot{m} = -m \cdot \alpha_m(V) + (1 - m) \cdot \beta_m(V),$$

$$\alpha_m = \frac{0.28 \cdot (-40 + V - V_t)}{-1 + exp\left[\frac{-40 + V - V_t}{5}\right]},$$

$$\beta_m = \frac{0.32 \cdot (13 - V + V_t)}{-1 + exp\left[\frac{13 - V + V_t}{4}\right]},$$

$$\alpha_h = \frac{4}{1 + exp\left[\frac{40 - V + V_t}{5}\right]},$$

$$\beta_h = 0.128 \cdot exp\left[\frac{17 - V + V_t}{18}\right], \tag{2.4}$$

where E_{Na} denotes the reversal voltage, g_{Na} denotes the maximum conductance and p denotes the probability of this ion channel to be open at the current time. The value of g_{Na} is determined by the density of Na channels. Also, when we calculating the probability p, we need to know that there are two independent processes in each channel: the activation and the inactivation state. In order to get the current of a sodium channel we assume that there are one inactivation gate and three activation gates for each channel. We use m and h to describe the probabilities of the "activation" and "inactivation" gates to be open, respectively.

$$I_K = g_K \cdot (V - E_K) \cdot m^4,$$

$$\dot{m} = (1 - m) \cdot \beta_m(V) - m \cdot \alpha_m(V),$$

$$\alpha_m = 0.5 \cdot exp\left[\frac{10 - V + V_t}{40}\right],$$

$$\beta_m = \frac{-0.032 \cdot (V - 15 - V_t)}{exp\left[\frac{15 - V + V_t}{5}\right] - 1}, \tag{2.5}$$

where E_K is the reversal potential and normally it is -90mV, m is the activation variable and its dynamic is formulated by a one order differential equation and similarly as the sodium channel, g_K is the maximal conductance.

b) Slow potassium current

$$I_M = g_M \cdot (V - E_K) \cdot n,$$

$$\dot{n} = \frac{n_\infty(V) - n}{\tau_n(V)},$$

$$n_\infty(V) = \frac{1}{1 + exp\left[\frac{-V - 35}{10}\right]},$$

$$\tau_p(V) = \frac{\tau_{max}}{exp \cdot \left[\frac{-V - 35}{20}\right] + 3.3 \cdot \ exp\left[\frac{V + 35}{20}\right]}, \quad (2.6)$$

where g_M is the maximum conductance, E_K is the reversal potential and *n* is the activation variable.

c) Calcium currents to generate bursting

Generally, the firing pattern of most neurons in thalamus is bursting which was modeled by the low threshold calcium current. As illustrated in (Huguenard & Prince, 1992), it is formulated as follows:

$$I_T = g_T \cdot (V - E_{Ca}) \cdot m \cdot {s_\infty}^2,$$

$$\dot{m} = \frac{m_\infty(V) - m}{\tau_m(V)},$$

$$s_\infty(V) = \frac{1}{1 + exp\left[\frac{-V - V_x - 57}{6.2}\right]},$$

$$m_\infty(V) = \frac{1}{1 + exp\left[\frac{V + 81 + V_x}{4}\right]},$$

$$\tau_m(V) = \frac{(211.4 + exp\left[\frac{(113.2 + V + V_x)}{5}\right]) + 30.8}{3.7 \cdot (1 + exp\left[\frac{(84 + V + V_x)}{3.2}\right])}, \qquad (2.7)$$

where g_T is the maximal conductance of the T type current and V_x is a uniform shift of the voltage dependence.

There is another type of bursting which is modeled by high threshold calcium current.

$$I_L = g_L \cdot (V - E_{Ca}) \cdot q \cdot p^2,$$

$$\dot{p} = \frac{p_\infty(V) - p}{\tau_p(V)},$$

$$\dot{q} = \frac{r_\infty(V) - r}{\tau_q(V)},$$

$$p_\infty(V) = \frac{1}{1 + exp\left[\frac{-V - V_x - 50}{7.4}\right]},$$

$$q_\infty(V) = \frac{1}{1 + exp\left[\frac{V + V_x + 78}{5}\right]},$$

$$\tau_p(V) = 3 + \frac{1}{exp\left[-\frac{V + V_x + 100}{15}\right] + exp\left[\frac{V + V_x + 25}{10}\right]},$$

$$\tau_q(V) = 85 + \frac{1}{exp\left[-\frac{V + V_x + 405}{50}\right] + exp\left[\frac{V + V_x + 46}{4}\right]}, \qquad (2.8)$$

where, E_{Ca} denotes the reverse voltage and g_L denotes maximum conductance of this channel.

2.3 Models of Synapses

We first discuss the modeling of various synaptic receptors. Through these receptors, the postsynaptic part of a synapse receives signals and carries out its functionality. Each synapse transmitting excitatory signals has two types of receptors in the postsynaptic part: AMPA and NMDA. Similarly, each synapse transmitting inhibitory signals also has two types of receptors: $\mathrm{GABA_A}$ and $\mathrm{GABA_B}$. In the model used in this work, each synapse transmitting excitatory signals has one AMPA and one NMDA receptors. Similarly, each synapse transmitting inhibitory signals has one $\mathrm{GABA_A}$ and one $\mathrm{GABA_B}$ receptors. For each compartment, the total synaptic current is simulated as (Izhikevich & Edelman, 2008)

$$I_{syn} = n_{AMPA} \cdot g_{AMPA} \cdot (V - 0) + n_{NMDA} \cdot g_{NMDA} \cdot \frac{(v + 80)^2}{60 + (v + 80)^2} \cdot (V - 0) + n_{GABA_A} \cdot g_{GABA_A} \cdot (V + 70) + n_{GABA_B} \cdot g_{GABA_B} \cdot (V + 95), \quad (2.9)$$

where n_{AMPA}, n_{NMDA}, n_{GABA_A} and n_{GABA_B} are the number of AMPA, NMDA, $\mathrm{GABA_A}$ and $\mathrm{GABA_B}$ receptors, respectively. We describe the dynamic of g_{AMPA}, g_{NMDA}, g_{GABA_A} and g_{GABA_B} as α function when receiving an input signal at time $t = 0$.

$$g = \frac{t}{T} \cdot e^{-\frac{t}{T}}, \quad (2.10)$$

where the values of T for AMPA, NMDA and $\mathrm{GABA_A}$receptors are 5 ms, 150 ms and 6 ms, respectively. The model of the dynamics of conductance g_{GABA_B} is different from that of the other three. The response of g_{GABA_B} to a signal at time t = 0 is described by the following equations (Rubchinsky, Kopell, & Sigvardt, 2003):

$$f(t) = \begin{cases} 0.5\,mM & 0\,ms \le t \le 0.3ms \\ 0\;\; nM & t < 0\,ms \;or\; t > 0.3\,ms' \end{cases}$$

$$\frac{dr}{dt} = k_1 \cdot f(t) \cdot (1 - r) - \; k_2 \cdot r,$$

$$\frac{dG}{dt} = k_3 \cdot r - \; k_4 \cdot G,$$

$$g_{GABA_B} = \frac{G^4}{G^4 + k_d}, \tag{2.11}$$

where $k_1 = 0.5\ \mathrm{mM}^{-1}$, $k_2 = 0.0012\ \mathrm{ms}^{-1}$, $k_3 = 0.18\ \mathrm{ms}^{-1}$, $k_4 = 0.034\ \mathrm{ms}^{-1}$ and $k_5 = 100\ \mu\mathrm{M}^{-1}$.

2.4 Model of the Network

In most existing neural network simulation projects, researchers only choose the number of each kind of neurons and connect them randomly no matter what their locations are (Nageswaran et al, 2009). However, in this work, we use a more detailed model to describe the structure of the cortex and thalamus. As illustrated by Figure 4, the brain is divided into as many as 70 regions in two hemispheres. Each region has a cortex part and a thalamus part. There are two kinds of connections in this network: local connections and global connections.

The local connections are the connections inside each region. When the dendrite of a neuron locates inside the axon of another soma, there is a possibility of a local connection between them. The local cortical circuitry is adopted from the cat area 17 (Binzegger, Douglas, & Martin, 2004). The axon structure of each kinds of neuron is based on the research in (Izhikevich & Edelman, 2008).

On the other hand, the global connections are the cortico-cortical connections between different regions. Biologists have published the data based on diffusion-magnetic resonance imaging (MRI). An adjacency matrix is generated to describe the global connectivity (Zalesky & Fornito, 2009).

As illustrated in (Izhikevich & Edelman, 2008), the signal speed for myelinated fibers in whiter matter is about 1m/s. Then based on the coordinate of each neuron, distance between connected neurons and the delays of transmitted signals are calculated.

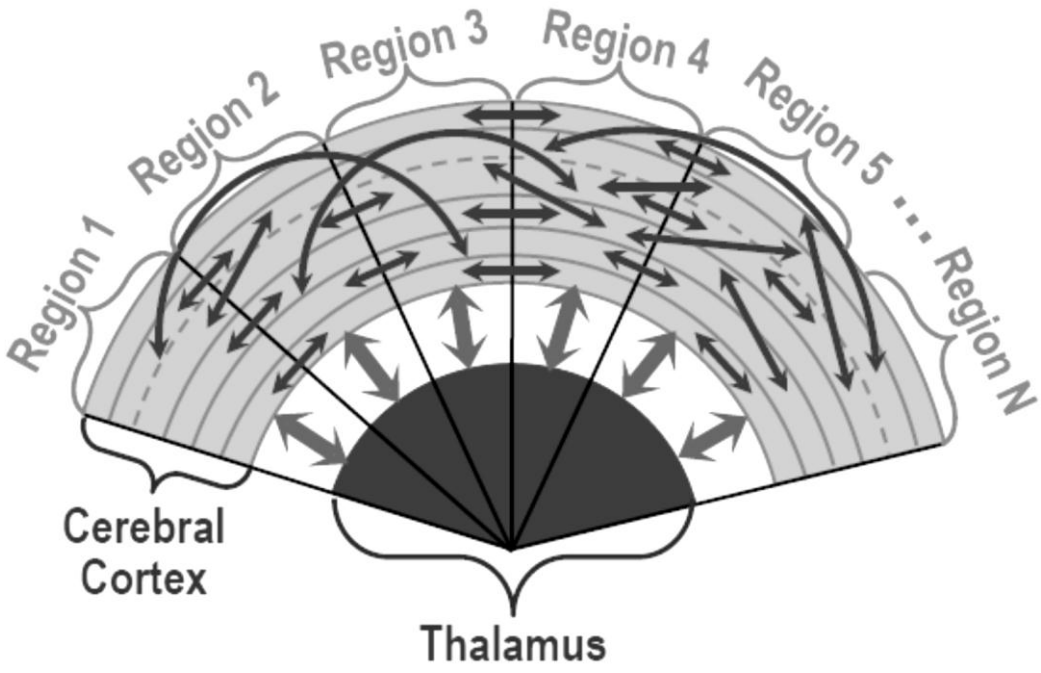

Figure 4 Structure of cerebral cortex and thalamus.

In this work, different sizes of networks are constructed for simulation. When scaling down the number of neurons, the number of synapse inputs to each compartment and the density of neurons are kept constant. That means only scaling down the area of

the cerebral cortex and the total number of synaptic connections for the entire network with the total number of neurons.

2.5 Numerical Integration Methods

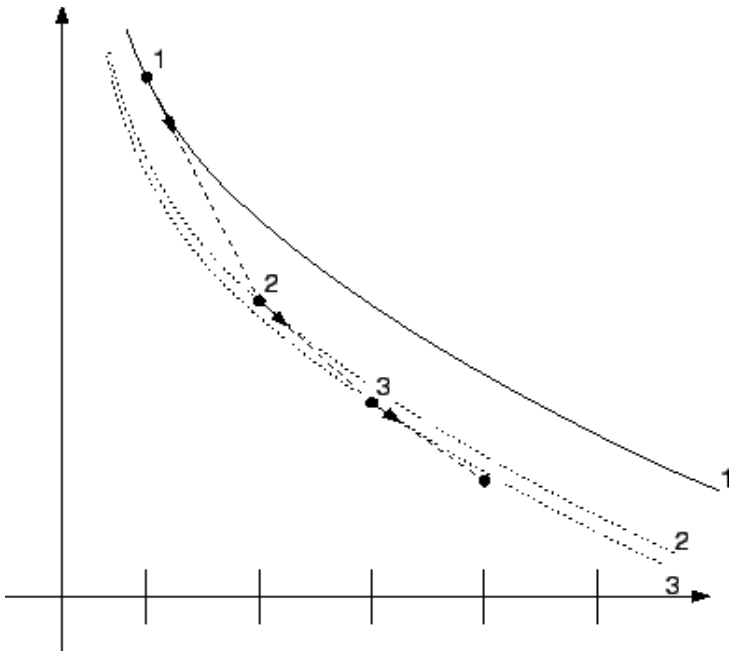

Figure 5 Forward Euler.

