Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

HHS Public Access

Author manuscript

J Simul. Author manuscript; available in PMC 2018 August 01.

Published in final edited form as:

J Simul. 2017 August ; 11(3): 267–284. doi:10.1057/s41273-016-0033-x.

Load balancing for multi-threaded PDES of stochastic reaction-diffusion in neurons

Zhongwei Lin[1,3], **Carl Tropper**[2,*], **Yiping Yao**[3], **Robert A. Mcdougal**[4], **Mohammand Nazrul Ishlam Patoary**[2], **William W. Lytton**[5], and **Michael L. Hines**[4]

[1]State Key Laboratory of High Performance Computing, National University of Defense Technology, China

[2]School of Computer Science, McGill University, Canada

[3]College of Information System and Management, National University of Defense Technology, China

[4]Department of Neurobiology, Yale University, USA

[5]SUNY Downstate Medical Center, USA

Abstract

Stochastic simulation of chemical reactions and diffusion in a neuron helps to provide a realistic view of the molecular dynamics within a neuron. We developed a multi-threaded PDES simulator, Neuron Time Warp-Multi Thread, suitable for the stochastic simulation of reaction and diffusion in a neuron. In this paper we make use of Q-Learning and Simulated Annealing to determine the parameters for a dynamic load balancing algorithm and for dynamic window control. During the simulation, the runtime statistics of each thread are collected and used to determine the execution time of the simulation. Based upon this assessment, workload is migrated from the most overloaded threads to the most under-load ones. As the results for a calcium wave model show, both approaches can improve the execution time for small simulations by up to 31% (Q-Learning) and 19% (SA). The simulated annealing approach is more suitable for larger populations, decreasing execution time by 41%.

Keywords

Stochastic Neuronal Simulation; Multi-threaded PDES; Load Balancing; Window Control; Q-Learning; Simulated Annealing

1 Introduction

The human brain may be viewed as a sparsely connected network containing approximately 10^{14} neurons. Each neuron receives inputs from thousands of dendrites and sends outputs to thousands of other neurons by means of its axon.

[*]**Correspondence:** School of Computer Science, McGill University, Montreal, Quebec, H3A2K6, Canada. carltropper@gmail.com.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Electrical models for neurons were developed some time ago, using well-known laws of electricity (Ohm, Kirchkoff, capacitance) (Lytton, 2002; Carnevale and Hines, 2006; Sterratt et al., 2011). However, these models provide a limited view of neuronal activity since calcium and other species (nucleotides, peptides, proteins …) diffuse within the cytoplasm of a cell and function as information messengers. In order to develop realistic models of neurons it is necessary to develop models which account for the movement and functioning of these messengers.

The combination of chemical reactions within a cell, flux of ions through the membrane, and diffusion of ions in the cytoplasm can be modeled as a reaction-diffusion system and simulated by (parabolic) partial differential equations (McDougal et al., 2013). However, such a continuous model is not appropriate when there are a small number of molecules, i.e. < 1000. In this case, a stochastic model provides a more realistic and accurate representation (Sterratt et al., 2011; Ross, 2012; Blackwell, 2013). The concentration of calcium in the cytosol is low, between 50–100 nM (Foskett et al., 2007). These low concentrations mean that a small numbers of ions can play an important role in intracellular dynamics.

Calcium plays an important role in regulating a number of cellular processes, including fertilization, gene transcription, muscle contraction and cell death. The calcium used for signaling comes from both extracellular calcium that enters through the cell membrane, and from intracellular sources (Berridge, 1998). An important intracellular source is Ca^{2+}-induced-Ca^{2+}-release (CICR) (Berridge, 1998; Roderick et al., 2003). CICR can occur spontaneously within a cell (Ross, 2012). This kind of localized CICR event is called a ”spark” (Cheng et al., 1993; Tsugorka et al., 1995) or ”puff” (Parker and Ivorra, 1990; Koizumi et al., 1999). It occurs stochastically in both temporally and spatially. In this perspective, a stochastic model is the best choice to represent CICR.

Gillespie’s Stochastic Simulation Algorithm (SSA) (Gillespie, 1977) is a widely-used stochastic algorithm. 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 Next Subvolume Method (NSM) (Elf and Ehrenberg, 2004) is an extension of Gillespie’s algorithm which incorporates the diffusion of molecules into the model. NSM partitions space into cubes called subvolumes. The number of cells involved in a realistic simulation of a network of neurons is immense, hence it is necessary to make use of Parallel Discrete Event Simulation (PDES). In PDES the subvolumes are Logical Processes (LP) (Wang et al., 2011). The diffusion of ions between neighboring subvolumes are events.

NEURON (Carnevale and Hines, 2006, 2013) is a widely used simulator in the neuroscience community. It makes use of deterministic simulators for both reaction-diffusion (McDougal et al., 2013) and electrical models. We are developing PDES simulators to simulate stochastic reaction-diffusion models. We previously developed a process based simulator, Neuron Time Warp (NTW) (Patoary et al., 2014). We verified and examined its performance on a calcium buffer model and a predator-prey (Schinazi, 1997) model. The queuing structure used in NTW and NTW-Multi Thread (NTW-MT) (Lin et al., 2015) is an outgrowth of the multi-level queue in XTW (Xu and Tropper, 2005).

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Communication latency is the main bottleneck of PDES systems (Fujimoto, 1999). In a multi-threaded program each thread has its own execution flow and all threads share the same memory space. Hence it is attractive to develop multi-threaded PDES systems.

The architecture of our multi-threaded simulator, NTW-MT, is depicted in Figure 1. One process is the controller, exercising global control functions (GVT computing and load balancing etc.). The remaining processes are worker processes, used to process events at the LPs residing in each process. Each worker process contains a *communication* thread along with *processing* threads. The communication overhead for this arrangement is high, hence we remove message receiving and sending in the main event-processing loop and employ a *communication* thread for receiving and sending messages for *processing* threads in the same process. All of the worker processes have the same number of threads.

We extend the Multi-Level-Queue (MLQ) and RB-message algorithms in (Xu and Tropper, 2005) in order to reduce contention at the Thread Event Queues (TEQs) and the roll-back overhead. Since the processing threads in NTW-MT are not aware of external messages, we devised a hybrid of Mattern's (Mattern, 1993) and Fujimoto's (Fujimoto and Hybinette, 1997) algorithm to compute the GVT in order to get rid of the overhead for thread synchronization.

PDES consumes a massive amount of memory, involves frequent allocation and deallocation, and generates huge amount of data. Memory usage is managed in order to avoid locking and unlocking when allocating and deallocating memory, and try to maximize cache locality.

(Dematté and Mazza, 2008) points out that a conservative synchronization algorithm using the NSM algorithm will perform poorly because of (1) the zero-lookahead property of an exponential distribution and (2) the dependency graph of the reactions is likely to be highly connected and filled with loops. This means that an optimistic synchronization algorithm is necessary. NTW-MT uses Time Warp. However, the number of roll-backs increases as the number of threads increase (Patoary et al., 2014; Lin et al., 2015) due to the zero-lookahead property, meaning that it is important to control the optimism. Furthermore, our experiments clearly indicated that load balancing is a very important issue.

In this paper, we present our solutions for these problems. The remainder of this paper is organized as follows. Section 2 describes background and related work, section 3 is devoted to static distribution of LPs to threads, section 4 describes our multi-stated Q-Learning (QL) approach for dynamic load balancing, and section 5 presents the use of Simulated Annealing (SA) for dynamic load balancing and window control. Section 6 describes our experimental results and analysis. The conclusion and future work are presented in section 7.

2 Background and Related Work

In NSM the virtual time increment τ is inversely proportional to the sum of reaction rates and diffusion rates. The reaction rate r of a reaction is proportional to the number of reactant molecules, and the diffusion rate s is proportional to the number of molecules which can diffuse (see the calculation of τ, r, s in (Elf and Ehrenberg, 2004)). Hence the virtual time

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

advances more slowly in subvolumes which contain more molecules then in those which have a smaller number of molecules. Let us examine the CICR model as an example of this.

In Figure 2, a dendrite is partitioned into 16 subvolumes which are evenly distributed between two processing threads. At first the number molecules in each subvolume is the same and the virtual time increment is (on the average) equal-hence the workload of the two processing threads is balanced. Inositol 1,4,5-triphosphate (IP_3) molecules are injected into subvolume 0, activating the IP_3 receptor (IP_3R) in the subvolume. The number of Ca^{2+} molecules in the cytosol begins to increase and as a result, the virtual time increment in subvolume 0 becomes smaller than that of the other subvolumes. As the wave spreads along the dendrite, the same thing happens in the neighboring subvolumes (subvolumes 1, 2), and the LPs residing on thread 2 encounter many stragglers.

In a parallel simulation LPs are partitioned into groups and are placed in Processing Elements (PEs) prior to a run. Each PE is responsible for processing the events at the LPs which it hosts. This leads us to static partitioning, the premise of which is that all of the PEs host approximately same number of LPs. Recursive Bisection (RB) and Space-Filling Curve (SFC) (Devine et al., 2005), are examples of static partitioning. Graph partitioning (Feldmann, 2012; Andreev and Räcke, 2004) was introduced to minimize the communication between PEs. Static partitioning is meant for applications in which the workload is static during the simulation run (Devine et al., 2005). However in many simulations the workload of the PEs changes due to the dynamics of the model being simulated. Examples of such changes are (1) dynamic creation and destruction of LPs, e.g. war game in (Yao and Zhang, 2008); (2) motion of LPs, e.g. motion in molecular dynamics simulation (Zheng et al., 2010) (3) events which change the behavior of models, e.g. the injection of IP_3 molecules decreases the virtual time increment in subvolumes in a CICR model (Lin et al., 2015). In the end, dynamic load balancing is a necessity for many simulations.

Load sharing is a straightforward way to implement load balancing. In load sharing the PEs share a pool of tasks. They fetch tasks from the pool, process them and put newly-generated tasks back into the pool. In (Chen et al., 2011) and (Miller, 2010), all of the PEs for an individual process share a global pending event set known as the task pool. In (Vitali et al., 2012) the authors first compute an estimate of the computation workload for each kernel instance(PE) and compute the CPU time for each PE. An overloaded PE gets more CPU time than under-loaded PEs. Load sharing is easy to implement and can achieve good results. Clearly the task pool can suffer contention (Chen et al., 2011).

A dynamic load balancing algorithm has three components (Alakeel, 2010) (1) an information strategy-the collection of statistics about each PE in the system and a decision as to whether the system is unbalanced (2) a location strategy-a decision as to which LPs are to be transferred and when to transfer them (3) a transfer strategy-a determination of the source and destination of the migrated workload.

There are two categories of dynamic load balancing algorithms-centralized and decentralized. In a centralized algorithm, a specific node (the controller) makes the three

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

decisions and other nodes transfer LPs. (Meraji et al., 2010) and (Meraji and Tropper, 2012) employ centralized algorithms. They employ an AI algorithm and a simulated annealing algorithm to determine the number of LPs which are to be transferred between PEs. (Zheng et al., 2010) uses a hierarchical centralized load balancing scheme to balance the workload for Charm++ applications. (Pilla et al., 2014) also employs a centralized scheme. It models the Non-Uniform Memory Access (NU-MA) feature of multi-core clusters in order to estimate the cost of migrating workload.

In decentralized algorithms there is no centralized controller-each PE collects information and transfers workload independently. For example, (Cosenza et al., 2011) employs a decentralized algorithm in which each PE changes workload by adjusting the boundary regions between neighbors independently.

In general, centralized algorithms are easy to implement and as reported in (Zheng et al., 2010) can scale up to a few thousand processors. Their drawbacks include (1) they may fail due to a controller crash (2) the controller can be a system bottleneck (Cosenza et al., 2011). (Zheng et al., 2010) argues that decentralized approaches can yield poor performance due to incomplete information exchanged by neighboring processors.