Most of the computation in our work is concentrated on differential equations solving. There are many different numerical integration methods to solve this problem. Forward Euler, Backward Euler and Trapezoidal are very popular ones and these methods can be classified into several categories: explicit or implicate integration methods (Pillage, Rohrer, & Visweswariah, 1995). In our simulator, explicit integration methods are used for integration for the following reasons. Compared with implicit

integration, explicit methods not only decrease iterations for each time step, but also avoid solving matrix equations (the runtime usually grows super-linearly with the size of the problem or linearly with a large coefficient). These advantages may potentially accelerate the simulation.

First, Forward Euler method will be illustrated (Figure 5). Given an differential equation

$$\dot{y(t)} = f(y(t), t), \qquad y(t_0) = y_0. \tag{2.12}$$

Assuming step size is h, the Forward Euler method defines y_{n+1} as

$$y_{n+1} = y_n + h \cdot f(y_n, t_n) ,$$

$$h = t_n - t_{n-1}. \tag{2.13}$$

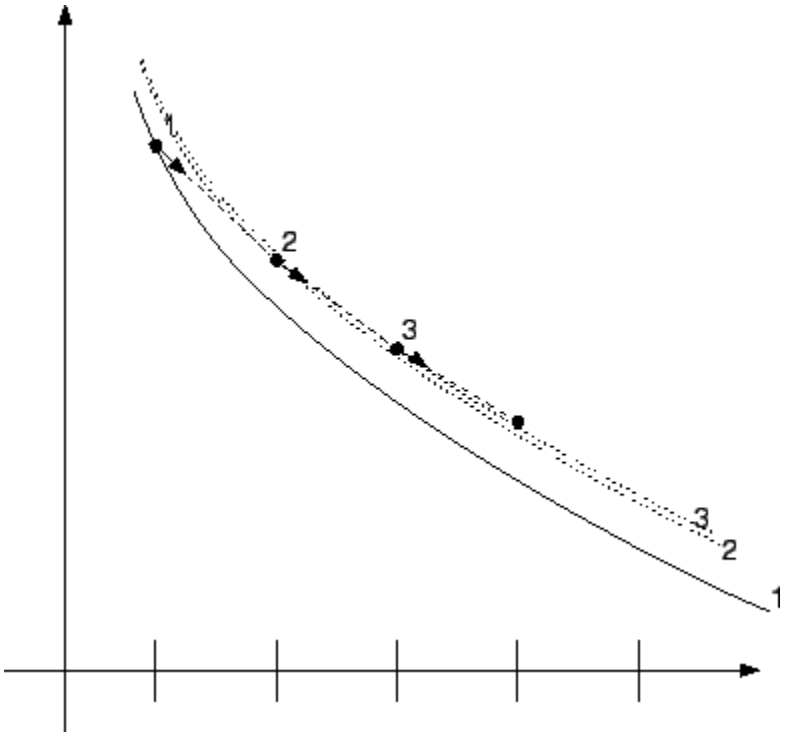

Figure 6 Backward Euler.

So, this method is a first order technique. Also we may easily find that Forward Euler method is an explicit method. From the known value of y_1 and $f(y_1, t_1)$, we can explicitly get y_2. However, the disadvantage is the stability problem caused by large time steps. If we expand Taylor series at t_n, we get

$$y_{n+1} = y_n + h \cdot f(y_n, t_n) + O(h^2). \tag{2.14}$$

Local truncation error (LTE) at each time step will scale with h^2.

Backward Euler method computes y_{n+1} as

$$y_{n+1} = y_n + h \cdot f(y_{n+1}, t_{n+1}). \tag{2.15}$$

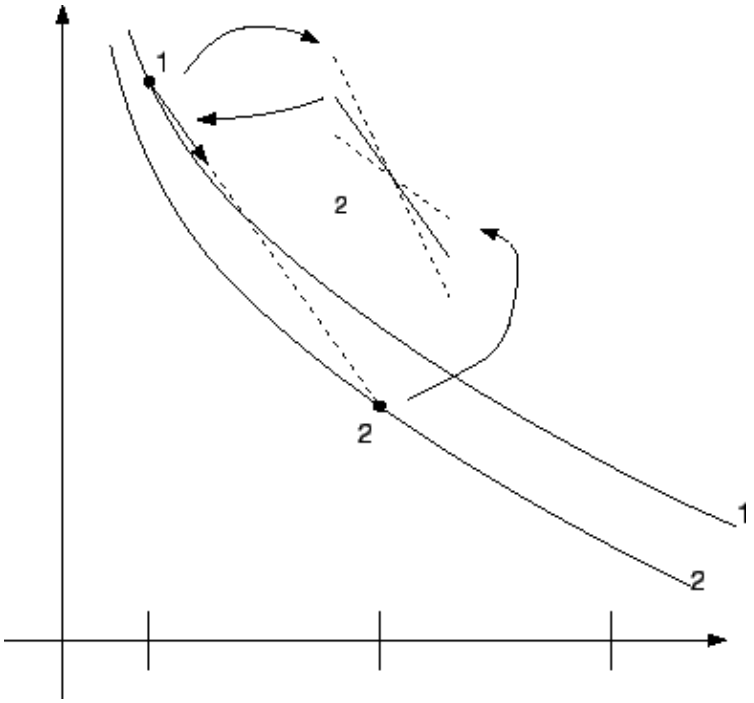

Figure 7 Trapezoidal rule.

It is an implicit method, because (y_{n+1}, t_{n+1}) is not known when computing y_{n+1}. So we have to solve a non-linear equation to find y_{n+1}, most of time, Newton-Raphson method is used. Evidently, the above procedure consumes more computation than Forward Euler method. Similarly, the LTE is also scaled with h^2. But the stability of this method is better, because it always undershoot the original curve as describe in Figure 6.

Similarly, the trapezoidal method is an average of the Forward Euler and Backward Euler and is also an implicit method. y_{n+1} is computed as:

$$y_{n+1} = y_n + h \cdot \frac{(f(y_{n+1}, t_{n+1}) + f(y_n, t_n)}{2}. \qquad (2.16)$$

And the LTE scales with h^3.

In this work, we choose standard forward Euler as our method with a sufficient small step size to guarantee the stability and accuracy. Based on some experiments, the step size need to be as small as 0.01 ms in our basic implementation. In addition, since we adopt multi-compartment HH type models for each neuron and employ large network models consists of up to 1 million neurons, considerable computational effort is needed for each step of simulation. Under this circumstance, the computing power becomes a bottleneck for carrying out the simulation. For a large neural network as large as millions of neurons, solving nonlinear equations using Newton method will cost tremendous computing power. So it is a good choice to use explicit method, Forward Euler. In case of stability problem, we use a small enough time step size in the simulation.

2.6 Parallel Computing

With the increasing complexity of practical problems, parallel computing platforms are playing an increasingly significant role in many areas of research. In recent years, many multicore platforms with different architectures have been proposed by processor manufactures (Grama, Gupta, Karypis, & Kumar, 2003).

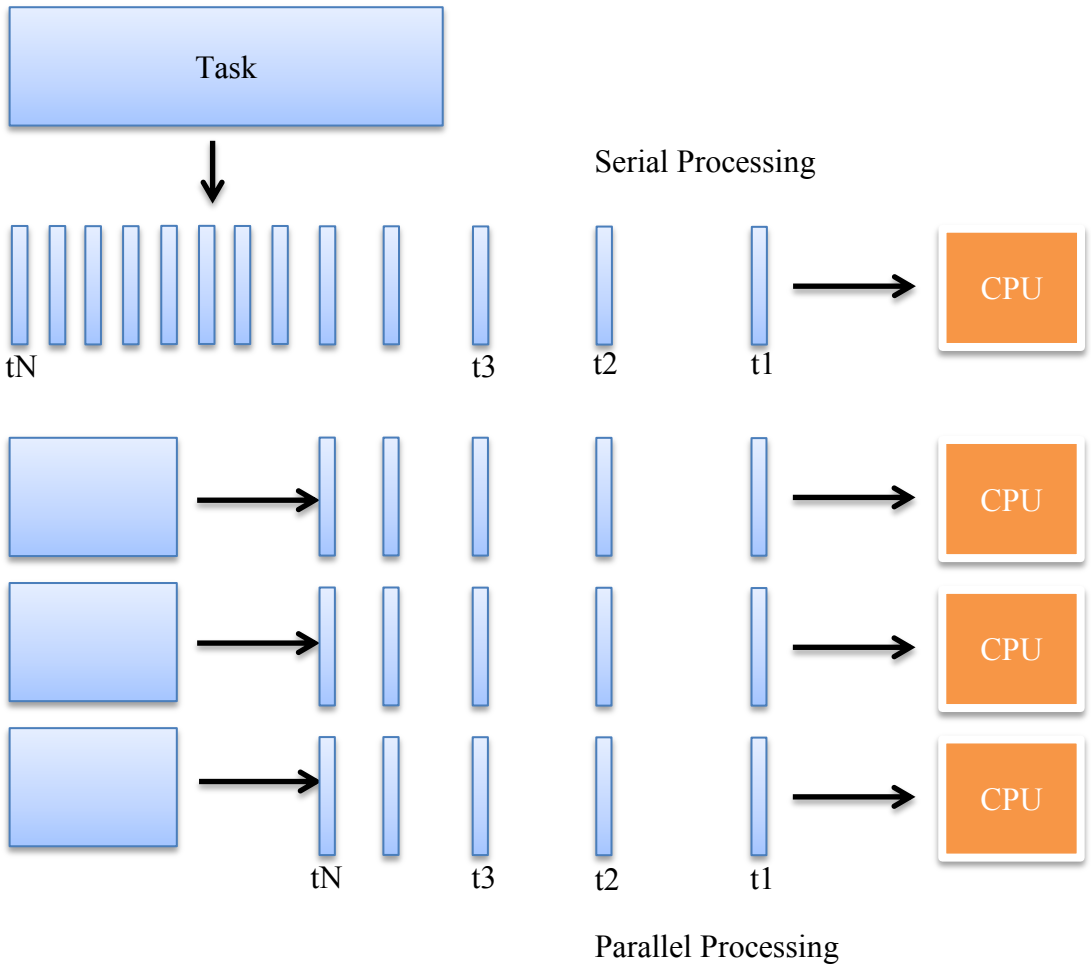

Figure 8 Comparison between serial processing and parallel processing.

As described in Figure 8. In serial computing, instructions will be executed one after another on a single processor. While for parallel computing, the problem is partitioned into discrete smaller tasks which will be executed simultaneously on multiple processors. Generally, there is a certain speed up using parallel computing considering the less workload for a single processor.

Currently, there are three kinds of parallel platforms according to their different memory models: shared-memory platform, distributed-memory platform and hybrid platform. For shared-memory platforms, all processors share the same memory. For distributed-memory platforms, different memories are separated to each processor. Unlike above two platforms, in Hybrid platforms, there are both districted-memory model among nodes and shared-memory model among processors in each node (Barney, 2012b).

Correspondingly, there are three types of parallel programming models for these three platforms. For instance, Message Passing Interface (MPI) is a library based programming model on distributed memory platforms (Barney, 2012b). Each processor has its own memory and they communicate by sending or receiving messages. There are two types of communication routines: point to point and collective routines; While OpenMP and Pthreads are commonly used in shared memory platforms (Grama, Gupta, Karypis, & Kumar, 2003).

Pthreads is a low-level API for working with threads. Thus, programmers have fine-grained controls in thread management (create/join/etc.), mutexes, and so on. On the other hand, OpenMP is at a much higher level and more easily to be used than Pthreads

(Barney, 2012b). One specific example is OpenMP's work-sharing constructs, which makes dividing work across multiple threads straightforward.