3 Initial Distribution of LPs to Threads

Our initial distribution of LPs to threads places the same number of LPs on each thread. In NTW (Patoary et al., 2014) and NTW-MT (Lin et al., 2015), the LPs are partitioned using their global identifier. Suppose there are N LPs, 0, 1, ⋯, $N-1$, and p threads, 0, 1, ⋯, $p-1$. then a $block = \lfloor N/p \rfloor$ of LPs are placed in each thread -LPs with IDs from $block \times i$ to $block \times (i+1) - 1$ are placed in the ith thread, and the remaining LPs are placed in the last thread (we called this the BLOCK strategy).

METIS (Schloegel et al., 2000) (the GRAPH strategy) was compared to BLOCK. In METIS each subvolume is a node in a graph, and there is a bidirectional edge between each pair of neighboring nodes. Since a molecule has the same probability for diffusing to each one of its neighboring subvolumes, the weight of all of edges is set to the same value. We found that both strategies had similar performances. The explanation for this is as follows. Note that the ID of an individual LP is assigned by $gid = x + y \times X + z \times (X \times Y)$, where X, Y and Z are the number of grids in each dimension, (x, y, z) is the coordinate of an individual grid. Adjacent grids have contiguous IDs and probably reside in the same block. As a result neighboring LPs are placed in the same thread. We made use of the BLOCK strategy in our research.

4 Q-Learning Approach

We employ a centralized algorithm for dynamic load balancing. We first estimate the workloads for each thread and detect imbalances using runtime statistics (section 4.1). This is the *information strategy* of section 2. We then use multi-state Q-Learning in (Watkins and Dayan, 1992; Meraji et al., 2010) to evaluate parameters which are critical to the performance of the simulation (section 4.2). These parameters are (1) the type of workload (communication or computation) (2) the number of threads involved in workload migration

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

and (3) the load to be exchanged (Meraji et al., 2010). LPs are migrated from the most overloaded threads to the most under-loaded threads (section 4.3)-this corresponds to our *location strategy*. The selection of which LPs (*transfer strategy*) depends on the type of workload to be balanced, computation or communication.

4.1 Workload Estimation and Imbalance Detection

Assume that there are $p = m \times n$ *processing* threads (n worker processes, m *processing* threads in each worker process), numbered 0, 1, ⋯, $p - 1$, The workload for each processing thread is checked every T_L seconds (*one round*). Rounds are numbered in increasing order, i = 1, 2, 3, ⋯. In the *ith* round, processing thread j processes a_j^i ordinary events. The receive time of the smallest event in its TEQ is c_j^i. $b_j^i = c_j^i - GVT^i$ indicates how far thread j is running ahead of the *ith* Global Virtual Time (GVT). Let $\mu_j^i = a_j^i / \max\ a_j^i$ and $\nu_j^i = b_j^i / max\ b_j^i$, $0 \quad j < p$, then the workload of the *jth processing* thread is calculated by equation (1), and the computation balancing coefficient of the *ith* round is calculated by equation (2).

$$cpl_j^i = \zeta(\mu_j^i - \overline{\mu}^i) - \eta(\nu_j^i - \overline{\nu}^i) \quad (1)$$

$$CPL^i = \sqrt{\frac{1}{p}\sum_{j=0}^{p-1}\left(cpl_j^i - \overline{cpl}^i\right)^2}, \overline{cpl}^i = \frac{1}{p}\sum_{j=0}^{p-1} cpl_j^i \quad (2)$$

where $\overline{\mu}^i = \frac{1}{p}\sum_{j=0}^{p-1}\mu_j^i, \overline{\nu}^i = \frac{1}{p}\sum_{j=0}^{p-1}\nu_j^i$, $\zeta, \eta \in [0, 1], \zeta + \eta = 1$. The computation workload is unbalanced if $CPL^i > T_{cmp}$, where T_{cmp} is threshold. In order to compute the communication workload we proceed as follows. Each LP has an array *LPComm* of length $m \times (n - 1)$, in which the *ith* element records the number of external events sent from this LP to the corresponding thread since the last round. Each processing thread also holds an array *ThreadComm* of length $m \times (n - 1)$ in which the *ith* element is set to the sum of $LPComm_i$ of LPs hosted by that thread. Analogous to the definition of the computation workload, the communication workload coefficient CML^i is defined by replacing μ_j^i in equation (1) by $\sum_{k=1}^{m\times(n-1)} ThreadComm_k$, except for the second item.

4.2 Determination of Workload Migration

We utilize the multi-state Q-Learning algorithm from (Meraji et al., 2010). In Q-learning, an agent stores information (Q) about the environment- the expected reward for each action in each state of the environment. It selects an optimal action a_m in the present state s_t and executes that action. In this way, the agent can find the optimal policy even if there is no prior knowledge about the effects of the actions on the environment (Watkins and Dayan, 1992). The update rule (Watkins and Dayan, 1992) is:

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

$$Q(s_t, a_t) \leftarrow \beta Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{FA} Q(s_{t+1}, a)] \quad (3)$$

where $\alpha, \beta \in (0, 1)$, $\alpha + \beta = 1$. α and γ are the learning step and the discount rate of the Q-learning algorithm. FA contains feasible actions for state s_{t+1}. r_{t+1} is the reward for the action executed in state s_t.

The agent (a simulation run) can have four states: (1) Balanced computation and Balanced communication (BPBM), if $CPL^i \leq T_{cmp}$ and $CML^i \leq T_{cmm}$; (2) Unbalanced computation and Balanced communication (UPBM), if $CPL^i > T_{cmp}$ and $CML^i \leq T_{cmm}$; (3) Balanced computation and Unbalanced communication (BPUM), if $CPL^i \leq T_{cmp}$ and $CML^i > T_{cmm}$; (4) Unbalanced computation and Unbalanced communication (UPUM), if $CPL^i > T_{cmp}$ and $CML^i > T_{cmm}$.

Three parameters are used to define actions: (1) A, the type of workload-computation or communication (2) P, the percentage of threads involved in a migration (3) L, the number of LPs transferred from one thread to another. A can have two values. If P has M values and L has N values, there are $2 \times M \times N$ actions possible in each state.

The reward function plays an essential role to a Q-learning algorithm. We define the GVT Advance Rate, GVTAR, to be the increment in GVT per second. Since the workload is checked every T_L seconds, GVTAR is defined by equation (4).

$$GVTAR^i = \frac{GVT^i}{i} \quad (4)$$

where i is the sequence number of the round and GVT^i is the GVT for the ith round. If the agent is in state s_i and executes action a_x in the ith round, then the effect of a_x could be assessed in the successive round. However we can obtain a longer term assessment of the effects of the action by calculating it later, in round $i + k$, $k \in N^*$, where $i + k$ is set as the sequence number of next round in which the agent executes another action. The reward for the action a_x is computed as $r_a^i = GVTAR^{i+k} - GVTAR^i$. We count the number of times that an action a has been executed in state s, C_a, and set the reward of action a in state s to

$$R_a = R_a + (r_a^i - R_a)/(C_a + 1) \quad (5)$$

For the first D rounds that the load check is executed, no migration is allowed, even if the load is unbalanced, thereby making it possible to compute a reference value for GVTAR. If executing action a_x in state s_t leads to a higher GVTAR, the reward for this action increases in equation (5) and the action has a higher probability of being selected in the future, i.e. good actions are reinforced.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Migrations in successive load check rounds i and $i + 1$, are not allowed-the workload estimate for the second round may be inaccurate because it takes time for workload on the threads involved in the migration to arrive at steady state.

The controller process sends a *LOAD_CHECK* message to each worker process every T_L seconds and awaits a reply. Upon receiving a *LOAD_CHECK* message, the communication thread in each worker process collects the appropriate statistics and sends them to the controller process. The controller process executes algorithm 1 after all of the replies have been received.

Algorithm 1

QL load balancing

1: Compute computation and communication workload via equation (1), store the values in *cplVector* and *cm-l Vector* respectively, sort the elements in each vector in decreasing order.

2: Compute CPL^i and CML^i via equation (2).

3: Check the new state of the agent s_i by using T_{cmp} and T_{cmm}.

4: **if** s_i == BPBM **then**

5: return. {no need to balance workload}

6: **else**

7: Set the state of the simulation to s_i.

8: Compute $GVTAR^i$ via equation (4), set $r = GVTAR^i - GVTAR^{i-k}$.

9: Run the Q-learning algorithm with r as the input.

10: **end if**

We use a straightforward strategy for selecting the threads involved in a computation migration-we migrate workload from the most overloaded threads to the most underloaded ones. For a communication workload, LP-s are migrated from the thread x which has the largest communication workload to the thread y which receives the most messages from x. The controller sends a *LOAD_MIGRATION* message containing the A and L values to each process which has the threads which must transfer workload.

Algorithm 2

Q-Leanring

Input: reward of the latest execution of the last action

r

Output: A, P, L

1: Update the reward of the last action via equation (5).

2: Update the Q matrix by equation (3).

3: Select the action with maximum reward in state s_i with probability $1 - \epsilon$, else select an action from the remaining feasible actions randomly.

4: Compute A, P and L from the selected action.

4.3 Migrating Workload

Workload is transferred between worker processes directly. When a communication thread receives a *LOAD_MIGRATION* message, it forwards this message to the source thread by inserting this message into the corresponding TEQ. Each LP has at least one representative event in the TEQ, then we say that LP x is more urgent then LP y if $e_x.rt < e_y.rt$ $(x \quad y)$, where $e_x.rt$ is the receive time of the representative event of LP x. When a processing thread processes a *LOAD_MIGRATION* event, it selects the LPs and sends them to the destination thread. The selection of LPs depends on the type of the workload. If it is *computation*, then L LPs which are urgent are selected, otherwise it selects L LPs which have the most communication with the target thread.

4.4 Window Control

Based on our observations, window control is a useful mechanism to control the optimism in Time Warp. It works as follows- at a given wall clock time, suppose the GVT is G and the window value is W, then only events with timestamp $t \in (G, G + W]$ can be processed. The value chosen for W is pivotal role- a small value can lead to many PEs being idle, while a high value can fail to prevent excessive optimism.

5 Simulated Annealing

As pointed out in (Meraji and Tropper, 2012), the major drawback of Q-Learning is that actions must be specified prior to the simulation. Worse yet, the time to pick the best action in a given state is proportional to the number of actions-more actions take more time to evaluate.

Simulated Annealing (SA) (Kirkpatrick et al., 1983) is a search technique used to produce an approximate global optimum for a function defined on the search space. A defining characteristic of the algorithm is that it can probabilistically pick a less then optimal solution at each iteration of the algorithm in order to preserve the possibility of choosing a global optimum. It takes its inspiration from an algorithm used to control the thermodynamic free energy of a metal when cooling it. The parameter T in SA plays the role of the temperature in the real annealing process. In the real annealing process the probability that molecules are in state s satisfies the Boltzman distribution (Kirkpatrick et al., 1983) as follows.

$$P\{\bar{E} = E(s)\} = \frac{1}{Z(T)} \exp - \frac{E(s)}{bT} \quad (6)$$

where b is Boltzmann constant, $E(s)$ calculates the energy of the system in state s, $Z(T) = \sum_{s \in S} \exp - \frac{E(s)}{bT}$, S refers to all states at which molecules are at temperature T. A

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

new state s' is accepted if $E(s')$ $E(s)$ or with probability $\exp - \frac{\Delta E}{bT}$, where $E = E(s') - E(s) > 0$. Once when the state s' is accepted, the state of the system is updated by s'. An outline for the SA algorithm is given by:

1. Initialization-set the the initial temperature T and the end conditions for the algorithm. Set the initial state for the model to s_0. Define two functions, the objective function $f: S \rightarrow \mathbb{R}$ which evaluates each state and the the state transition function $g: S \rightarrow S$, which picks a neighboring state s_{i+1} given the present state s_i. Two types of end conditions are used- the algorithm ends if $T < \varepsilon$ or if the number of iteration $n > N_{max}$, where ε and N_{max} are given parameters.

2. Iteration- generate a new state $s_{i+1} = g(s_i)$ and evaluate the state $f_{i+1} = f(s_{i+1})$. Accept s_{i+1} if $f_{i+1} < f_i$, else generate a random number $r \sim U(0, 1)$ and accept s_{i+1} if $\exp(-\ f/T_i) > r$, where $f = f_{i+1} - f_i$. Terminate the algorithm if the end condition is satisfied.

3. Cooling down- decrease the temperature gradually and go to iteration phase. Our cooling down scheme is defined by $T_{i+1} = \alpha_i T_i$, where $\alpha_i \in (0, 1)$.

The SA algorithm does not depend on the initial state and can find an approximate global optimum within a finite number of steps (Kirkpatrick et al., 1983).

Both computation and communication workload should be accounted for in defining our workload. Hence our Load Coefficient (LC) in the *ith* round is defined by equation (7).

$$LC^i = \lambda CPL^i + (1 - \lambda) CML^i \quad (7)$$

where $\lambda \in [0, 1]$ is used to balance the weights of computation and communication workload.

Let us first note that our simulation is balanced before the injection of IP_3 molecules, hence initiating a migration is unnecessary. LC^i in equation (7) is small before the injection of IP_3 molecules, while it can increase to a larger value if the simulation becomes unbalanced. To distinguish between these two circumstances, a parameter B is used. The simulation is balanced if LC^i B, otherwise it is unbalanced.

We use the same two parameters, P and L which we used in the Q-Learning approach to control the total number of LPs to be moved. A parameter W is used for the window size. The values for each of these parameters is chosen from an interval of permissible values. So we have a quadruple $< B, P, L, W >$, where each parameter has a valid interval (X_{low}, X_{up}), where $X \in \{B, P, L, W\}$. Each parameter is initialized randomly, choosing values from these intervals.

As to the state transition function, we suppose that each parameter $X, X \in \{B, P, L, W\}$, can increase or decrease at the same probability in each iteration step. To prevent the state jumping from one end to the other end in the parameter space, we suppose an individual parameter picks a value within an interval ρ_X in each iteration step. To adjust the distance

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

between neighboring states, in each iteration step ρ_X is multiplied by a random number in uniform distribution. In summary, the next state of a parameter is given by:

$$X_{i+1}=\begin{cases} X_i+\varphi \times \rho_X & \psi<0.5 \\ X_i-\varphi \times \rho_X & \psi \geq 0.5 \end{cases} \quad (8)$$

where φ, $\psi \sim U(0, 1)$, $X \in \{B, P, L, W\}$. Each parameter, i.e. B, P, L and W, has its own interval, say ρ_B, ρ_P, ρ_L and ρ_W, then the next state would locate within a four-dimensional cube determined by the 16 points ($b \pm \varphi_1 \times \rho_B$, $p \pm \varphi_2 \times \rho_P$, $l \pm \varphi_3 \times \rho_L$, $w \pm \varphi_4 \times \rho_W$), where ($b, p, l, w$) is the present state and also the center point of this cube. Here we can see the use of ρ_X, φ and ψ is to help SA to explore the whole parameter space better.

We make use of GVTAR as our objective function because it is a measure of how fast a simulation advances reactive to wall clock time-the bigger the better.

The SA algorithm is described in algorithm 3. The algorithm terminates when the temperature gets close to zero or it uses 70%–80% of the iterations (Meraji and Tropper, 2012).

Algorithm 3

The SA algorithm

1: $i = 0$, $i_{\upsilon} = 0$
2: **while** *temperture* $> \varepsilon$ **and** $i < N_{max}$ **do**
3: $i++$
4: Wait for all the replies from worker processes
5: Calculate LC^i via equation (1), (2) and (7)
6: **if** $i < i_{\upsilon}$ **then**
7: Continue to the next load-check round
8: **end if**
9: **if** $LC^i > B^i$ **and** i i_{υ} **then**
10: Balance workload among threads with P^i and L^i
11: $i_{\upsilon} = i + I$
12: Continue to the next load-check round
13: **end if**
14: Calculate the new GVTAR f_i
15: Evaluate the present state by algorithm 4
16: Generate a neighboring state using equation (8)
17: Cool down, *temperature*$* = c$, $c = 0.8499$
18: Broadcast the new window to all worker processes
19: **end while**

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Algorithm 4

State evaluation

Input: New GVTAR f_i

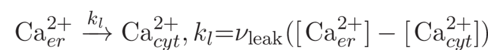
1: **if** $f_i > f_{i-1}$ **then**
2: Accept the neighboring state
3: **else**
4: Generate a random number $rand \sim U(0, 1)$
5: **if** $\exp\{(f_i - f_{i-1})/temperature\} > rand$ **then**
6: Accept the neighboring state
7: **end if**
8: **end if**

6 Experimental Study

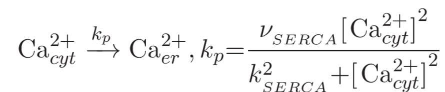
6.1 Model

A deterministic CICR model has been developed in NEURON (Neymotin et al., 2015). Based on this model we developed a discrete event model and simulated it. In our experiments we only take the IP_3 and Ca^{2+} molecules into account. We simplified the CICR receptor model by assuming that (1) the IP_3 receptor opens when the concentration of IP_3 and Ca^{2+} are both higher than some respective threshold (2) after opening an IP_3 receptor channel will close in a period of time determined by an exponential distribution. The reactions include:

$$\mathrm{Ca}_{er}^{2+} \xrightarrow{k_r} \mathrm{Ca}_{cyt}^{2+}, k_r = \nu_{IP3R} m^3 n^3([\mathrm{Ca}_{er}^{2+}] - [\mathrm{Ca}_{cyt}^{2+}])$$



$$\mathrm{Ca}_{er}^{2+} \xrightarrow{k_l} \mathrm{Ca}_{cyt}^{2+}, k_l = \nu_{\mathrm{leak}}([\mathrm{Ca}_{er}^{2+}] - [\mathrm{Ca}_{cyt}^{2+}])$$



$$\mathrm{Ca}_{cyt}^{2+} \xrightarrow{k_p} \mathrm{Ca}_{er}^{2+}, k_p = \frac{\nu_{SERCA}[\mathrm{Ca}_{cyt}^{2+}]^2}{k_{SERCA}^2 + [\mathrm{Ca}_{cyt}^{2+}]^2}$$

where $\mathrm{Ca}_{\mathrm{er}}^{2+}$ refers to Ca^{2+} in the Endoplasmic Reticulum (ER), $\mathrm{Ca}_{\mathrm{cyt}}^{2+}$ refers to Ca^{2+} in the cytosol, [•] refers to the concentration of the species •, $m = [IP_3] / ([IP_3]+k_{IP_3})$, $n = [\mathrm{Ca}_{\mathrm{cyt}}^{2+}]/([\mathrm{Ca}_{\mathrm{cyt}}^{2+}] + k_{\mathrm{act}})$, k_{IP_3}, k_{act}, ν_{IP3R}, ν_{leak}, ν_{SERCA} and k_{SERCA} are given constant parameters, the values of which can be found in (Neymotin et al., 2015). $\mathrm{Ca}_{\mathrm{er}}^{2+}$ can only diffuse within the ER, while $\mathrm{Ca}_{\mathrm{cyt}}^{2+}$ and IP_3 can only diffuse within the cytosol.

The initial concentrations of Ca^{2+} (in the ER and cytosol) and IP_3 are set to 9.511765 μM, 0.1 μM and 0.1 μM respectively. In each subvolume, 17% of the volume corresponds to the

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

ER while the remaining 83% corresponds to the cytosol. The threshold for controlling the channel opening is 0.2 μM and 2 μM for cytosolic Ca^{2+} and IP_3, respectively.

We executed this simplified model on a one-dimensional geometry for illustrative purposes. The figures are in the online appendix. As these appended figures show, this simplified model produces a calcium wave.

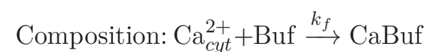
6.2 Platform and Geometry

We use two platforms. One machine (PEPI) is a cluster with 4 Intel(R) Xeon(R) E7 4860 2.27 GHz, 10 cores per processor, 1 TB memory, with Linux 2.6.32-358.2.1.el6.x86_64, Red Hat Enterprise Linux Server release 6.4 (Santiago). The other is the SW2 node of Guillimin, 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 2.6.32-279.22.1.el6.x86 64 GNU/Linux.

We simulate the above CICR model for a CA1 hippocampal pyramidal neuron (Ishizuka et al., 1995). The hippocampal pyramidal neuron used in our experiment is from NeuroMorpho.Org (Ascoli et al., 2007) (NO. c91662). A three-dimensional view of this neuron is given in the online appendix. The neuron is partitioned into mesh grids, where each grid is a subvolume. We selected 14749 subvolumes with a distance of less than 50 μm from the soma (a three-dimension view of the selected region is also displayed in the online appendix). The length of each subvolume is 0.5 μm.

6.3 Verification

Free calcium is buffered by intracellular buffers (calmodulin or parvalbumin) and is therefore unavailable. However, it can escape from these buffers, resulting in an almost constant concentration of cytosolic calcium. This observation can be used to verify our simulator. The buffer model includes two reactions as follows.



$$\text{Composition: } \mathrm{Ca}^{2+}_{cyt} + \mathrm{Buf} \xrightarrow{k_f} \mathrm{CaBuf}$$

$$\text{Decomposition: } \mathrm{CaBuf} \xrightarrow{k_b} \mathrm{Ca}^{2+}_{cyt} + \mathrm{Buf}$$

We executed this buffer model using the deterministic NEURON simulator on a ”Y” shaped geometry which consists of three cylinders (10 μm long, 1 μm diameter). A sketch of this geometry can be found in (Patoary et al., 2014) and is depicted in online appendix.

To show spatial effects, we initialize a high concentration of free calcium in one cylinder, a low concentration in the remaining two cylinders, and trace the evolution of the molecules. The initial high and low concentrations of Ca^{2+} are set to 1.0 mM and 0.1 mM respectively, while the concentrations of Buf and CaBuf are set to 0.5 mM and 0.001 mM everywhere. The reaction rates k_f and k_b are set to 0.06 and 0.01. In NTW-MT, the geometry is partitioned into 2766 subvolumes, and the length of each subvolume is 0.25 μm. The high

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

concentration of Ca^{2+} is initialized in the first 922 subvolumes (922/2766=1/3) by placing 10 molecules in each of those subvolumes, while 1 calcium molecule is placed in each of the remaining 1844 subvolumes. The initial numbers of Buf and CaBuf molecules in each subvolume are 5 and 0, respectively. From Figure 3, we can see that the stochastic model exhibits a similar behavior to that of the deterministic model.

6.4 Performance Results and Analysis

6.4.1 Q-Learning—The Q matrix is set to zero, indicating the agent has no knowledge about the environment. The reward for a feasible action in each state is initialized to 0, while the rewards for infeasible actions are $-\infty$. ε, γ, α are initialized to 0.1, 0.9 and 0.1, respectively.

P has two values (0.1 and 0.2), and L has three values (0.003, 0.004 and 0.005). The number of LPs to be migrated is $L \times NLP_{avg}$, where $NLP_{avg} = N/nThread$, N is the number of LPs and $nThread$ is the number of processing threads.