Table 2 Comparison between MPI and OpenMP.

	MPI	OpenMP
Cons	1. Requires more programming changes to go from serial to parallel version 2. Harder to debug 3. Performance is limited by the communication network between the nodes	1. Can only be run in shared memory computers. 2. Requires a compiler that supports OpenMP 3. Mostly used for loop parallelization
Pros	1. Runs on either shared or distributed memory architectures 2. Used on a wider range of problems than OpenMP 3. Each process has its own local variables	1. Easier to program and debug than MPI 2. Directives can be added incrementally - gradual parallelization 3. Serial code statements usually don't need modification 4. Code is easier to understand and maybe more easily maintained

Also, there is another Hybrid Model which combines more than one of the previous models. Currently, the combination of Message passing model and the threads model (OpenMP) are widely used. By taking the hybrid model, multiple threads are generated inside each MPI node while MPI controls the communication between different nodes.

Considering the practical problem of our work, we choose MPI and Hybrid MPI/OpenMP models to simulate the neural network. Because the work is similar in each MPI node, there are many "for" loops which may be parallelized efficiently by OpenMP. The following two sections will introduce some important fundamentals of MPI and OpenMP which will be used in our simulation.

2.6.1 MPI communication routines

When using MPI, programmers concentrate on how to manage communications between different processors. There are totally two types of communication routines: point to point routines and collective communication routines (Barney, 2012b).

MPI point to point routines usually involve communicates between only two processors (Figure 9). While one processor sends the data, the other processor perform the receive operation. Also in each processor, there is a system buffer. When the current processor receives multiple inputs, the data will be stored in the buffer. There are two kinds of point to point routines: Blocking and Non-blocking. For blocking mode, the program will not return until send or receive is completed; while for non-blocking mode,

no matter whether send and receive routines have been completed, program will be executed normally without waiting.

However, collective communication routines are used to conduct a global operation among processors in the network. For example, the Figure 10 illustrates that the local data in one processor can be scatted to the other processors through the MPI Scatter routine. In this work, connections between neurons are complex and there are thousands of connections for each neuron. Most of time, one neuron's postsynaptic neurons are located in almost all the other processors. If point to point routines are adopted, communications are hard to manage and cost more time. So, collective communication routines are chosen in the neural network simulation.

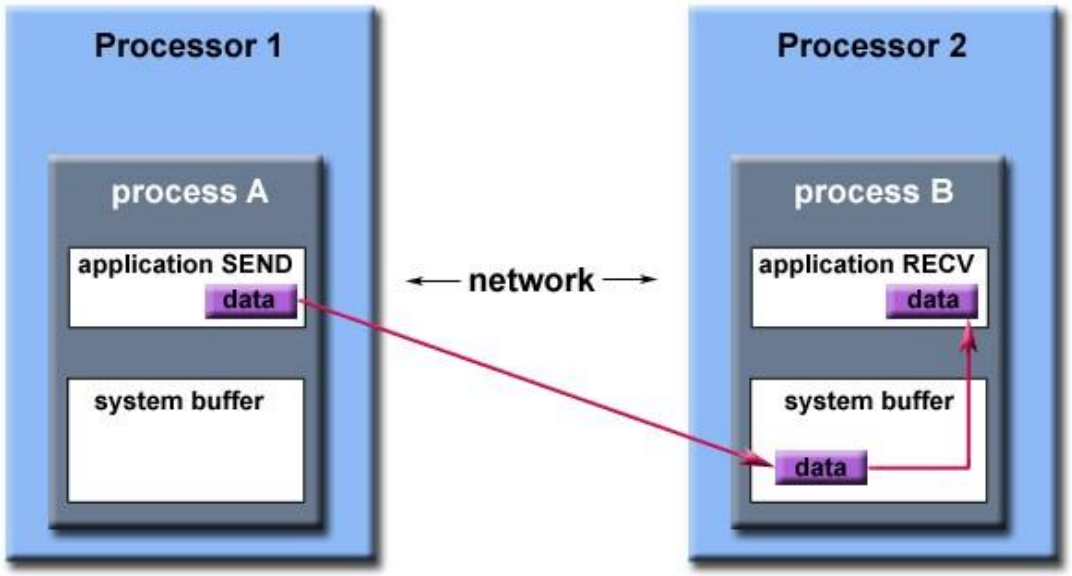

Figure 9 Point to point routine.

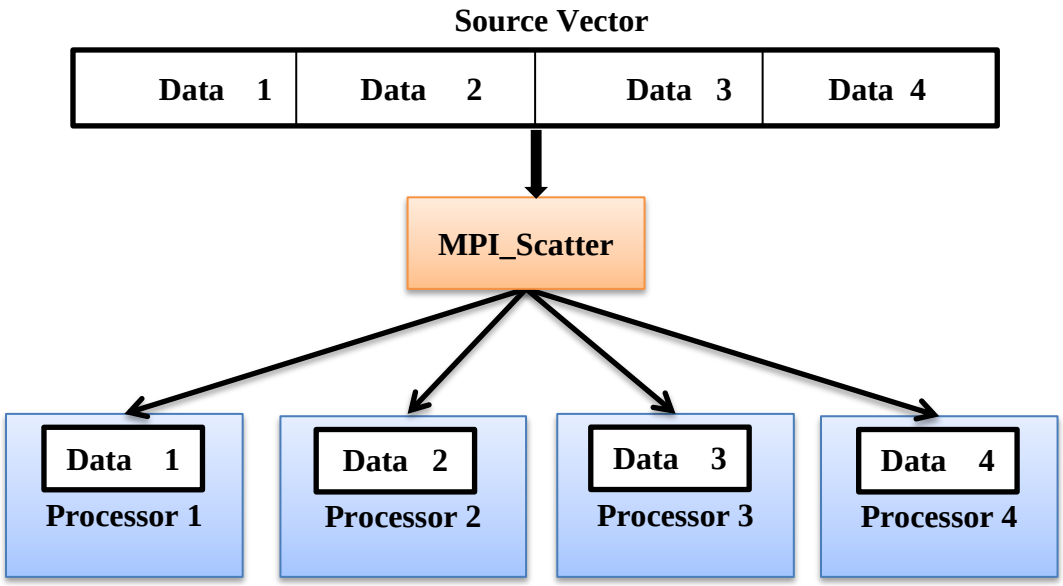

Figure 10 Collective scatter routine.

2.6.2 OpenMP scheduling schemes and "CRITIAL" directive

In OpenMP directives, there are mainly three scheduling schemes to conduct load balancing: STATIC, DYNAMIC and GUIDED scheduling (Barney, 2012a).

For STATIC Scheduling, fixed numbers of loop iterations (block) are assigned to each OpenMP thread statically. The size of the block is equals to $(number\_of\_iterations)/(number\_of\_threads)$. For DYNAMIC Scheduling, loop iterations are assigned one after another dynamically to each thread. When a thread finishes the current iteration, another iteration is assigned to this thread. For GUIDED Scheduling, several Iterations in a block are dynamically assigned to threads when threads request them. Block size is not constant but decreases each time when iterations

are distributed to a thread. The way to calculate the size of the block is described as follows:

$$Size\ of\ the\ initial\ block = (number\_of\_iterations\ )/(2 \times number\_of\_threads)$$

$$Size\ of\ subsequent\ blocks \propto (number\_of\_iterations\_remaining)/(2 \times number\_of\_threads)$$

For a loop with 1000 iterations and 10 OpenMP threads, the initial block size is 50. When a thread finishes current work, 25 iterations are assigned to it. In this way, blocks of iterations will continuously be assigned until no more iteration unprocessed.

In shared memory platforms, race condition is an important problem when designing programs. In OpenMP, a "CRITICAL" directive is used to specify a block of code which can be accessed by one thread at a time. When the "CRITICAL" block is executed by one thread and another thread comes, the later thread is block until the first one finishes its execution.

3. PROPOSED NETWORK SIMULATION TECHNIQUES

3.1 Introduction

Recent years, event-driven simulation method is applied to the simulation of neural networks to improve the computational efficiency. In the work of (D'Haene, Schrauwen, Campenhout, & Stroobandt, 2009), since the integrate and fire neurons with linear models of postsynaptic potential are used, the time evolution of the leaky integrate and fire model of neurons without synaptic inputs is derived analytically. Through exact prediction of the firing time and state update upon arrival of synaptic inputs, the simulation avoids the conventional way of integration over small steps. The event-driven simulation significantly improves the computational efficiency with appropriate computational models. However, the Hodgkin-Huxley type models used in our simulation are more sophisticated. These models integrate complicated nonlinear dynamics of ion channels. Under this circumstance, it is difficult to analytically solve the time evolution of neuron models, thereby hard to predict the exact firing time. Therefore, simulation by integrating over small steps is still necessary.

Moreover, considering that synaptic receptors may exert influence on the postsynaptic neurons as long as 150ms and the firing frequency of a neuron is at least several Hertz, the dynamics of synaptic receptors need to be tracked all the time. Therefore, for biologically realistic network models we target, we cannot adopt event-driven simulation but turn to improve the computational efficiency for continuous

simulation by adopting two techniques: merging linear synaptic receptors and using two level time steps.

3.2 Merging of Linear Synaptic Receptors

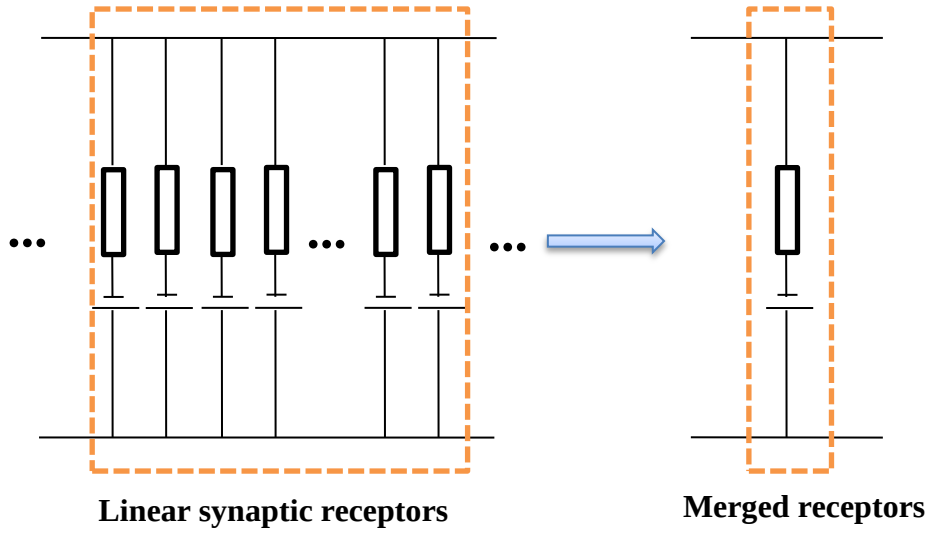

Figure 11 Merging of linear synaptic receptors.

In the synapse model, for each pair of connections, there are corresponding excitatory or inhibitory receptors. On average, more than 30 receptors are located in each compartment. That means we need to solve at least 30 differential equations for each compartment and there are tens of compartments in each neuron.

To reduce the computation of solving differential equations, we may merge the same linear synaptic receptors including AMPA, NMDA, $GABA_A$. So, for each compartment, the conductance of each type of linear receptors can be added together

(Figure 11). Then there are three linear receptors and several nonlinear $GABA_B$ receptors for each compartment. By exploring the linearity of synaptic receptors, the responses of all the inputs applied to the synaptic receptors of the same type on a given compartment are merged mathematically in our implementation, which significantly reduce the cost of computation without sacrificing accuracy.

3.3 Simulation Speedup Considering the Data Locality

In a large scale neuron network simulation, the size of the data is far more than the capacity of the cache. Processors need to load the data to the cache once for every step of integration. Considering the accuracy and stability problem of Forward Euler, the minimum time step is as small as 0.01 ms. Therefore, a large amount of time is consumed on accessing memories.