We used PEPI for this experiment. All of the processing threads were placed in the same process, hence no messages were used to send LPs between processing threads (note that the migrated LPs are reassigned to the *target* thread and still accessible there). The results are shown in Figure 4 and 5.

From Figure 4 and 5, the execution time and the number of roll-backs both decrease when load balancing is used. The biggest improvement (about 31% in execution time) takes place when 8 threads are used and ζ is set to 0.5.

Comparing the weight of the two items in equation (1), we see that when ζ is set to 1 and 0.8 we improve the performance by up to 25%. The thread which runs furthest ahead receives more stragglers and has more roll-backs then the other threads. Setting both terms to equal weights brings the best improvement. The results of giving more weight to the second item are inconsistent. When fewer threads are used the threads did not get seriously unbalanced. When more threads (8, 16) are used equation (1 may give an overly optimistic estimate of workload because the virtual time increment varies from 0 to a much higher value, thereby leading to unnecessary migration.

A collection of cluster nodes are used for larger models in which LPs are distributed among the processes and migrate between them. We start several processes on PEPI and Guillimin and transfer LPs between the processes using shared memory or MPI. The results are shown in Figure 6 and 7.

From Figure 6, we see that the execution time decreases by up to 21% when 8 threads are used. The results are not as good as those achieved by intra-process migration (differs about 10%) because of the extra overhead for sending LPs. Placing more threads in the same process results in slightly better performance.

Excessive LP migration can overwhelm the gains obtained by load balancing. In the ”pure remote” mode of Figure 7, in which each worker process has one processing thread and all worker processes occupy separate nodes, all inter-process messages are transferred between

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

the nodes. When two processing threads are used, the execution time is about 8% more than that of a simulation which does not employ load balancing. In the remaining cases (the number of migrated LPs is not excessive), the best improvement is about 16% when 8 threads are used and ppn is 3.

6.4.2 Simulated Annealing—The parameters ζ and η in equation (1) are both set to 0.5 as is λ in equation (7). The intervals for the parameters in SA are as follows: $B \in (0, 4)$, $P \in (0, 0.5)$, $L \in (0, 0.1)$, $W \in (0, 10)$. The number of LPs to be migrated is $L \times NLP_{avg}$.

In the following experiment, run on PEPI, all of the processing threads are placed in the same worker process. The results are depicted in Figure 8 and 9.

From Figure 8 and 9, we see that both Q-Learning and SA can decrease the execution time as the number of processing threads increases. SA did not perform as well as Q-Learning-the best improvement for Q-Learning is about 30%, while only 19% improvement results from SA. The reason for this is that the size of this simulation is small-the simulation ends before SA finds the optimal value. At some points (small B and large P, L), SA may lead to unnecessary migration or a too large migration, resulting in an increase in execution time. This is quite clear when 2 threads are used because too many LPs are transferred in an individual migration. SA results in fewer rollbacks then Q-learning. For example, 37% of the roll-backs are reduced by SA when four threads are used. Since SA employed window control and QL did not, one direct inference is that window control contributes to decreasing the overall roll-back.

We ran SA in the same scenario as Q-Learning in order to see the effects of interprocess communication. We depict the results in Figure 10.

In Figure 10 we see similar behavior; SA takes longer than an execution which does not employ load balancing when a small number of threads are used. This is due to the overhead for LP migration. The execution time decreases when more threads are used.

To determine the effect of SA on large scale runs, we increase the total number of LPs in the simulation and the *END* condition. This is done by selecting subvolumes in a larger region around the soma of the CA1 neuron. There are 23547 subvolumes within a radius 120 μm from the soma of the CA1 neuron. We run the same scenario on the Guillimin. Results are contained in Figure 11.

From Figure 11, we see that for a larger model SA performs well-it can achieve a 41% improvement in execution time when 8 threads are used within a process, and even decreases execution time by up to 34% in the pure remote mode. Big migrations (i.e. 2 threads are used) always leads to the least improvement due to the overhead of transferring LPs. This implies that the number of LPs to be exchanged should be selected properly. We note that in all cases, the best improvement happens when 8 threads are used. The explanation for this is as follows. The performance of PDES is highly dependent on the number of LPs which can be run in parallel. In a three-dimensional grid, one subvolume can have no more than 6 adjacent subvolumes, which leads to at most 7 threads inserting events at an individual LP simultaneously. Let $\theta = \textit{number of processing threads}/7$, if $\theta < 1$, there is not enough threads

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

to process events even though some events can be processed (note that only the host thread can process the events at the LPs hosted by that thread). The total rollback is not large in this case. On the other hand, if $\theta \gg 1$, neighboring LPs are separated and distributed to threads, and get opportunity to be processed (note that there are enough threads now), which can lead to concatenate rollback among those neighboring LPs. As a result, it shows a rollback expansion when many threads are used. In both cases, the highest parallelism cannot be achieved. This leaves us with 8 threads as the best choice for the number of threads to use. Finally, we note that better performance is achieved by placing more threads within the same process/node.

7 Conclusion

Stochastic simulation of reaction and diffusion in neurons can provide a realistic view of the molecular dynamics within and between neurons. We have developed a multi-threaded PDES simulator, NTW-MT (Lin et al., 2015) which accomplishes this. It makes use of a Multi-Level Queue (MLQ) and a RB-message-involved roll-back (Xu and Tropper, 2005) mechanism to disperse contention and decrease the overhead of roll-backs.

Load balancing is an important issue for parallel and distributed systems. In this paper, we present load balancing algorithms and a dynamic window control algorithm for NTW-MT which make use of techniques from artificial intelligence.

Prior to the start of the simulation we use a static partitioning. During run time we make use of dynamic load balancing. In static partitioning, LPs are given IDs according to their geometric location and are placed in processing threads so that neighboring LPs are in the same thread.

We make use of two load balancing algorithms. One is based on reinforcement learning while the second makes use of simulated annealing. The reinforcement learning algorithm employs multi-state Q-Learning (Meraji et al., 2010) to evaluate the parameters involved in the algorithm. These are the type of workload, the number of threads involved in LP migration and the load to be transferred. The transmission of LPs depends on the location of the source and destination threads-if they reside in the same process, the LPs are reassigned to the target thread, otherwise the LPs are transferred to the target thread. On a calcium wave model with 14749 LPs on a CA1 neuron, Q-Learning achieved a 31% improvement in execution time by migrating workload between threads within the same process, and a 16% when improvement when remote communication was used.

A drawback of Q-Learning is the need to specify its actions prior to actual simulation. Simulated Annealing (Kirkpatrick et al., 1983) does not depend on the initial state and can find quasi-optimal values within a finite number of steps. We employ SA to learn four parameters-B, the threshold beyond which the simulation is unbalanced; P, the number of thread pairs involved in a migration; L, the number of LPs to be transferred, and W, the window size. It is well known that employing a window in an optimistic simulation serves to decrease the number of rollbacks. SA performed well-it achieved a 41% improvement in execution time when 8 threads were used within a process, and decrease execution time by

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

up to 34% even in the pure remote mode. We attribute a decrease in the number of rollbacks (up to 37%) to the use of window control.

SA tends to transfer too many LPs when few (e.g. 2, 4) threads are used, thus one future effort is to find a better determination of this parameter. Deterministic simulations run much faster than exact algorithms (e.g. Gillespie's SSA, NSM), and are applicable in regions where the number of molecules is large, thus a hybrid (deterministic-stochastic) algorithm is another future effort.

Acknowledgments

This work is supported by China Scholarship Council and in part by the National Natural Science Foundation of China (No. 61170048), Research Project of State Key Laboratory of High Performance Computing of National University of Defense Technology of China (No. 201303-05) and the Research Fund for the Doctoral Program of High Education of China (No. 20124307110017). This work is also funded by U.S. National Institutes of Health grants R01MH086638 and T15LM007056.

References

Alakeel AM. A guide to dynamic load balancing in distributed computer systems. International Journal of Computer Science and Network Security (IJCSNS). 2010; 10(6):153–160.

Andreev, K., Räcke, H. Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '04. New York, NY, USA: ACM; 2004. Balanced graph partitioning; p. 120-124.

Ascoli GA, Donohue DE, Halavi M. Neuromorpho.org: A central resource for neuronal morphologies. The Journal of Neuroscience. 2007; 27(35):9247–9251. [PubMed: 17728438]

Berridge MJ. Neuronal calcium signaling. Neuron. 1998; 21(1):13–26. [PubMed: 9697848]

Blackwell K. Approaches and tools for modeling signaling pathways and calcium dynamics in neurons. Journal of neuroscience methods. 2013; 220(2):131–140. [PubMed: 23743449]

Carnevale, NT., Hines, ML. The NEURON book. New York, NY, USA: Cambridge University Press; 2006.

Carnevale NT, Hines ML. Neuron, for empirically-based simulations of neurons and networks of neurons. 2009–2013 [Last access on May 1st 2015] http://www.neuron.yale.edu.

Chen, L-l, Lu, Y-s, Yao, Y-p, Peng, S-l, Wu, L-d. Proceedings of the 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation. Nice, France: IEEE Computer Society; 2011. A well-balanced time warp system on multi-core environments; p. 1-9.

Cheng H, Lederer W, Cannell MB. Calcium sparks: elementary events underlying excitation-contraction coupling in heart muscle. Science. 1993; 262(5134):740–744. [PubMed: 8235594]

Cosenza, B., Cordasco, G., De Chiara, R., Scarano, V. Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on. IEEE; 2011. Distributed load balancing for parallel agent-based simulations; p. 62-69.

Dematté, L., Mazza, T. On parallel stochastic simulation of diffusive systems. In: Heiner, M., Uhrmacher, AM., editors. Computational methods in systems biology, volume 5307 of Lecture Notes in Computer Science. Berlin Heidelberg, Germany: Springer; 2008. p. 191-210.

Devine KD, Boman EG, Heaphy RT, Hendrickson BA, Teresco JD, Faik J, Flaherty JE, Gervasio LG. New challenges in dynamic load balancing. Applied Numerical Mathematics. 2005; 52(2):133–152.

Elf J, Ehrenberg M. Spontaneous separation of bi-stable biochemical systems into spatial domains of opposite phases. Systems biology. 2004; 1(2):230–236. [PubMed: 17051695]

Feldmann AE. Balanced partitioning of grids and related graphs. Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20371, 2012. 2012

Foskett JK, White C, Cheung K-H, Mak D-OD. Inositol trisphosphate receptor ca2+ release channels. Physiological Reviews. 2007; 87(2):593–658. [PubMed: 17429043]

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Fujimoto, RM. Parallel and Distribution Simulation Systems. 1st. New York, NY, USA: John Wiley & Sons, Inc.; 1999.

Fujimoto RM, Hybinette M. Computing global virtual time in shared-memory multiprocessors. ACM Transactions on Modeling and Computer Simulation (TOMACS). 1997; 7(4):425–446.

Gillespie DT. Exact stochastic simulation of coupled chemical reactions. The journal of physical chemistry. 1977; 81(25):2340–2361.

Ishizuka N, Cowan WM, Amaral DG. A quantitative analysis of the dendritic organization of pyramidal cells in the rat hippocampus. Journal of Comparative Neurology. 1995; 362(1):17–45. [PubMed: 8576427]

Kirkpatrick S Jr, C G, Vecchi M. Optimization by simulated annealing. SCIENCE. 1983; 220(4598): 671–680. [PubMed: 17813860]

Koizumi S, Bootman MD, Bobanovi LK, Schell MJ, Berridge MJ, Lipp P. Characterization of elementary ca 2+ release signals in ngf-differentiated pc12 cells and hippocampal neurons. Neuron. 1999; 22(1):125–137. [PubMed: 10027295]

Lin, Z., Tropper, C., Ishlam Patoary, MN., McDougal, RA., Lytton, WW., Hines, ML. Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM PADS ’15. New York, NY, USA: ACM; 2015. Ntw-mt: A multi-threaded simulator for reaction diffusion simulations in neuron; p. 157-167.

Lytton, WW. From Computer to Brain. New York, USA: Springer-Verlag; 2002.

Mattern F. Efficient algorithms for distributed snapshots and global virtual time approximation. Journal of Parallel and Distributed Computing. 1993; 18(4):423–434.

McDougal RA, Hines ML, Lytton WW. Reaction-diffusion in the neuron simulator. Frontiers in Neuroinformatics. 2013; 7(28)

Meraji S, Tropper C. Optimizing techniques for parallel digital logic simulation. Parallel and Distributed Systems, IEEE Transactions on. 2012; 23(6):1135–1146.

Meraji, S., Zhang, W., Tropper, C. Proceedings of the 2010 IEEE Workshop on Principles of Advanced and Distributed Simulation. IEEE Computer Society; 2010. A multistate q-learning approach for the dynamic load balancing of time warp; p. 142-149.

Miller RJ. Optimistic parallel discrete event simulation on a beowulf cluster of multi-core machines. PhD thesis, University of Cincinnati. 2010 [last access on May 1st 2015]

Neymotin SA, McDougal RA, Sherif MA, Fall CP, Hines ML, Lytton WW. Neuronal calcium wave propagation varies with changes in endoplasmic reticulum parameters: A computer model. Neural Computation. 2015; 27(4):898–924. [PubMed: 25734493]

Parker I, Ivorra I. Localized all-or-none calcium liberation by inositol trisphosphate. Science. 1990; 250(4983):977–979. [PubMed: 2237441]

Patoary, MNI., Tropper, C., Lin, Z., McDougal, R., Lytton, WW. Proceedings of the 2014 Winter Simulation Conference, WSC ’14. Piscataway, NJ, USA: IEEE Press; 2014. Neuron time warp; p. 3447-3458.

Pilla LL, Ribeiro CP, Coucheney P, Broquedis F, Gaujal B, Navaux PO, Mhaut J-F. A topology-aware load balancing algorithm for clustered hierarchical multi-core machines. Future Generation Computer Systems. 2014; 30:191–201.

Roderick H, Berridge MJ, Bootman MD. Calcium-induced calcium release. Current Biology. 2003; 13(11):R425. [PubMed: 12781146]

Ross WN. Understanding calcium waves and sparks in central neurons. Nat Rev Neurosci. 2012; 13(3):157–168. [PubMed: 22314443]

Schinazi RB. Predator-prey and host-parasite spatial stochastic models. The Annals of Applied Probability. 1997; 7(1):1–9.

Schloegel, K., Karypis, G., Kumar, V. Supercomputing, ACM/IEEE 2000 Conference. IEEE; 2000. A unified algorithm for load-balancing adaptive scientific simulations; p. 59-59.

Sterratt, D., Graham, B., Gillies, A., Willshaw, D. Principles of computational modelling in neuroscience. New York, USA: Cambridge University Press; 2011.

Tsugorka A, Rios E, Blatter LA. Imaging elementary events of calcium release in skeletal muscle cells. Science. 1995; 269(5231):1723–1726. [PubMed: 7569901]

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

Vitali R, Pellegrini A, Quaglia F. Load sharing for optimistic parallel simulations on multi core machines. ACM SIGMETRICS Performance Evaluation Review. 2012; 40(3):2–11.

Wang B, Hou B, Xing F, Yao Y. Abstract next subvolume method: A logical process-based approach for spatial stochastic simulation of chemical reactions. Computational biology and chemistry. 2011; 35(3):193–198. [PubMed: 21704266]

Watkins CJ, Dayan P. Q-learning. Machine learning. 1992; 8(3–4):279–292.

Xu, Q., Tropper, C. Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation. Monterey, California, USA: IEEE Computer Society; 2005. Xtw, a parallel and distributed logic simulator; p. 181-188.

Yao Y, Zhang Y. Solution for analytic simulation based on parallel processing. Journal of System Simulation. 2008; 20(24):6617–6621.

Zheng, G., Meneses, E., Bhatele, A., Kale, LV. Parallel Processing Workshops (ICPPW, 2010 39th International Conference on. IEEE; 2010. Hierarchical load balancing for charm++ applications on large supercomputers; p. 436-444.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

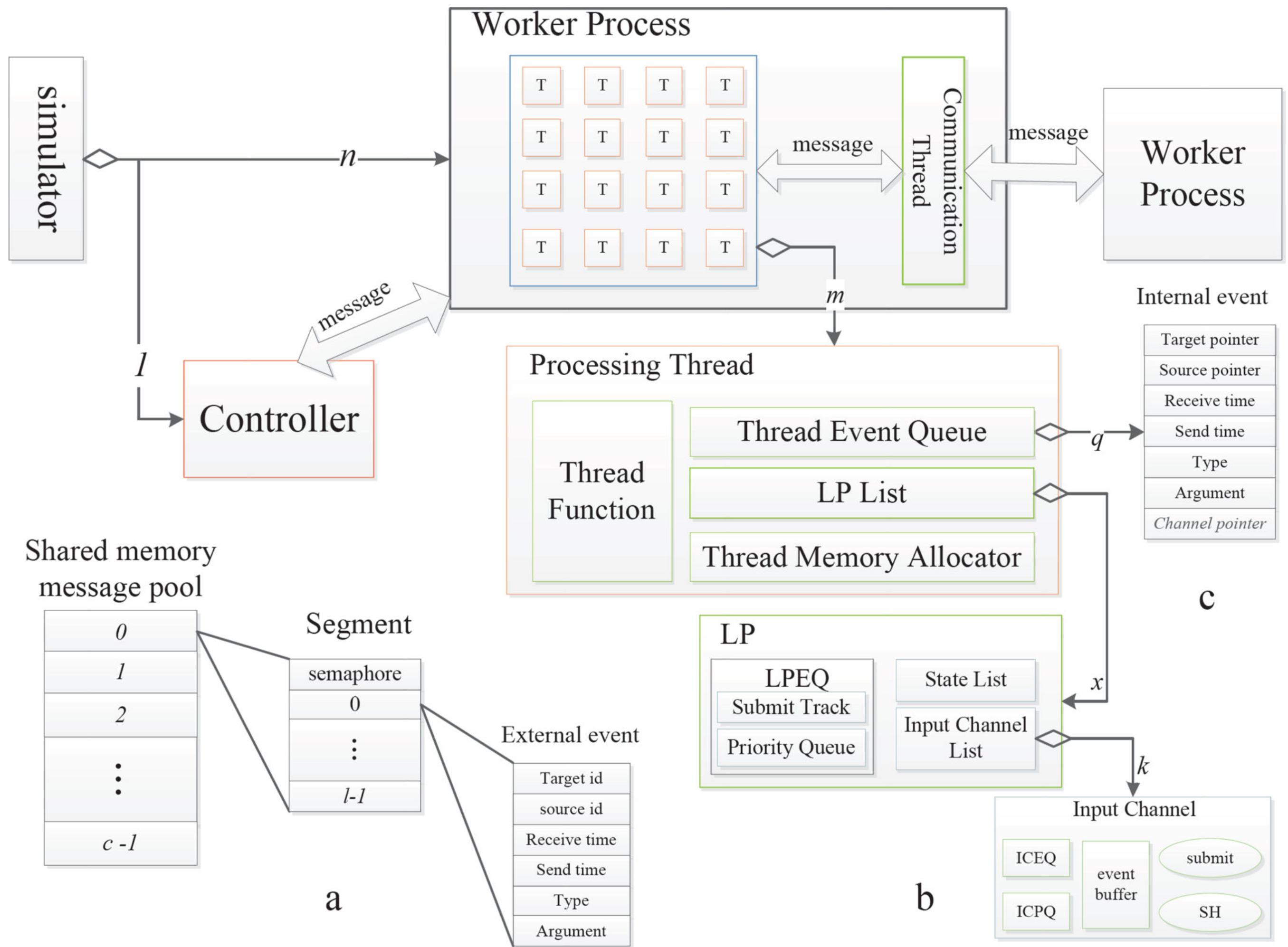


Figure 1.
Architecture of NTW-MT simulator.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

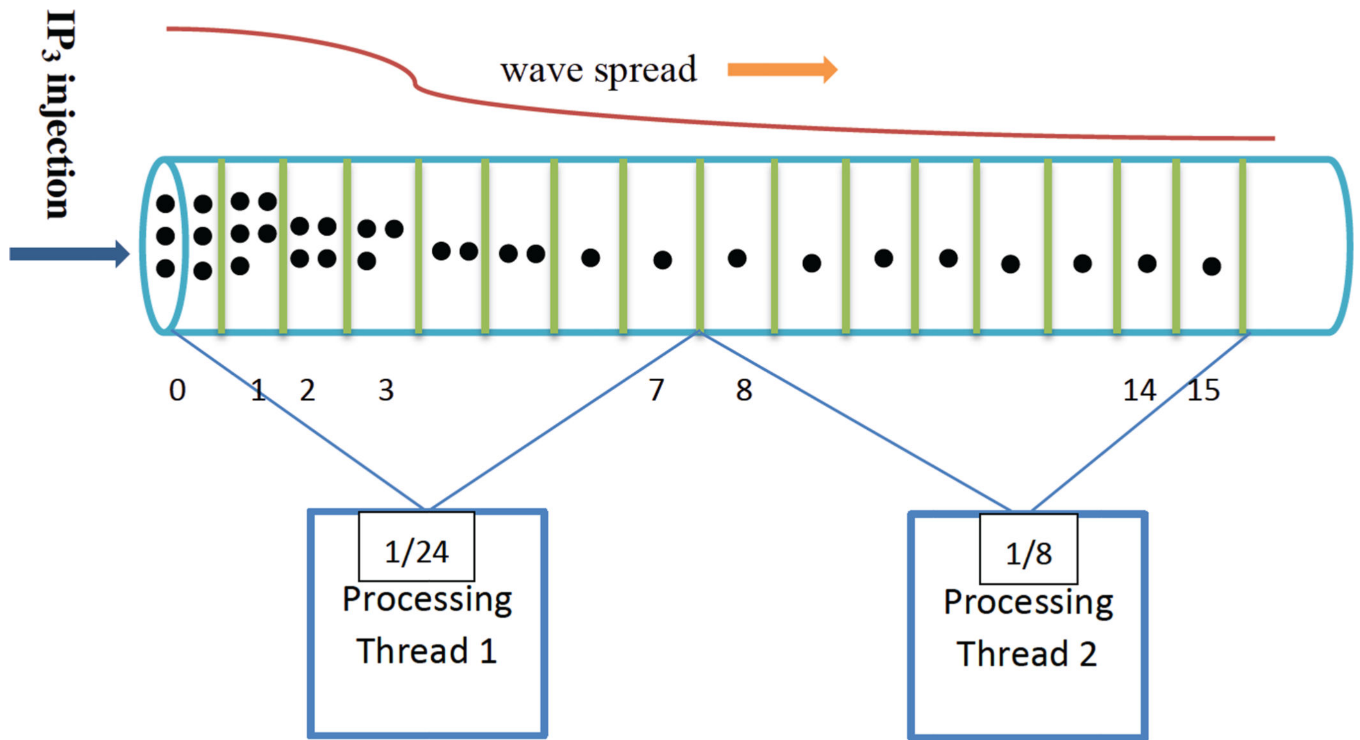
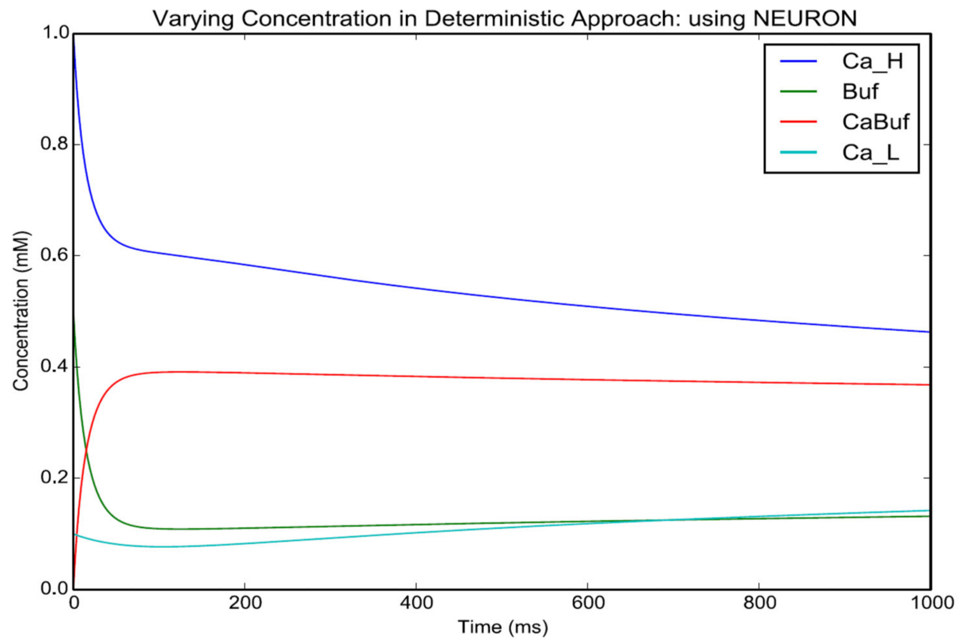