However, in this work, when the presynaptic neuron fires, it will send the firing message (time delay value) to its postsynaptic neurons. As described in the work of (Izhikevich & Edelman, 2008), the minimum axon delay is 1 ms. The firing message received in a macro-step is at least generated in the previous macro-step. So a two level time steps approach is adopted. Under this approach, a neuron is simulated continuously one hundred time steps (1 ms) without affecting the accuracy of the simulation. In the simulation, there are two kinds of time steps: macro-step and micro-step (Figure 12) and they are 1ms and 0.01 ms respectively.

If neurons are simulated one by one during the macro-step, the cache loads different data every time and may not be utilized efficiently. By taking this approach, the

cache hit rate is increased. Subsequently, the number of times for loading data is reduced. Thus, this technique improves the simulation efficiency.

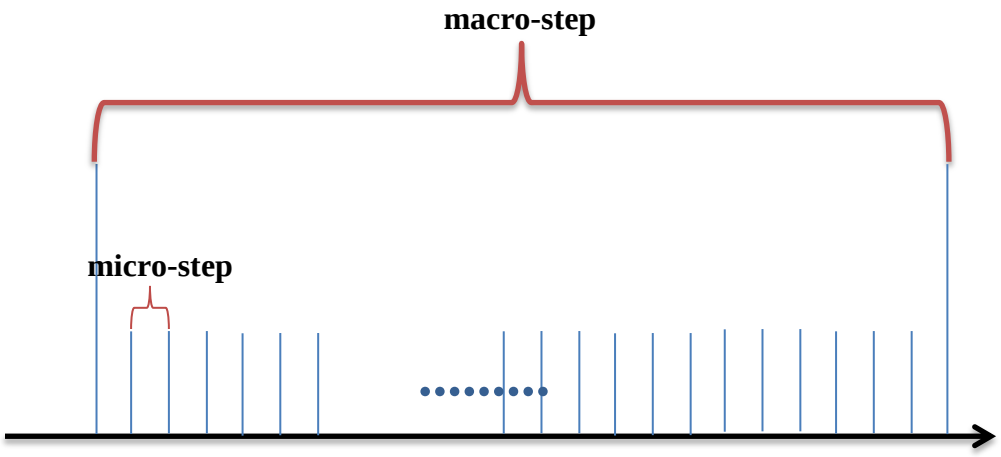

Figure 12 Two level time steps.

4. PARALLEL NEURAL NETWORK SIMULATION METHODS

4.1 MPI Parallelization

In this section, a MPI based approach is presented. There are two features of MPI programming: separated memory space and communication routines. As the result of the distributed memory, a task needs to be partitioned to different places. While works in several processors are dependent, MPI communication routines are used to transmit messages. Generally, by using MPI, different programs can be executed simultaneously on each of the processors. In neural network simulations, similar programs are executed among processors with little differences.

In neural network simulations, first, to make each processor own a similar amount of work, all neurons are partitioned into several small groups with the same size. As describe in chapter 2, different regions share the same local circuitry. For each region, there are same numbers of neurons and connections. The compartment of a neuron is labeled by region ID, neuron type, neuron ID and compartment ID. The whole neurons are partitioned according to the neuron ID which stands for multiple neurons in different regions. For example, in a network of 70 regions, there are 1000 neurons in one region and 10 processors. After the network is partitioned, each processor owns 700 neurons, in which there are 100 neuron IDs and each ID stands for 70 neurons.

Then all processors will simulate the local neural network and solving differential equations by integrate method. After the membrane voltage is calculated, if the value is larger than the thresh hold voltage, a spike is generated. All postsynaptic

neurons need to be informed. However, considering some of the postsynaptic neurons are stored in other processors, there will be several communications between processors.

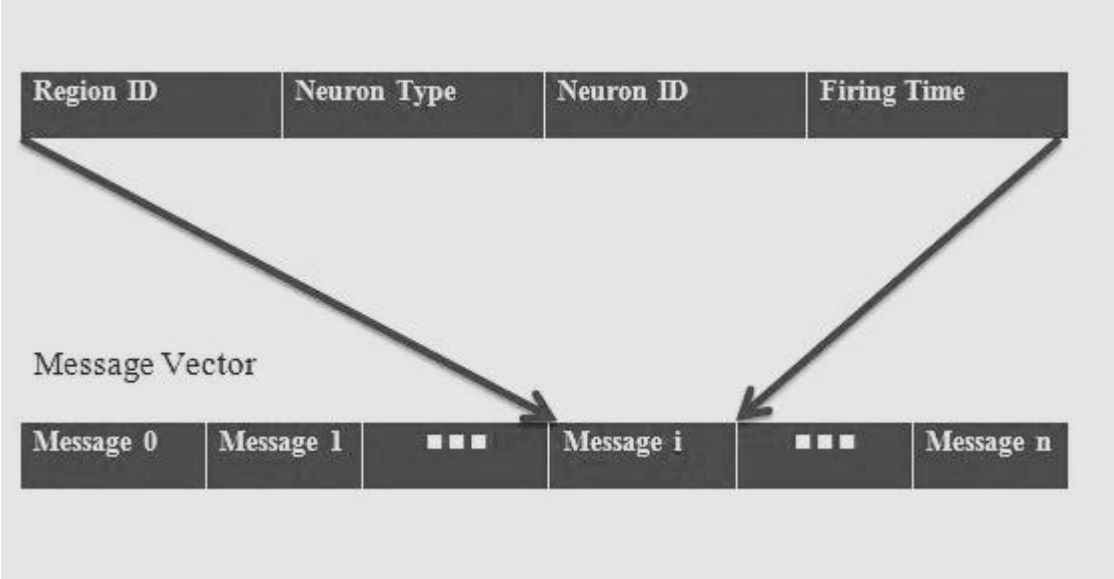

Figure 13 Message vector stored in each processor

A data structure referred as "Message" is designed to store the neuron firing information. As shown in Figure 13, the "Message" consists of Region ID, Neuron Type, Neuron ID and Firing Time of the presynaptic neuron. Once a presynaptic neuron fires, a new "Message" is added into the Message Vector. Normally, many neurons fire at each time step, so the Message Vector will contains lots of "Messages" in each processor. Then, the Message Vector needs to be transmitted to other processors to inform the postsynaptic neurons.

Considering the large complexity of connections between neurons, all postsynaptic neurons are stored in almost all the other processors. If MPI point to point

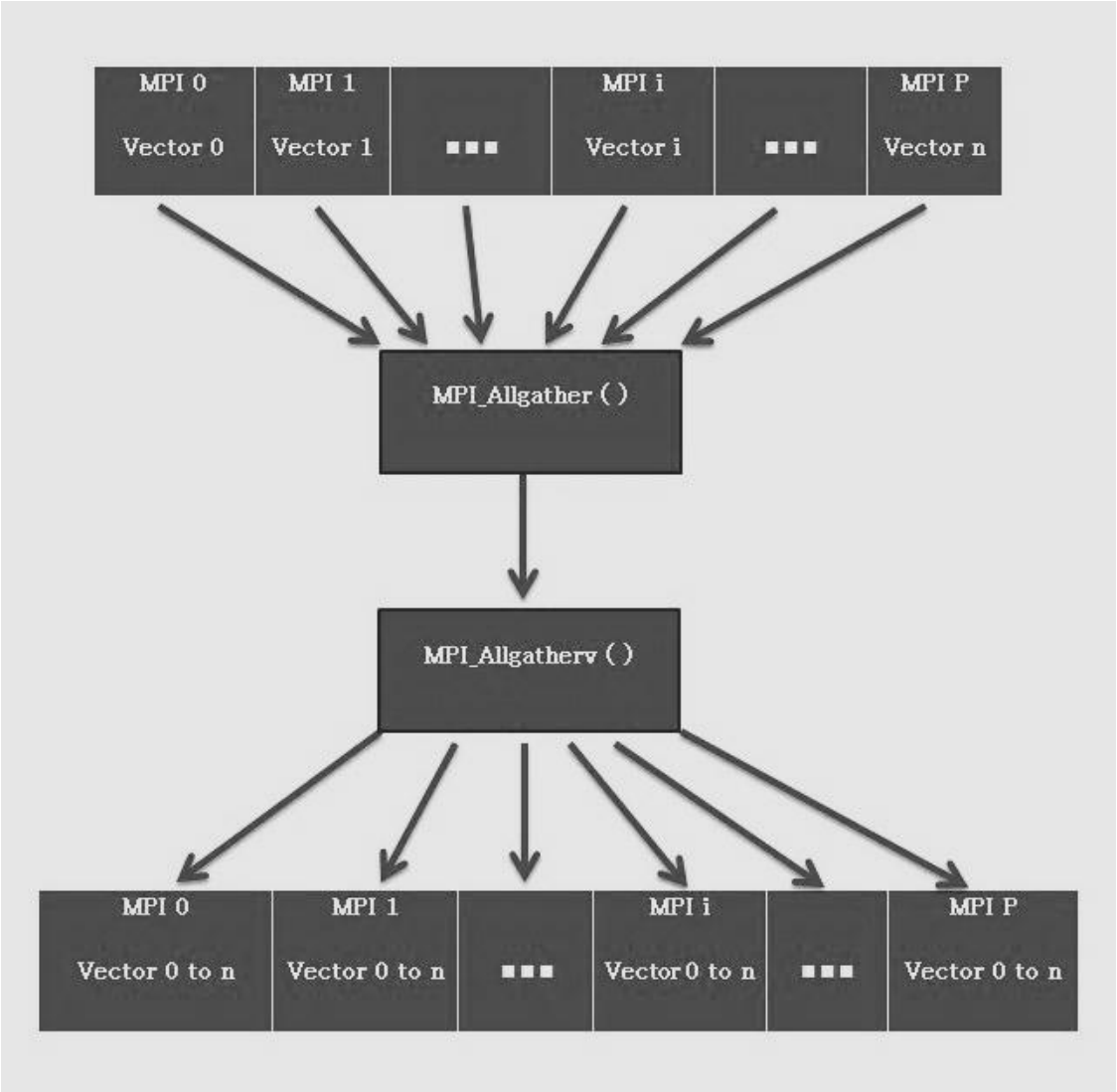

Figure 14 Gathering the entire firing message.

communication routines are used, tremendous rounds of send and receive will be involved, which greatly increase the runtime. So, in this work, we adopt MPI collective communication routines to make the communication process easier and faster. First, use "MPI_AllGather" function to gather the size of each Message Vector and appropriate

size of memory will be allocated to store all the message vectors by using "MPI_AllGatherv" as described in Figure 14.

After each processor gets a copy of the entire firing messages, the local postsynaptic neurons will receive the synaptic input. The whole processor for the parallelization is described in Figure 15.

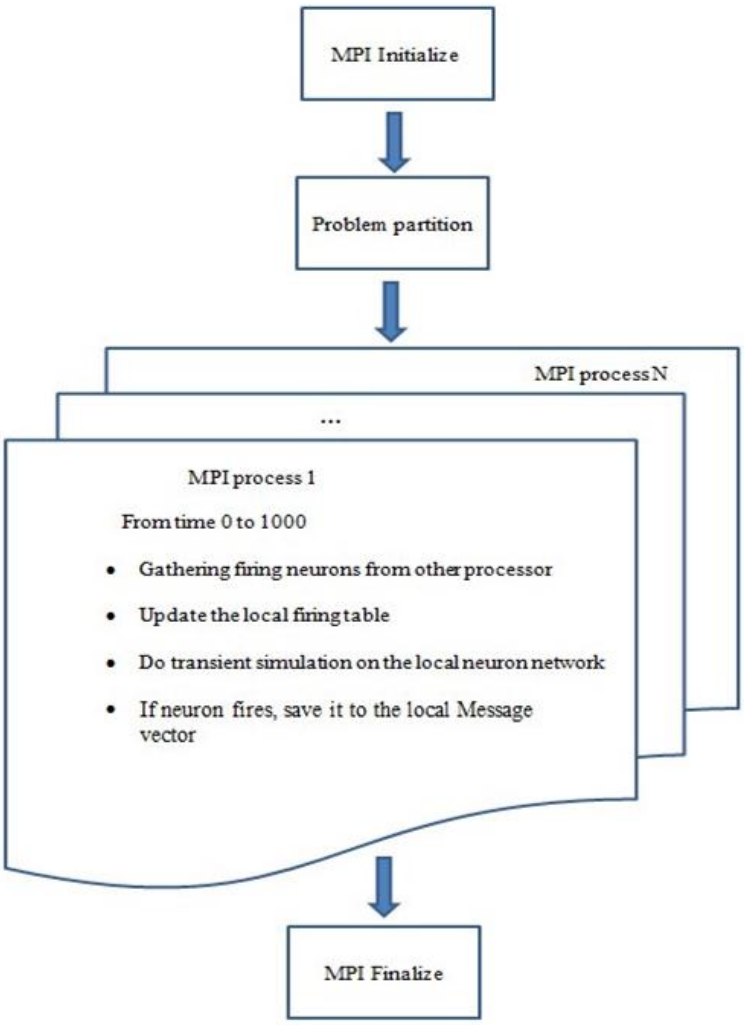

Figure 15 MPI parallel algorithm.