Figure 2.
Unbalanced workload among threads in CICR model.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript



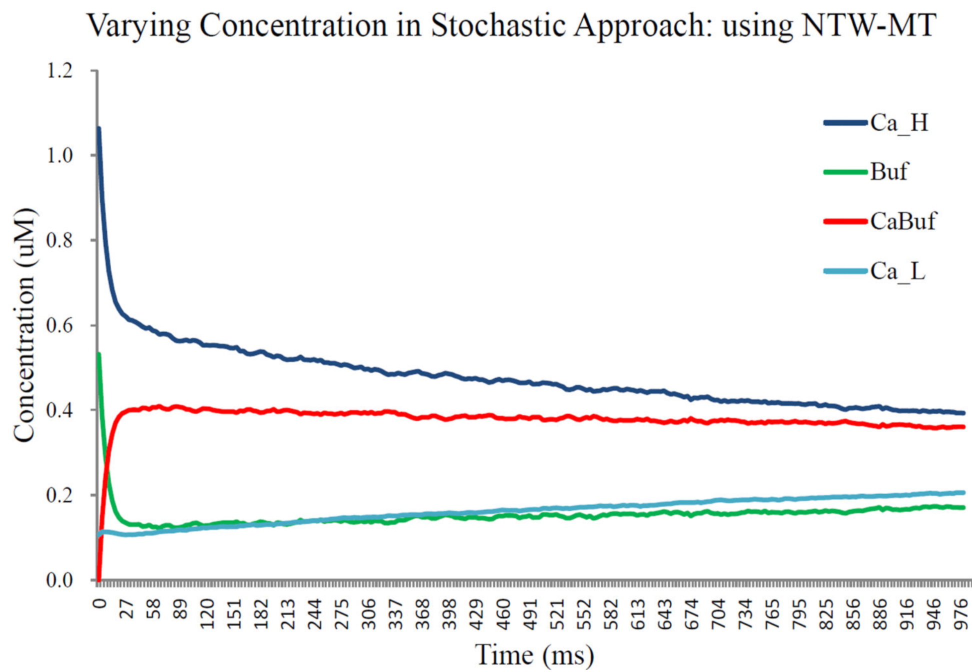


Figure 3.

Deterministic NEURON simulation vs. stochastic NTW-MT simulation, Ca_H and Ca_L refer to high and low concentration of calcium in respective region in deterministic and stochastic simulation.

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

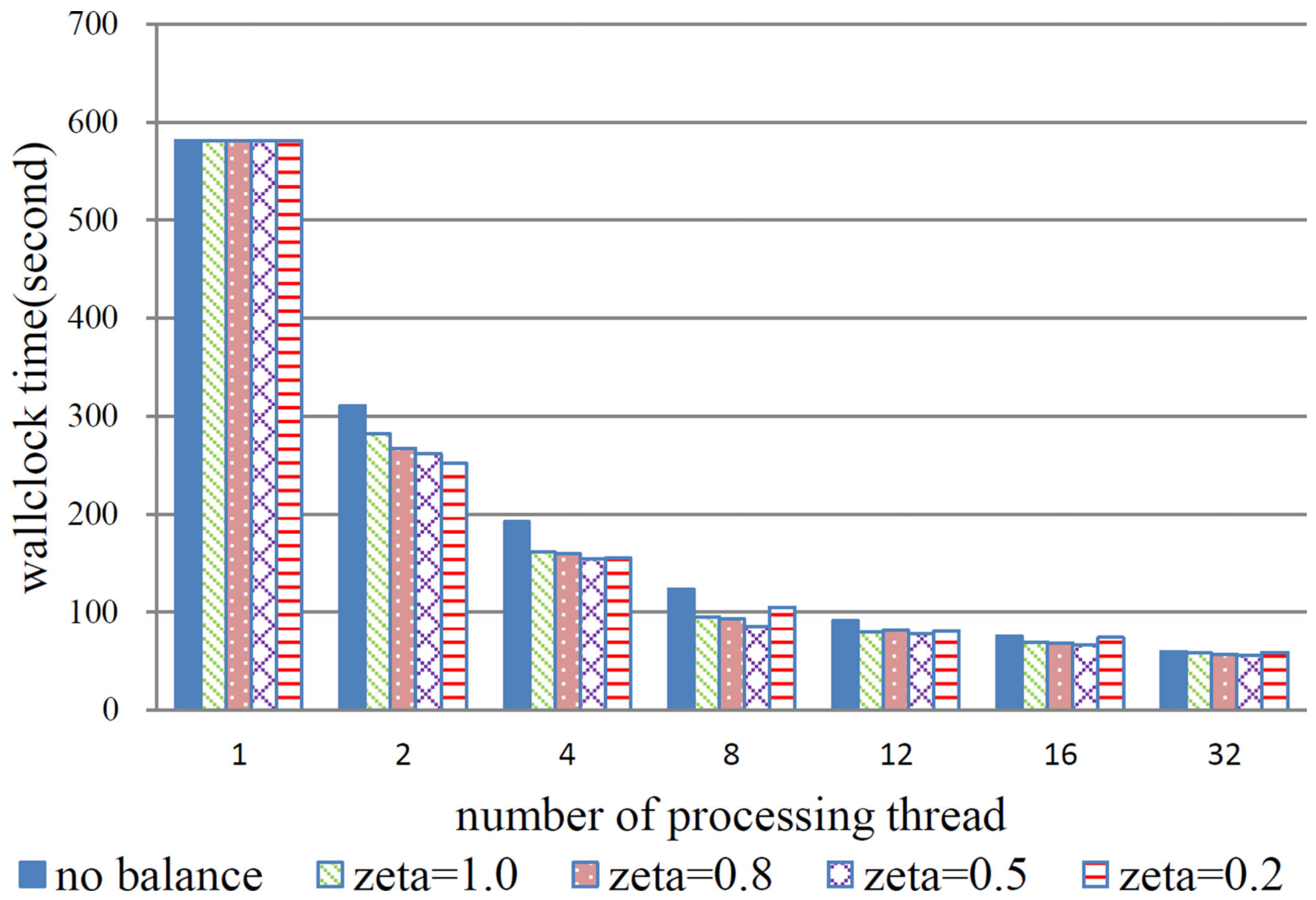


Figure 4.

Execution time on PEPI machine using Q-Learning.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

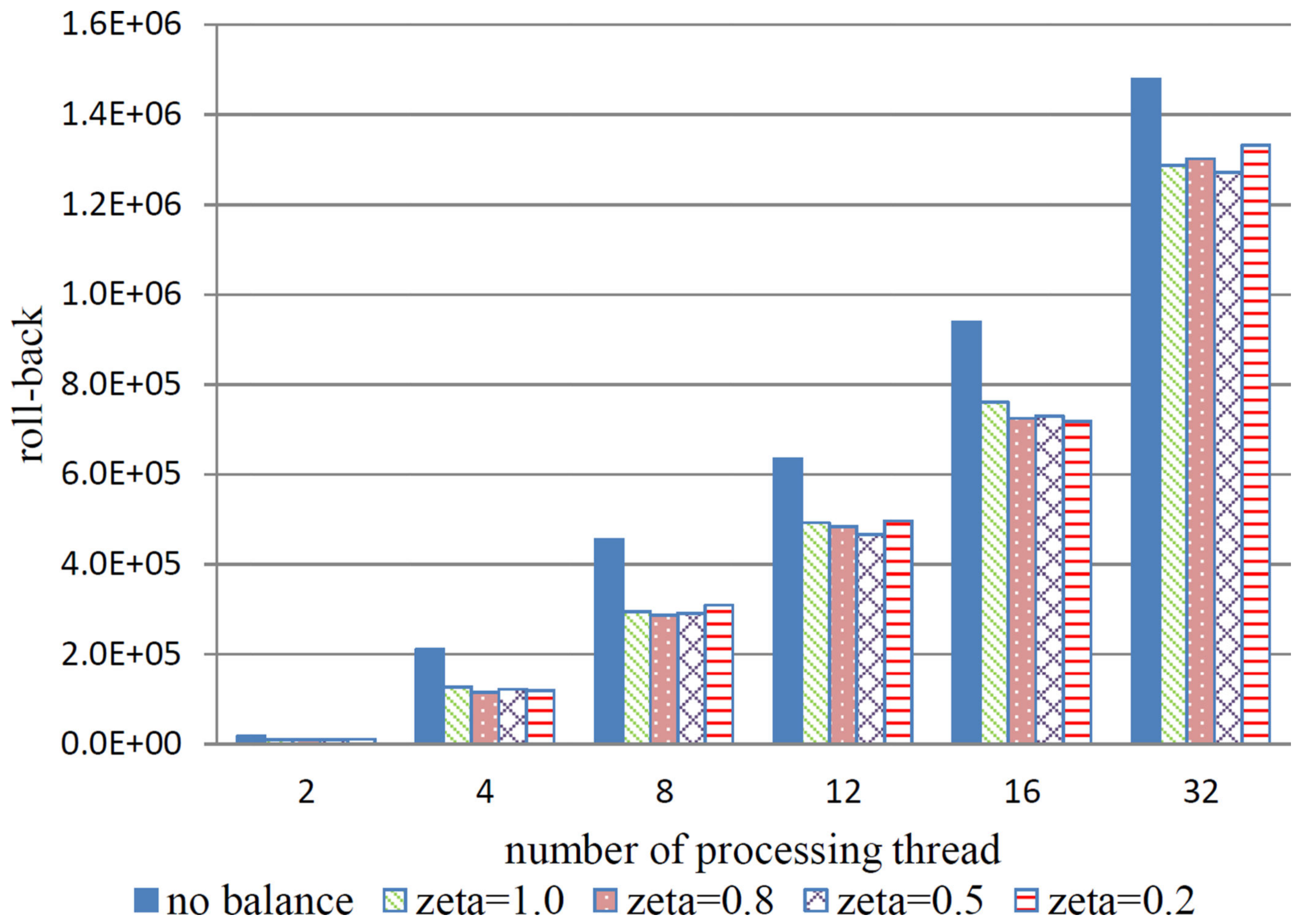


Figure 5.
Roll-backs on PEPI machine using Q-Learning.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