4.2 MPI Parallelization with Dynamic Load Balancing

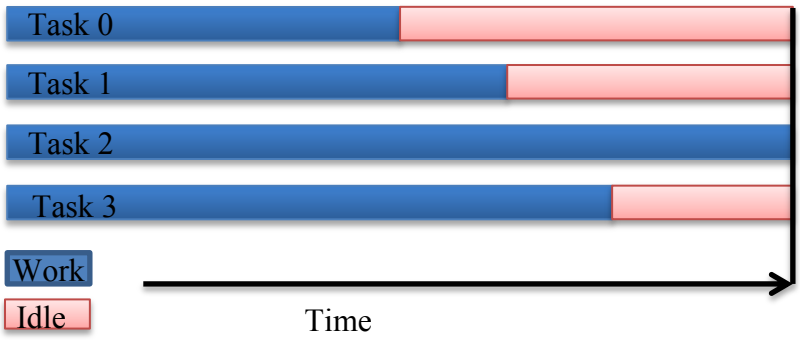

Figure 16 Load balancing problem.

Considering the imbalanced workload in each processor, the efficiency of the neuron network simulator is reduced. As illustrated in Figure 16, while Task 2 is still being executed, the other three tasks have already been completed. However, the slowest task determined the overall runtime. If each processor owns a similar workload, the overall runtime will be shortened. In this approach, two dynamic load balancing schemes are adopted with different work redistributing schemes.

Generally, load balancing is a technique to make workloads in different processors balanced by redistributing the workload. The ideal scheme is to perform load balancing to all the other processors. However, since there are lots of dependencies among different processors, it is not a practical method in neuron network simulation. In this work, a 2D-Torus topology is adopted and each processor communicates with its four neighbors to balance the workload.

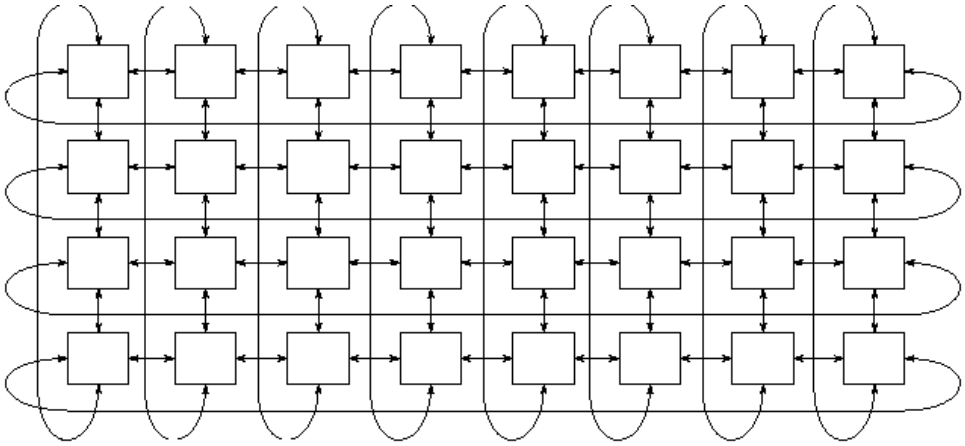

Figure 17 2D Torus.

For the first load balancing scheme (Figure 18), when the central processor P_i completes the local work, it will ask the neighbors to stop and calculate the average number of neurons which have not been simulated. If the neighbor has more than the average number of neurons unprocessed, it will send the excess work to the P_i; while if the neighbor has less work than the average, P_i will send appropriate number of neurons to it. Finally, these five processors will have balanced workload.

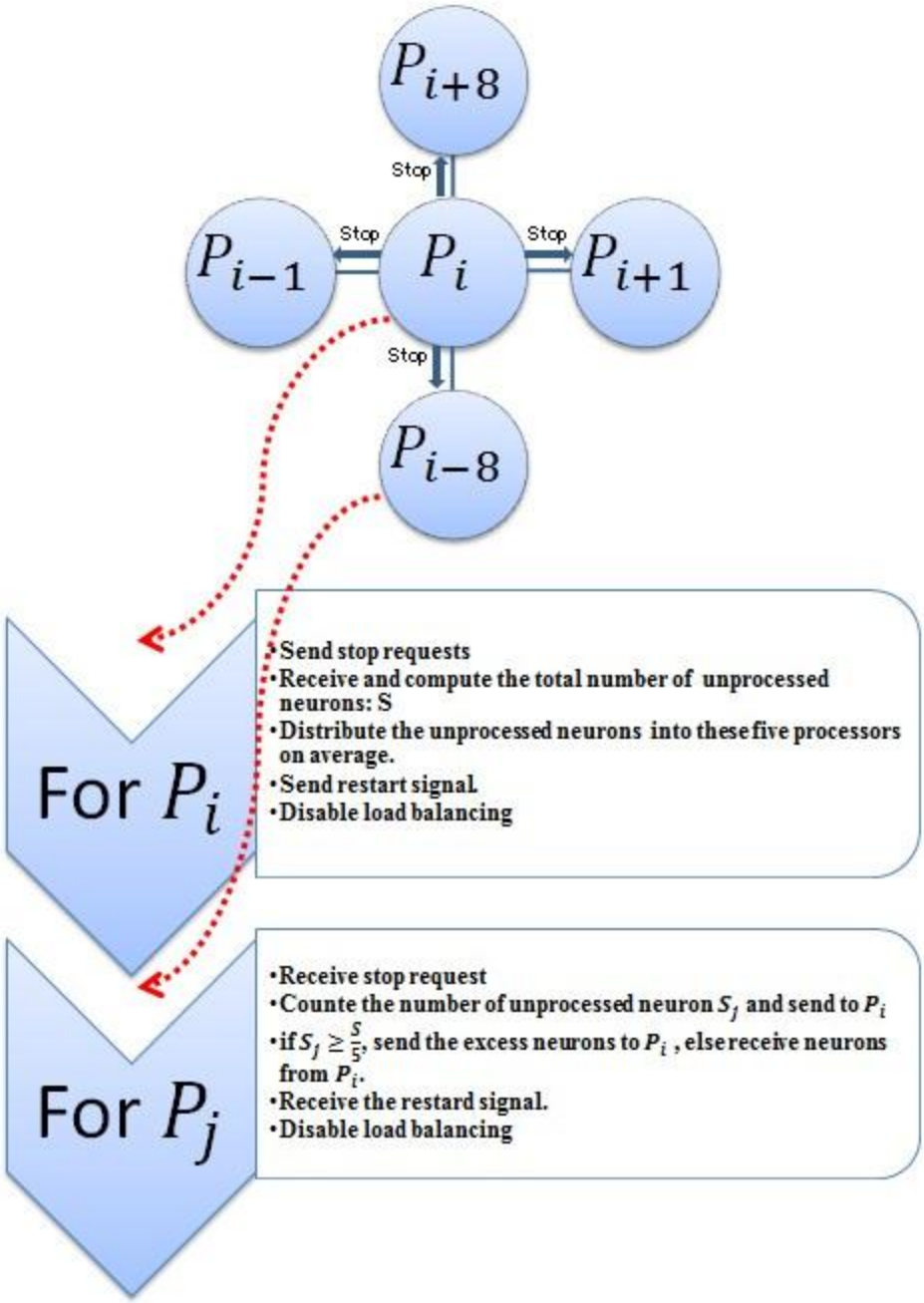

Figure 18 Dynamic load balancing scheme 1.

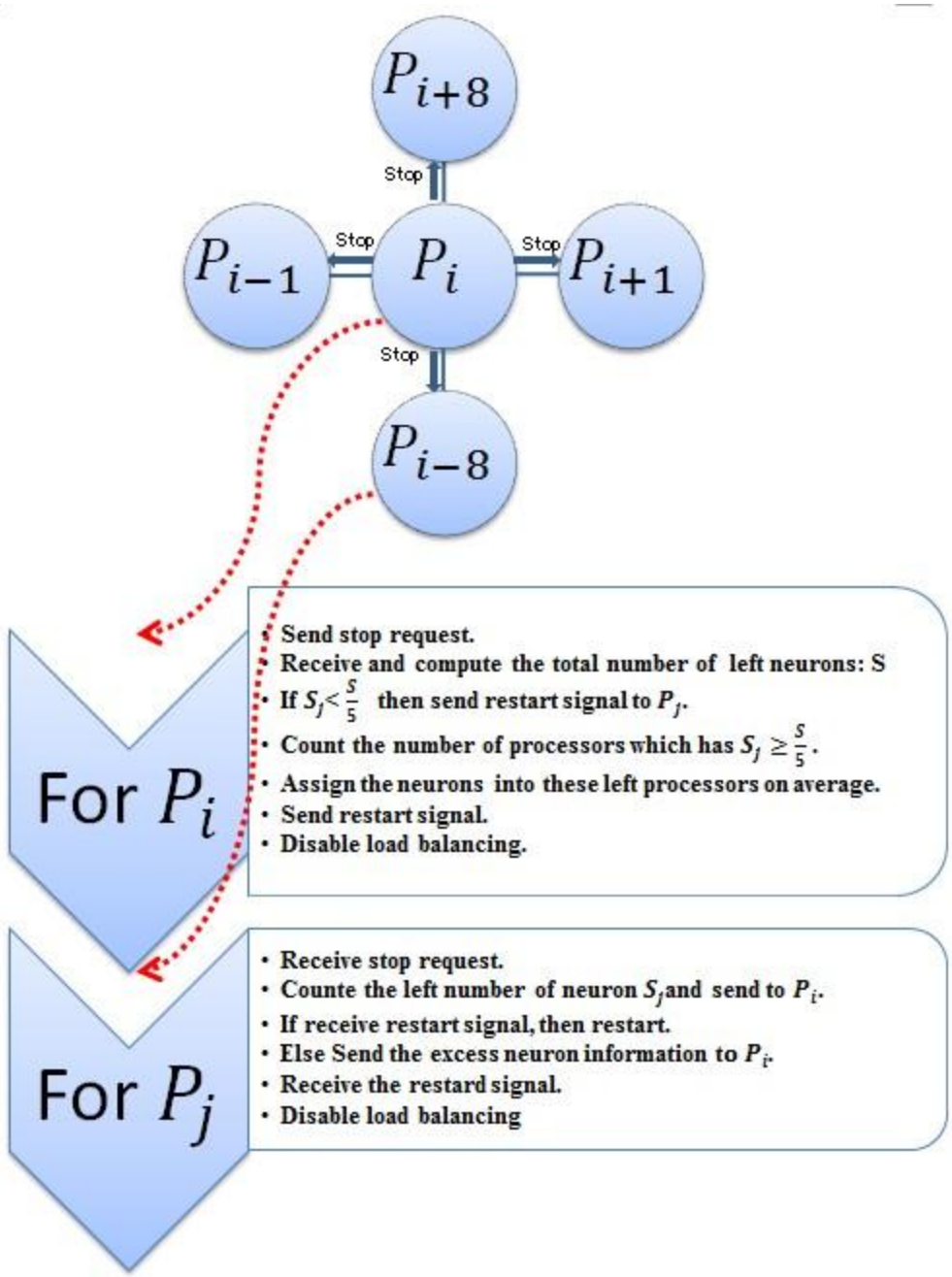

Figure 19 Dynamic load balancing scheme 2.

Moreover, considering that the central processor not only receives but also sends workloads, which greatly limits the efficiency of the simulation, the second load balancing scheme is proposed as shown in Figure 19. Under this scheme, the central processor P_i only distributes work to part of its neighbors and balances the workload among processors only once. First, after calculating the average unprocessed numbers of neurons, P_i only does load balancing with those neighbors which contain more than the average number of neurons. So the central processor only needs to receive work from other processors and the communication between processor becomes easier.

4.3 Hybrid MPI/OpenMP Parallelization

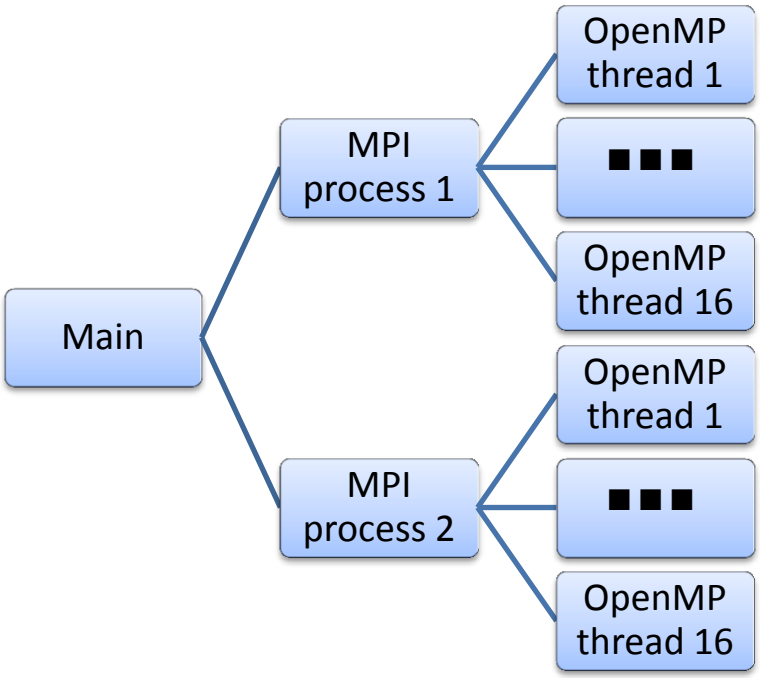

Figure 20 Structure of hybrid MPI/OpenMP parallelization.

By conducting simulations using above two MPI approaches, experimental results indicate that communications between different processors and the redistribution of workload consume most of the computational power. Therefore, a hybrid MPI/OpenMP parallelization approach is proposed with good load balancing schemes and reduced memory traffics.

In the hybrid MPI/OpenMP simulation, multiple OpenMP threads are generated inside each MPI process. Compared with the MPI implementation, it is much easier to change the serial code to parallel one using OpenMP: just add OpenMP "parallel" directives before the "for" loop.

As mentioned in chapter 2, there are three scheduling algorithms: Static, Dynamic and Guided. Experimental results show that based on the 105K neural network with 2 MPI processes and 8, 16 and 32 OpenMP threads; Dynamic scheduling scheme is more suitable for the large neural network simulation (Figure 27). In the following experiments, Dynamic scheduling scheme is chosen as the default one.

The main procedure of the hybrid MPI/OpenMP is similar as MPI approach. After the network is partitioned, a number of neurons are ready to be simulated in each MPI process. At each time step, the Dynamic scheduling scheme is adopted to process these neurons. The only special part is when the postsynaptic receives the synaptic input. If one compartment receives multiple synaptic inputs simultaneously, the race condition problem happens. It is a common problem in shared memory programs. In our program, A "CRITIAL" Directive is used to protect the simulation from race condition problems.

5. EXPERIMENTAL RESULTS AND ANALYSIS

5.1 Experiments Setup

In this work, we use a 52-node, 832-processor IBM Cluster-1600 system to simulate different size of neural networks (Texas A&M Supercomputing Facility, 2012). There are 16 IBM's 1.9GHz RISC Power5+'s in a node and these 16 Power5+ processors have a shared memory of 32 gigabytes. The message passings between processors are implemented by IBM high performance communication switch.

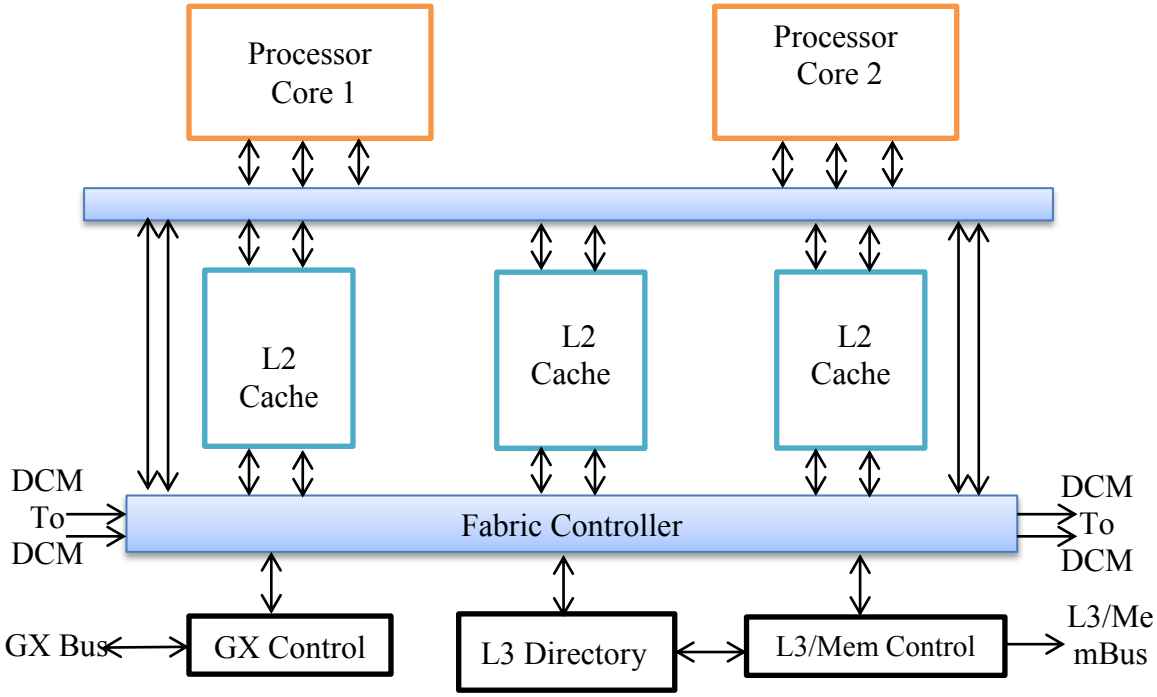

Figure 21 Architecture of one node in the supercomputer.

Three neural networks simulations are presented with different sizes of the network: 30K, 60K and 105K neurons. In the following experiments, first, Figure 22 shows the transition from delta oscillation to epilepsy, which validates our simulator. Later, the efficiency of the proposed three parallel simulation approaches including MPI parallelization, MPI parallelization with dynamic load balancing and hybrid MPI/OpenMP parallelization are demonstrated.

5.2 Simulation of Delta Oscillation and Epileptic Activities

From recent studies in vitro and vivo (Dossi, Nufiez, & Steriade, 1992) (Leresche et al, 1991), researchers get in-depth understanding about the delta oscillation of thalamocortical neurons within the frequency range of 1-4 Hz. Normally, this wave is caused by thalamic relay neurons as the result of low-threshold Ca2+ current I_T and hyperpolarization activated current I_h. The mechanism of the delta activity is described as follows. The rebound burst is caused by the slow activation of I_h after a long lasting hyperpolarization of thalamic cells. Gradually, the burst is deinactivated by the hyperpolarization mediated by I_T. As a result of the inactivation of I_h and I_T during the burst, the membrane voltage turns hyperpolarized after burst termination. (Bazhenov & Timofeev, 2006).

In this experiment, we simulate the 105K neurons network. In the first second, it shows the 3 Hz delta wave. At the end of the first second, the firing pattern changes to a slower oscillation at about 4 Hz with large amplitude negative spikes as the synaptic

$GABA_B$ receptors are suppressed. Two second later, delta oscillation is resumed gradually when the $GABA_A$-mediated inhibition comes back to normal condition.

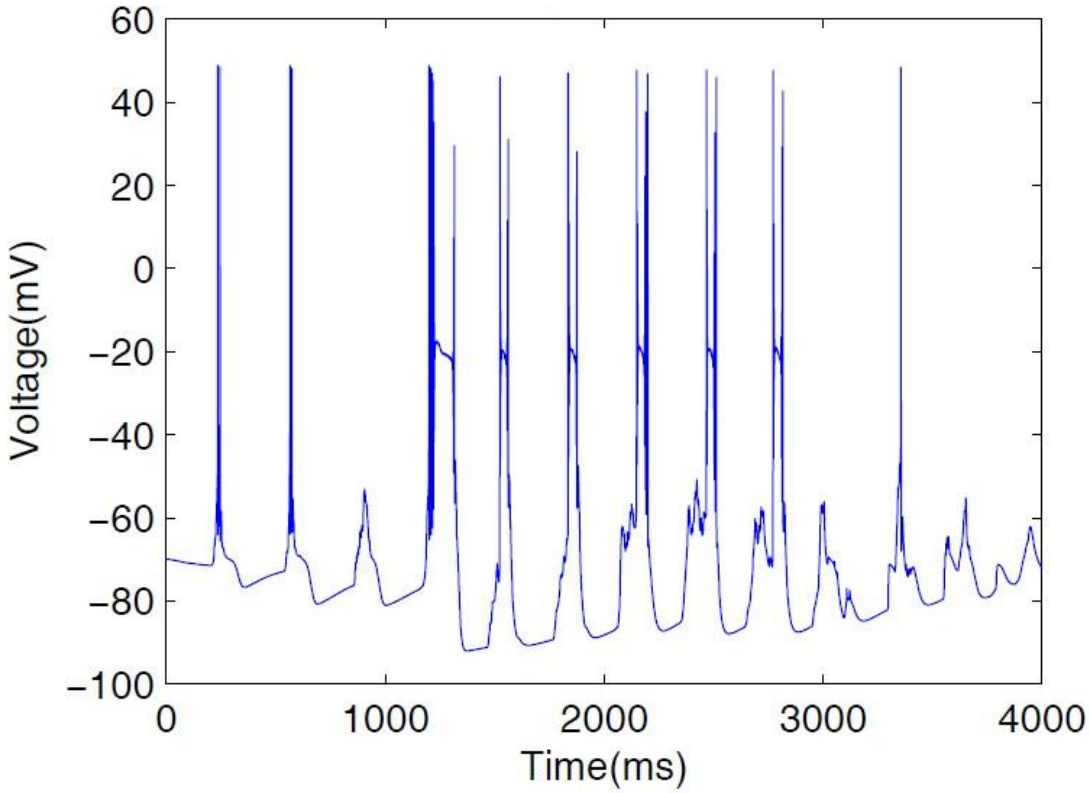

Figure 22 Delta wave and epilepsy.

5.3 MPI Parallelization

In this part of the experiment, we set the total simulation time as 1 minute and use 1, 2, 4, 8, 16 and 32 MPI processes with different sizes of the neural network.

When analyzing a parallel program, it is important to study the speedup and efficiency (Grama, Gupta, Karypis, & Kumar, 2003). They are described as

$$\text{Speedup(nPprocs)} = \frac{\text{Serial runtime}}{\text{Parallel computing time using nPROCS processors}},$$

$$\text{Efficiency(nPprocs)} = \frac{\text{Speedup}}{\text{nProcs}}. \quad (5.1)$$

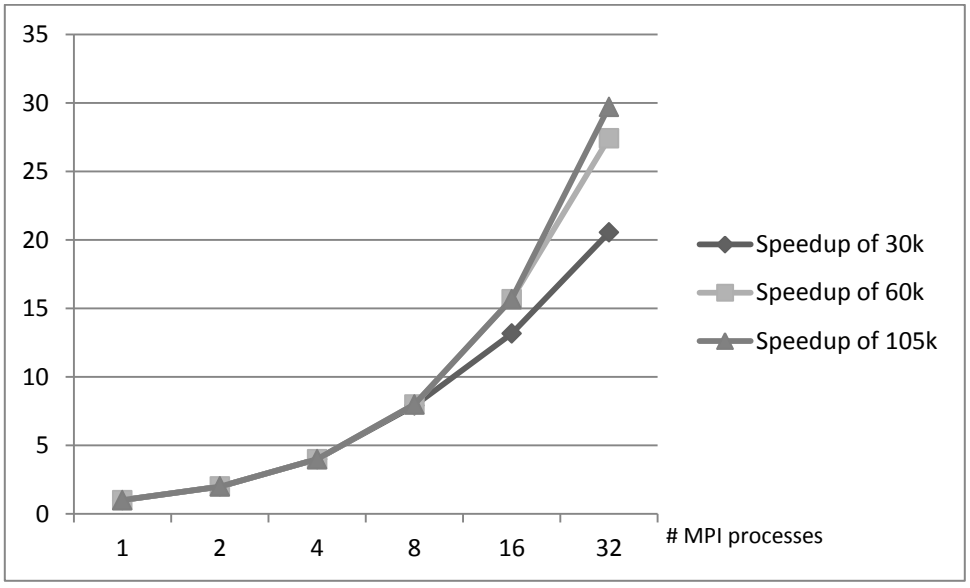

Figure 23 Speedup of the MPI implementation.