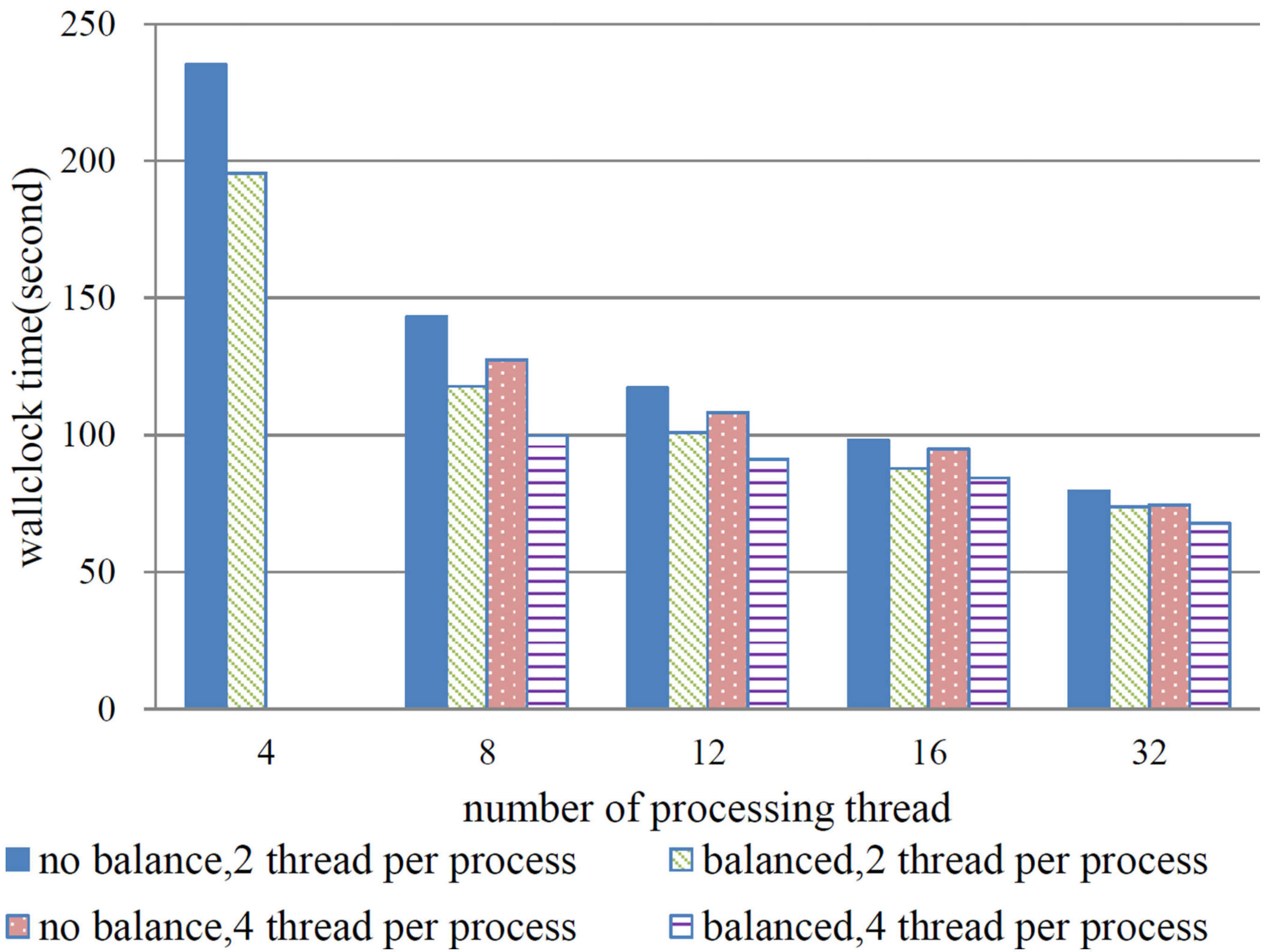


Figure 6.

Execution time with inter-process migration using Q-Learning on PEPI, $\zeta = 0.5$, $L = 0.003$, 0.004 and 0.005.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

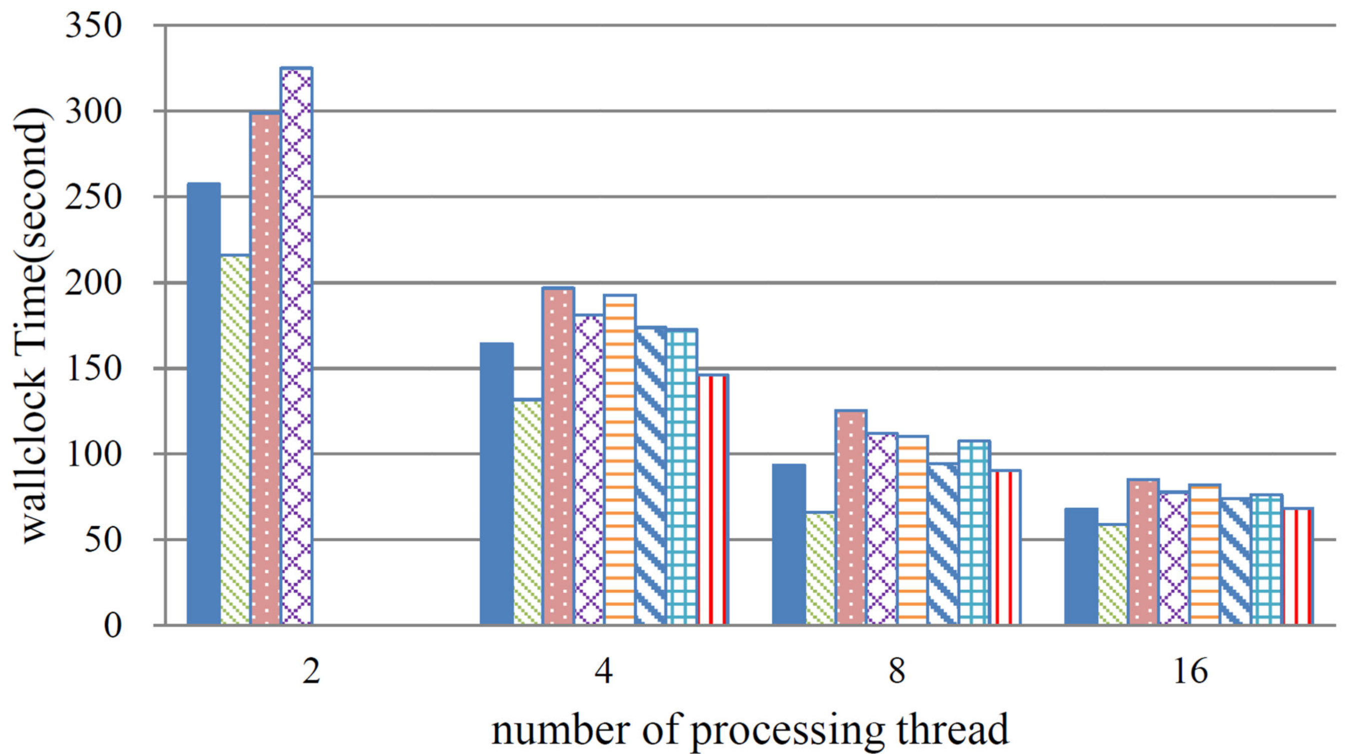


Figure 7.

Execution time with inter-process migration using Q-Learning on Guillimin,, $\zeta = 0.5$, L= 0.003, 0.004 and 0.005, ppn refers to process per node.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

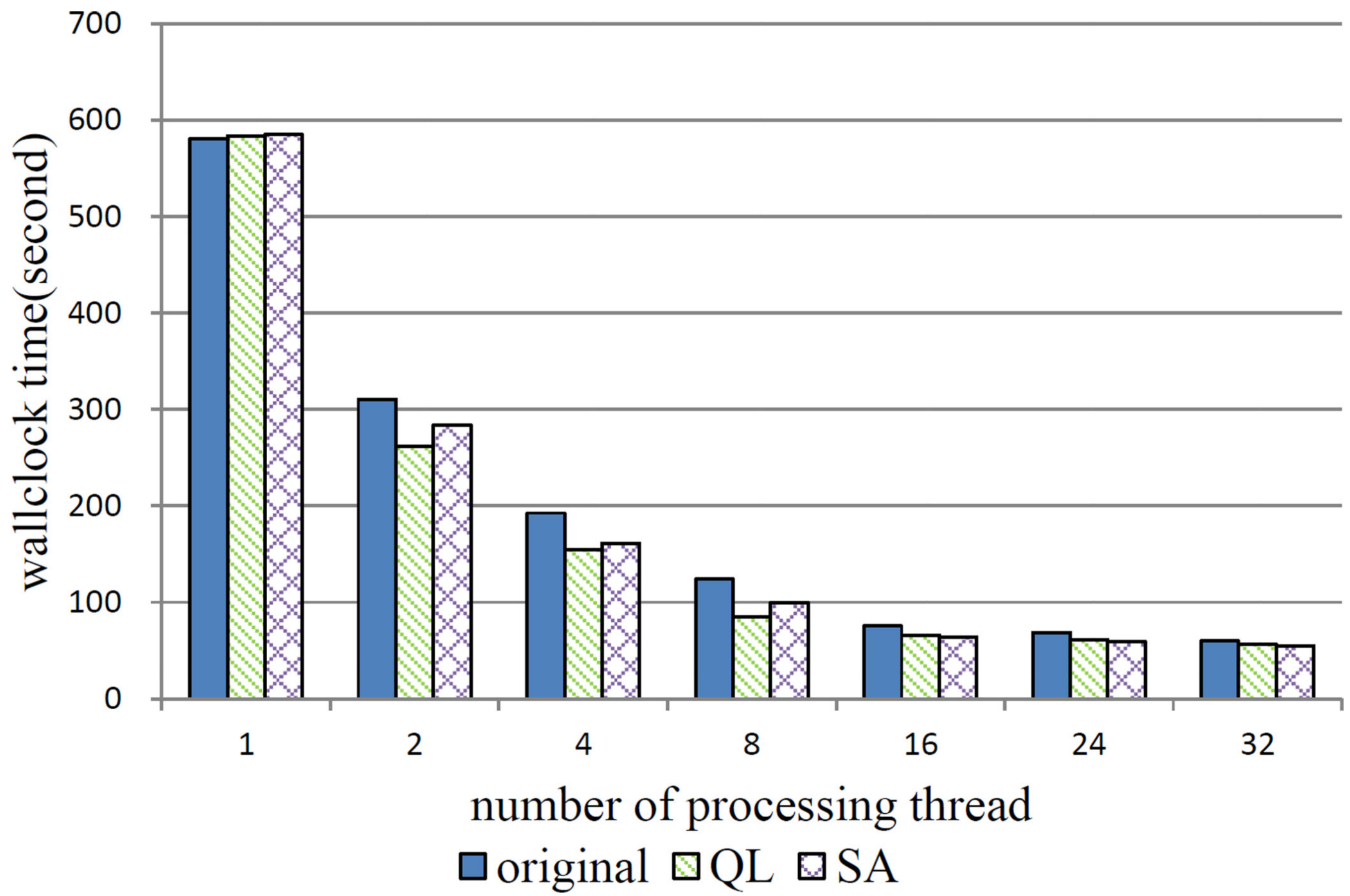


Figure 8.
Execution time using SA on PEPI.