In parallel computing, we also call this efficiency as Strong Scaling (Barney, 2012b). A program is considered to scale linearly if the speedup is equal to the number of processing elements used. In general, it is difficult to achieve good strong scaling when the number of the processors is very large since the communication overhead plays an important role. As illustrated in Figure 23 and Figure 24, the speedup and efficiency are very good. However, we find that the workload for each process is unbalanced as the result of the complex network connection. When analyzing the parallel load balancing

problem, imbalance ratio is generally used (Lan, Taylor, & Bryan, 2002), generally, the imbalance ratio is described as:

$$\sigma = \frac{Max - Avg}{Avg}, \tag{5.2}$$

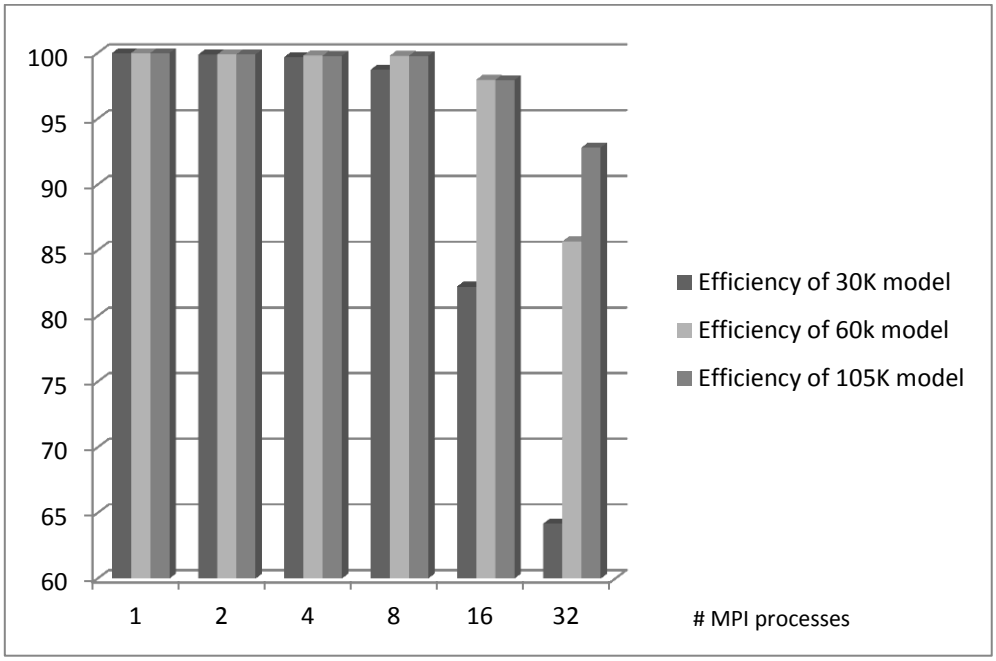

Figure 24 Efficiency of the MPI implementation.

where Max is the maximum running time of all processes and Avg is the average running time. From Figure 25, we observe that for 32 MPI processes implementation, the imbalance ratio is more than 10%. The imbalanced workload became a critical problem especially when using more than 8 MPI processes.

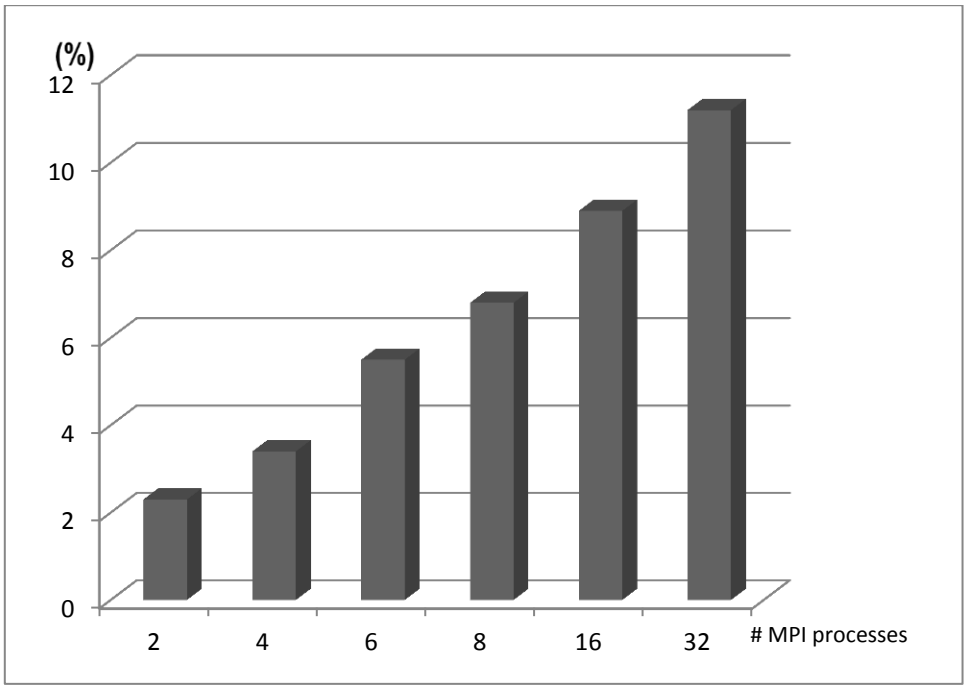

Figure 25 Imbalance ratio of the MPI implementation.

5.4 MPI Load Balancing Schemes

In this experiment, the performance of two proposed load balancing schemes are compared. Experimental result shows that the second scheme is much better than the first one (Figure 26). Because when using the second scheme, the central processor only receives work from other processors; while for the first scheme, the central processor not only receives work but also sends excess work to other processors which cause more running time.

Moreover, when using the second load balancing scheme, the imbalance ratio is reduced and each processor has balanced workload. However, the speed up is still lower than the MPI implementation and the simulation consumes more computation time as the result of the large number of communications between processors.

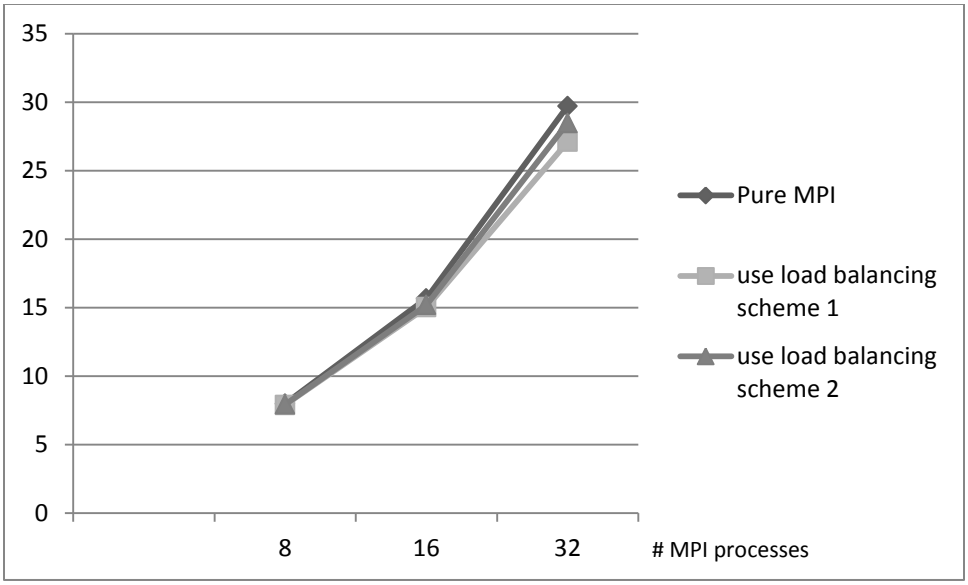

Figure 26 Comparison of speed up using different load balancing schemes.

5.5 Hybrid MPI/OpenMP Parallelization

Finally, an experiment to compare all the three methods: MPI parallelization, MPI parallelization with load balancing and Hybrid MPI/OpenMP parallelization is demonstrated on a network consisting 105K neurons. In the hybrid MPI/OpenMP parallelization approach, a “Dynamic” scheduling scheme is adopted due to the higher efficiency (Figure 27). Figure 28 and Figure 29 show the runtime and speedup of the simulator, and Table 3 shows the runtime of different approaches on the 105k network.

Results show that the hybrid MPI/OpenMP parallelization costs 10% less runtime than the MPI parallelization and proven to be a good method to solve the parallel problem of

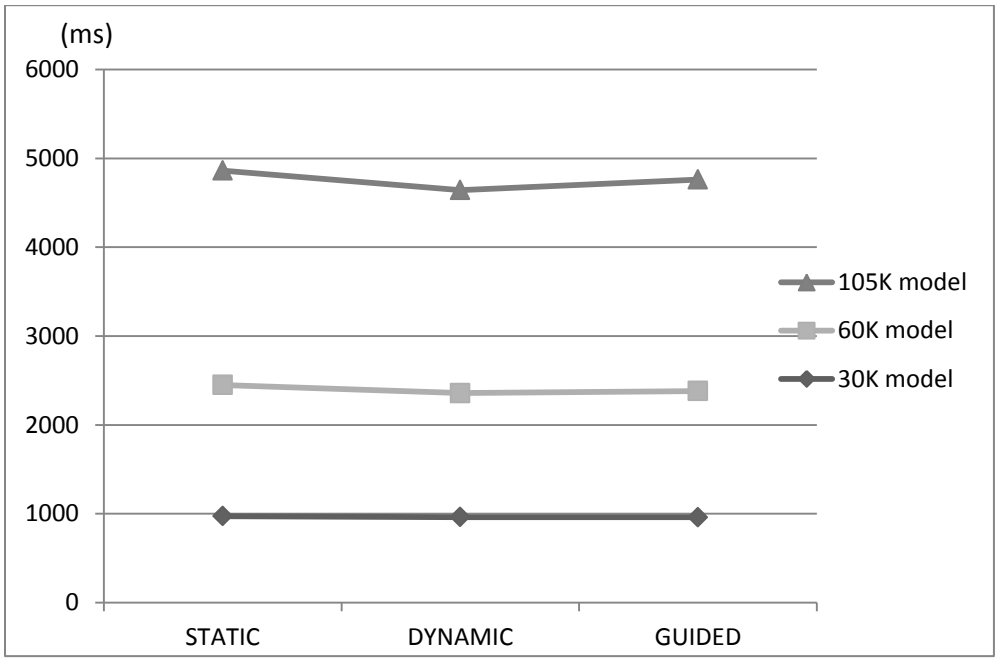

Figure 27 Comparison between different scheduling schemes.

neural network simulation. Moreover, experimental result shows that the runtime of MPI parallelization with load balancing is larger than the MPI implementation. Though the dynamic load balancing scheme makes each processor have balanced workload, the overhead of communication between processors caused by load balancing exceeds the runtime saved by the load balancing scheme. In most of some practical problem, the imbalance ratio is far more than 100% and our imbalance ratio is not so high. So if the workload of the local neural network among processors is extremely uneven, our load balancing scheme may perform better. When using the hybrid MPI/OpenMP, only two MPI processes are generated. Inside each MPI process, multiple OpenMP threads

processes the simulation in parallel and dynamically schedule the workload without any communications between processors. Only when the entire firing messages are gathered, MPI collective routines are used. There is a drawback of the hybrid MPI/OpenMP approach: in case of race condition problem, a "CRITICAL" directive is used which slows the simulation. However, result shows that hybrid MPI/OpenMP approach is still better than the other in the large scale neuron network simulation.