Author Manuscript
Author Manuscript
Author Manuscript
Author Manuscript

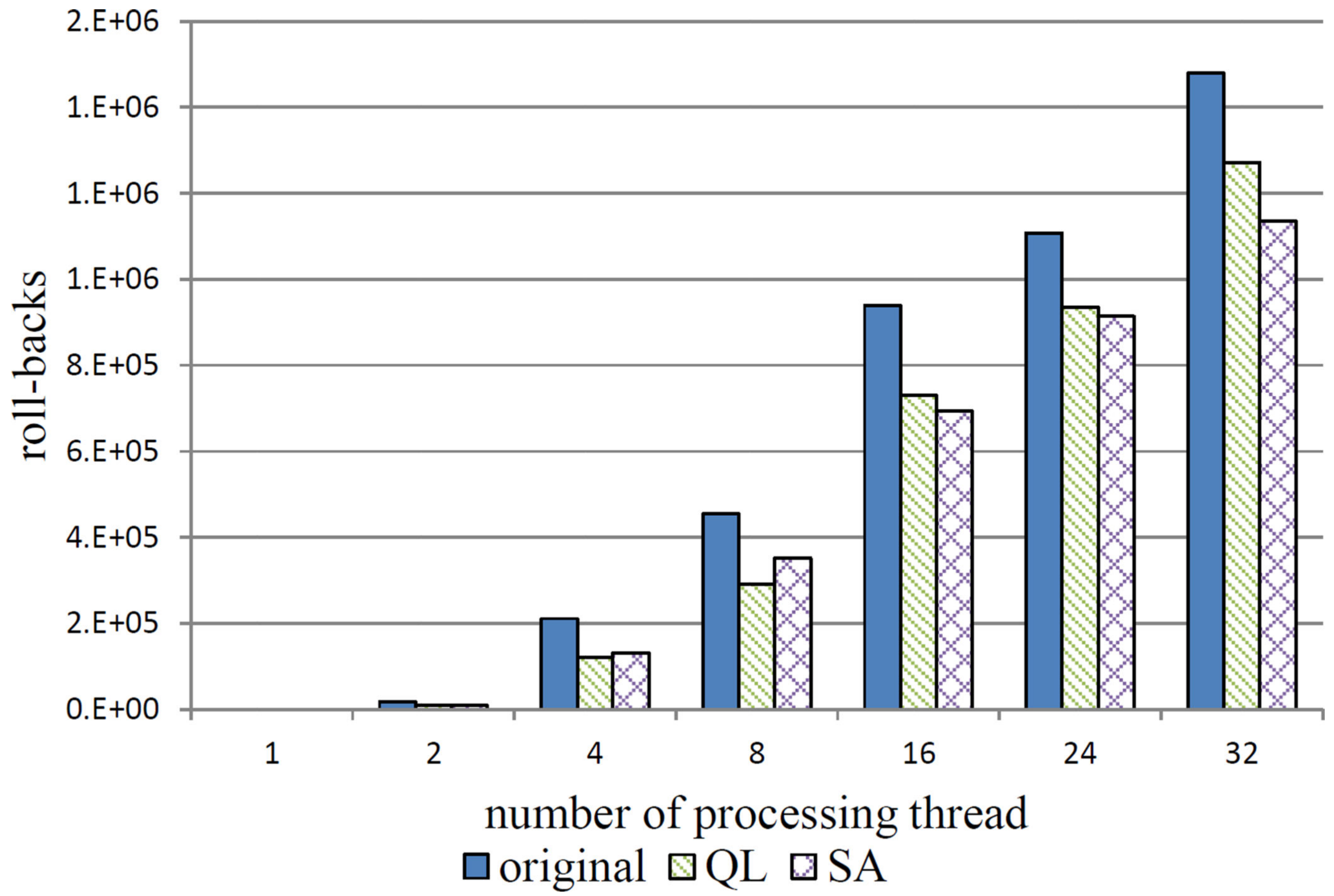


Figure 9.
Roll-backs using SA on PEPI.

Author Manuscript

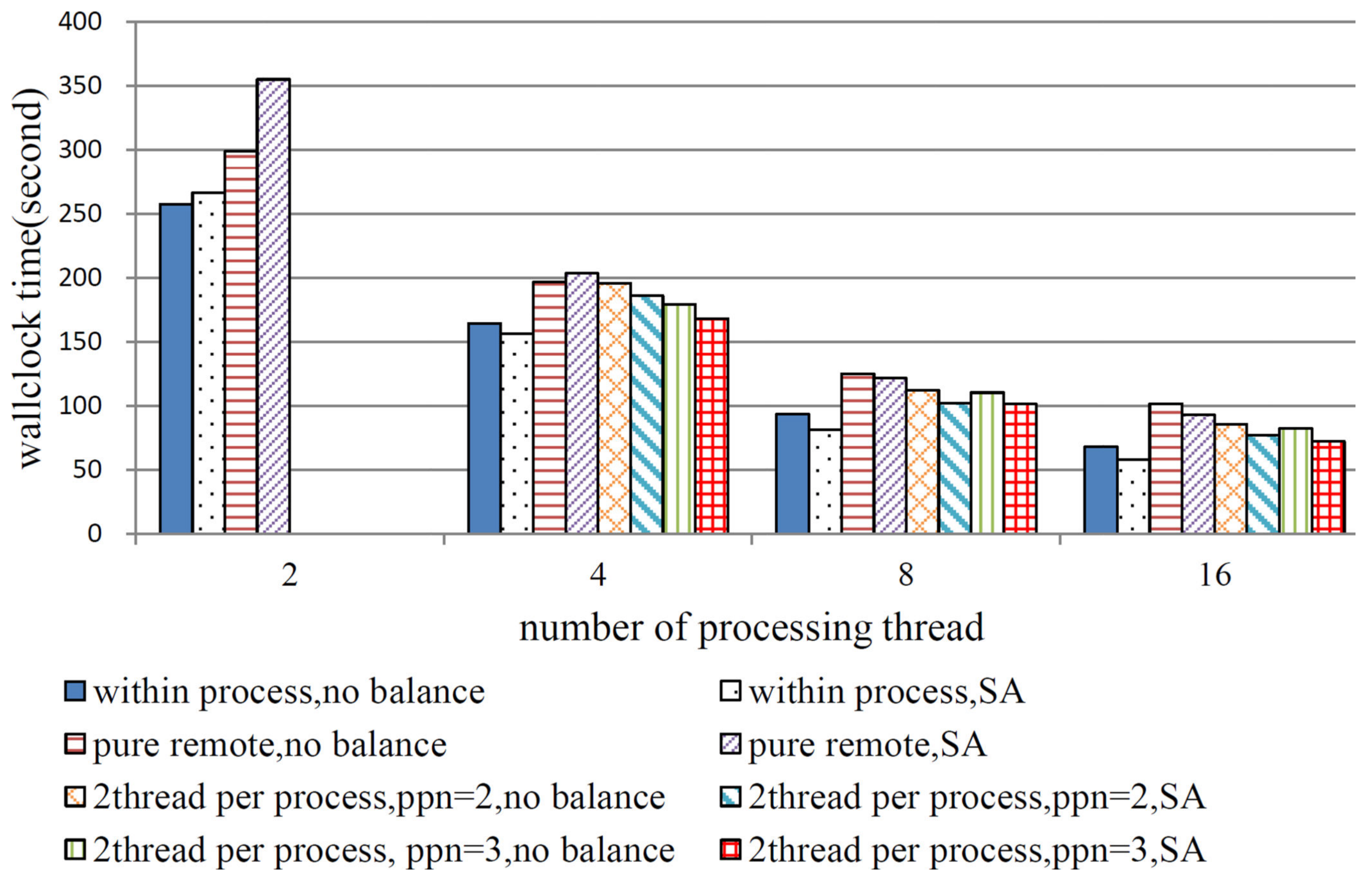


Figure 10.
Execution time with inter-process migration on Guillimin using SA.

Author Manuscript Author Manuscript Author Manuscript Author Manuscript

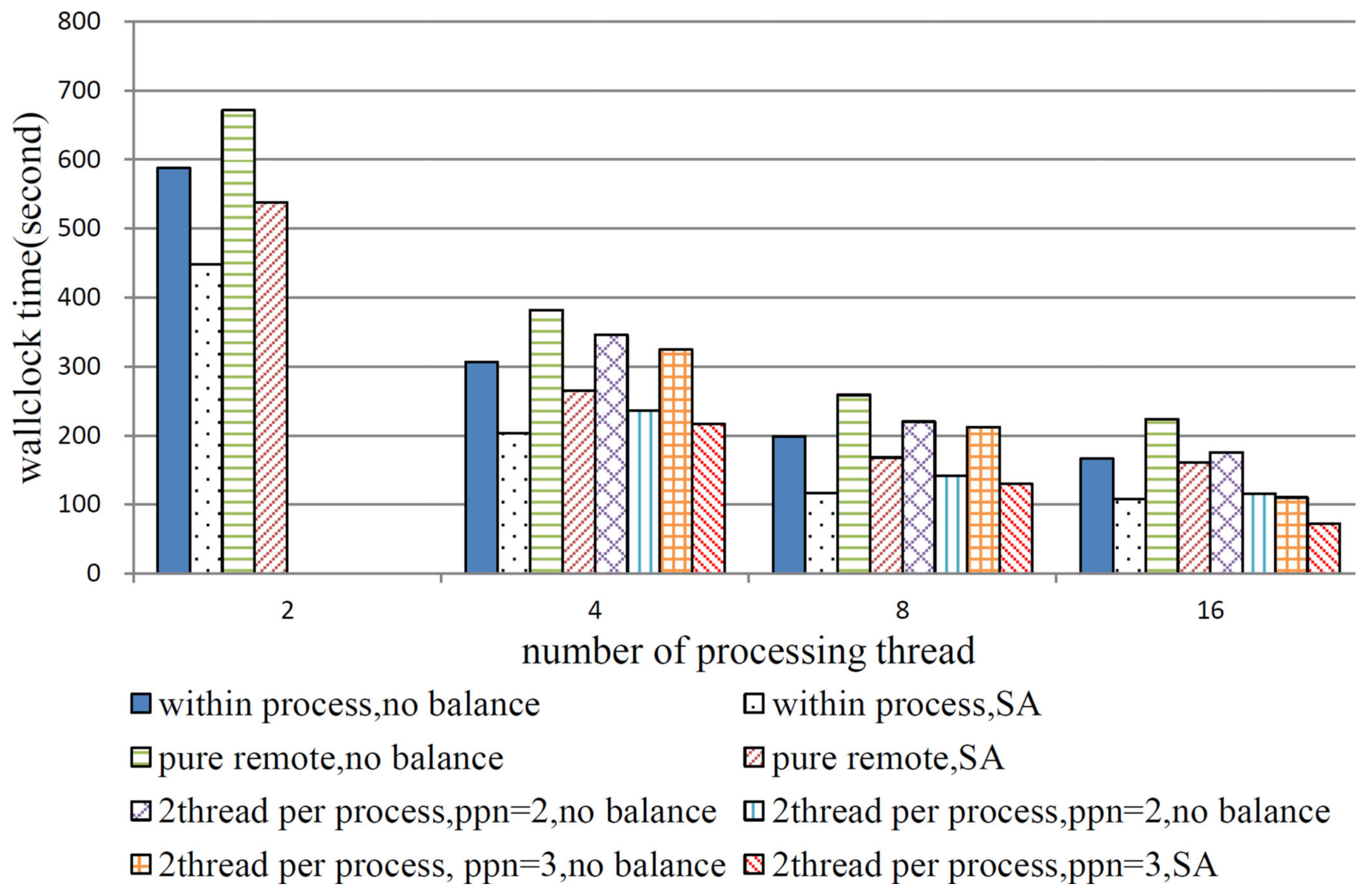


Figure 11.
Execution time for bigger geometry on Guillimin using SA.