Table 3 Comparison of runtime between different methods on 105k network.

Number of processors	MPI	MPI with load balancing scheme	Hybrid MPI/OpenMP
8	8909 ms	9089 ms	8894 ms
16	4538 ms	4764 ms	4513 ms
32	2394 ms	2624 ms	2288 ms

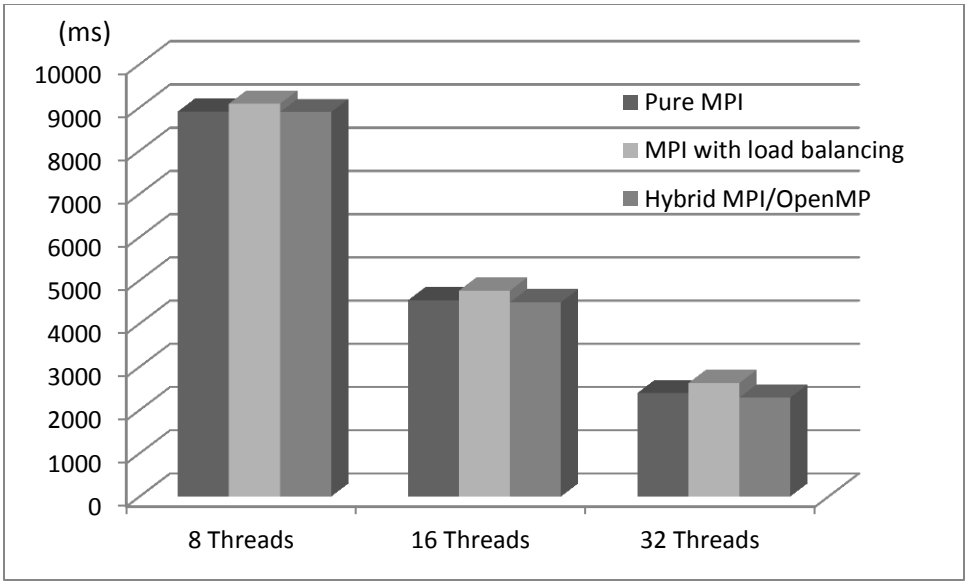

Figure 28 Comparison of simulation time based on different methods.

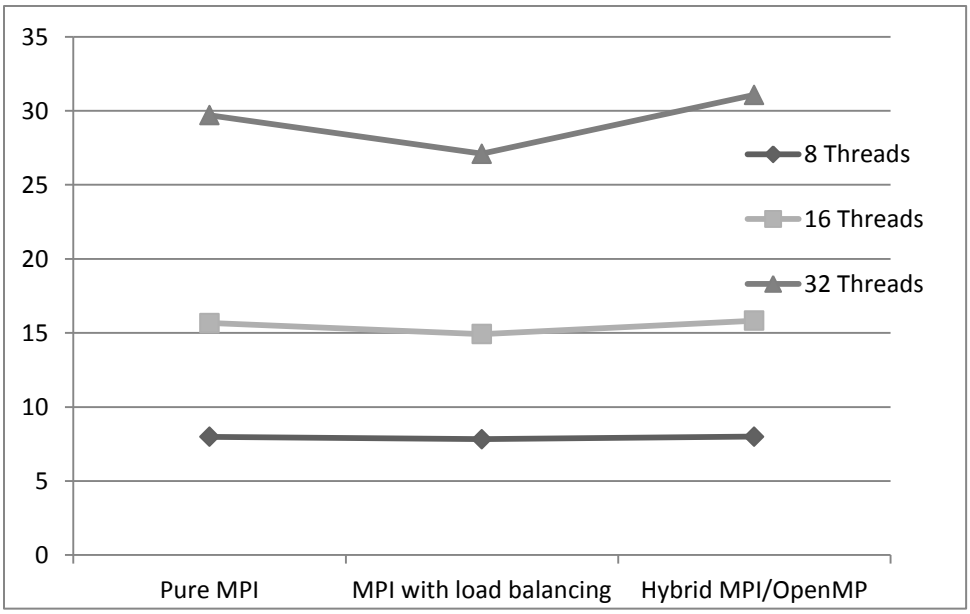

Figure 29 Comparison of speed up based on different methods.

6. SUMMARY AND CONCLUSIONS

In this work, the parallel algorithms for large scale neural networks with biophysically accurate models are studied. The limitations of the existing works have been addressed including the simplicity of the modeled network structures and lack of biophysical details in the neuron models. By taking advantage of emerging multicore-based distributed computing platform, the proposed hybrid MPI/OpenMP parallelization techniques can significantly improve the performance of neural network simulation.

The contributions in this research work can be classified as the following three categories.

Firstly, the biophysically detailed Hodgkin-Huxley type models have been used for all neurons. Models of various ion channels and synaptic receptors are explicitly included to reflect the roles played by these biophysical mechanisms. Using these neuron models as building blocks, a thalamocortical network model is constructed based on data from existing works.

Secondly, to get an accurate and good speedup transient simulator, an explicit forward Euler integration based two-level transient simulation technique has been suggested. The two time steps are 1ms and 0.01 ms respectively. Inside the outer level, one neuron is continuously simulated one hundred small time steps, which increases cache hit rate. Meantime, to speed up the simulation, another technique of merging lineal synaptic receptors is adopted.

In addition, we compare the performance of MPI, MPI with load balancing scheme and hybrid MPI/OpenMP parallelization on different sizes of network. Experiment results show that when the number of processors is small, the MPI implementation achieves good performance without load balancing problems. However, with the increasing size of the neural network and the number of processors, the imbalance ratio is more than 10%. By exploiting dynamic load balancing techniques on a 2D Torus topology, we successfully balance the workload among processors. However, more computation time is consumed as the result of large numbers of message passings between MPI processes. Finally, a hybrid MPI/OpenMP parallelization simulation method is proposed and it perfectly achieves our goal and it shows better speedup than the MPI implementation.

With further development, more interesting techniques may be studied. For dynamic load balancing problems, firing frequencies need to be taken into consideration when the whole neural network is partitioned. Because both the number of neurons and the number of fired neurons in each processor contribute to the workload. Moreover, in this work, we successfully simulate the transition from delta wave in 3 Hz to epilepsy. In the future, by simulating the biologically realistic neural networks, we may provide guidance to clinicians when testing new drugs.

REFERENCES

Abbott, L. F., & Van Vreeswijk, C. (1993). Asynchronous states in a network of pulse-coupled oscillators. *Phys. Rev. E, 48*, 1483-1490.

Barney, B. (2012a). OpenMP. *Lawrence Livermore National Laboratory*. Retrieved from https://computing.llnl.gov/tutorials/openMP/.

Barney, B. (2012b). Parallel computing. *Lawrence Livermore National Laboratory*. Retrieved from https://computing.llnl.gov/tutorials/openMP/.

Bazhenov, M., Timofeev, I. (2006). Thalamocortical oscillations. *Scholarpedia, 1(6)*, 1319.

Binzegger, T., Douglas, R. J., Martin, K. A. C. (2004). A quantitative map of the circuit of cat primary visual cortex. *Journal of Neuroscience, 24,* 8441-8453.

Destexhe, A., Bal, T., McCormick, D.A., & Sejnowski, T.J. (1996a). Ionic mechanisms underlying synchronized oscillations and propagating waves in a model of ferret thalamic slices. *Journal of Neurophysiology, 76(3),* 2049-2070.

D'Haene, M., Schrauwen, B., Campenhout, J.V., & Stroobandt, D. (2009). Accelerating eventdriven simulation of spiking neurons with multiple synaptic time constants. *Neural Computation*, *21(4),* 1068-1099.

Djurfeldt, M., & Lundqvist, M. (2008). Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer. *IBM Journal of Research and Development*, *52(1/2),* 31-42.

Dossi, R., Nufiez, A., & Steriade, M. (1992). Electrophysiology of a slow (0.5-4 Hz) intrinsic oscillation of cat thalamocortical neurons in vivo. *Journal of Physiology*, *447*, 215-234.

Grama, A., Gupta, A., Karypis, G., & Kumar, V. (2003). *Introduction to Parallel Computing (2nd edition).* New York: Addison-Wesley.

Hodgkin, A., & Huxley A. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, *117*, 500-544.

Huguenard, J. R., & Prince, D. A. (1992). A novel t-type current underlies prolonged Ca^{2+}-dependent bursts firing in GABAergic neurons of rat thalamic reticular nucleus. *The Journal of Neuroscience*, *12(10),* 3804-3817.

Izhikevich, E. M., Gally, J. A., & Edelman, G. M. (2004). Spike-timing dynamics of neuronal groups. *Cerebral Cortex, 14(8),* 933-944.

Izhikevich E. M., & Edelman, G. M. (2008). Large-scale model of mammalian thalamocortical systems. *Proceeding of National Academy of Science, 105(9)*, 3593-3598.

Kailasanath, V., & Fu, S. (2003). The HOPES Brain Tutorial. The Huntington's Outreach Program for Education at Stanford. Retrieved from https://www.stanford.edu/group/hopes/cgi-bin/wordpress/2010/06/the-hopes-brain-tutorial-text-version/.

Lan, Z., Taylor, V.E., & Bryan, G. (2002). A novel dynamic load balancing scheme for parallel systems. *Journal of Parallel and Distributed Computing*, *62(12),* 1763-1781.

Leresche N., Lightowler S., Soltesz I., Jassik-Gerschenfeld D., & Crunelti, V. (1991). Low frequency oscillatory activities intrinsic to rat and cat thalamocortical cells. *Journal of Physiology, 441,* 155-174.

Nageswaran, J. M., Dutt, N., Krichmar1, Jeffrey L., Nicolau, A., & Veidenbaum, A. (2009). Efficient simulation of large-scale spiking neural networks using CUDA graphics processors. *Proceedings of the 2009 International Joint Conference on Neural Networks* (pp. 3201-3208). Piscataway, NJ: IEEE Press.

Odabasioglu, E. M. (2004). Which model to use for cortical spiking neurons. *IEEE Transaction on Neural Networks*, *15*, 1063-1070.

Peters, A., & Payne, B. R. (1993). Numerical relationship between geniculocortical afferents and pyramidal cell modules in cat primary visual cortex. *Cerebral Cortex, 3,* 69–78.

Pillage, L.T., Rohrer, R. A., & Visweswariah, C. (1995). *Electronic circuit and system simulation methods.* New York: McGraw-Hill, Inc.

Pospischil,M., Toledo-Rodriguez, M., Monier, C., Piwkowska, Z., & Bal, T. (2008). Minimal hodgkinhuxley type models for different classes of cortical and thalamic neurons. *Biological Cybernetics*, *99*, 427-41.

Rubchinsky, L.L, Kopell, N., & Sigvardt, K.A. (2003). Modeling facilitation and inhibition of competing motor programs in basal ganglia subthalamic nucleus–

pallidal circuits. *Proceedings of the National Academy of Science, 100,* 14427-14432.

Texas A&M Supercomputing Facility. (2012). *Hydra supercomputer*, Retrieved from http://sc.tamu.edu/help/hydra/.

Van Drongelen, W., Lee, H.C., Hereld, M., Chen, Z., Elson, F.P., & Stevens, R.L. (2005). Emergent epileptiform activity in neural networks with weak excitatory synapses. *IEEE Transactions on Neural Systems and Rehabilitation Engineering, 13(2),* 236-241.

Wang, M., Yan, B., Hu, J., & Li, P. (2011). Large-scale simulation of neural networks with biophysically accurate models on graphics processors. *IEEE International Joint Conference on Neural Networks,* (pp. 3184-3193). Piscataway, NJ: IEEE Press.

Zalesky, A., & Fornito, A. (2009). A DTI-derived measure of cortico-cortical connectivity. *IEEE Transactions on Medical Imaging, 28(7),* 1023-1036.

VITA

Name: Jingzhen Hu

Address: WERC Room.102,
TAMU 3128
Texas A&M University
College Station, TX 77843-3128

Email Address: hujingzhen@tamu.edu

Education: B.S., Electrical Engineering, Beijing University of Posts and Telecommunications. Beijing, China, 2009
M.S., Computer Engineering, Texas A&M University, 2012