UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Performance Model and Load Balancer for a Parallel Monte-Carlo Cellular Microphysiology Simulator

A thesis submitted in partial satisfaction of the
requirements for the degree Master of Sciences
in Computer Science

by

Urvashi Rao Venkata

Committee in charge:

Professor Scott B. Baden, Chair
Professor Keith Marzullo
Professor Henri Casanova

2004

The thesis of Urvashi Rao Venkata is approved.

Chair

University of California, San Diego

2004

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT OF THE THESIS

A Performance Model and Load Balancer for a Parallel
Monte-Carlo Cellular Microphysiology Simulator

by

Urvashi Rao Venkata
Master of Sciences in Computer Science
University of California, San Diego, 2004
Professor Scott B. Baden, Chair

This thesis presents a performance model for a parallel cellular microphysiology simulator and discusses dynamic load balancing techniques that employ it. These simulations are characterized by highly irregular spatial workload concentrations that are determined by the initial data configurations and their time-dependent evolution. A performance model giving reliable spatial and temporal workload estimates is useful in facilitating the adaptive distribution of workload across processors during the course of the computation.

A hybrid performance model was derived and then tuned to accurately estimate and predict the characteristics of each simulation. One component of the performance model is the classical instantaneous model that estimates the workload based on retrospective information about the state of the system. The other is an evolutionary model that augments the instantaneous model by using information about the initial state of the system and the mechanisms that drive it to predict performance trends before they occur. Workload estimates obtained with the hybrid model were used to drive a partitioner that adaptively partitioned the workload and improved the parallel runtime performance by a factor of two on 16 processors.

Chapter I

Introduction

A Monte-Carlo cellular microphysiology simulation is typically characterized by a non-uniform spatial workload distribution. As a result, a parallel implementation using a regular domain decomposition is prone to a severe load imbalance. This leads to a parallel running time that is considerably high compared to an implementation in which all processors are equally loaded. An irregular domain decomposition that partitions regions of higher workload more finely than others, is therefore required. In addition, the workload distribution across the computational domain can vary with time, leading to the need for dynamic load balancing techniques that respond to changes or shifts in workload concentrations by adaptively partitioning the domain. Such load balancing techniques require workload estimates obtained from a reliable performance model.

Performance modeling consists of estimating the running time of an application based on parameters and conditions that drive the computation. One instance where a performance model is useful is efficient run-time load balancing of a parallel application in which varied data dependent behaviour can result in dynamic and irregular workload concentrations. The design of an accurate performance model becomes challenging when there are a variety of data dependent conditions that evolve dynamically through the course of the computation. The direct measurement of the running time per iteration may not always provide reliable and stable estimates, given the overhead of performing fine grained timing, limited timer resolution and inherently noisy running time variations. A performance model based on operation counts and simulation variables is therefore desired.

MCell-K[1] is a parallel simulator used to study sub-cellular communication via signaling pathways and neurotransmitters. It is based on MCell[2], a serial application developed at the Salk Institute for Biological Studies and the Pittsburgh Supercomputer Center. Spatial workload concentrations in a typical MCell run are highly irregular and determined by the initial data configurations and their time-dependent evolution. The use of a static regular domain decomposition can result in the maximally loaded processor holding more than twice the average workload. A performance model giving reliable spatial and temporal workload estimates would be useful in adaptively distributing the workload across processors during the course of the computation. MCell-K simulations are described by an elaborate model description and a single performance model is unlikely to apply under all circumstances. However a good model can enable a computation to scale well, thereby allowing simulations of large proportions to be run in a reasonable amount of time.

This thesis proposes a hybrid performance model for MCell-K, and discusses dynamic load balancing techniques that employ it. Two kinds of cost models are explored. One is an empirically derived short term instantaneous model that estimates the workload based on retrospective information about the state of the system [3, 9, 13, 14, 5]. Some MCell-K events can induce a sudden load imbalance and a purely instantaneous model cannot be used to predict and avoid the resulting imbalance. There is therefore the need for a predictive component that estimates the workload in advance, given information about the initial state of the system and the mechanisms that drive it. The hybrid performance model is validated by its ability to generate reliable and accurate workload estimates for dynamic load balancing algorithms that are based on adaptive irregular domain decompositions.

[1] http://www.cnl.salk.edu/ tilman/index.php?file=home.html
[2] http://www.mcell.psc.edu/

The performance models and load balancing algorithms were designed and validated using the serial MCell code, instrumented to simulate a parallel run. Simulations were performed on sample problems with characteristics typical of a standard MCell run. Load balancing was tested using two domain decomposition techniques. A recursive co-ordinate bisection algorithm was compared to a decomposition based on a Hilbert space filling curve. The load balancing tests revealed that both domain decomposition algorithms give similar results in terms of workload distribution but that the space filling curve algorithm incurred less load balancing computation and communication costs. The use of these adaptive load balancing techniques improved the performance of the simulated parallel run by a factor of two on 16 processors. The varied nature of MCell-K simulations are specified by a complex model description language and warrant a method of creating custom performance models. An analysis of the speedups obtained on 8,16,32 and 64 processors showed that with an accurate workload estimate, these load balancing algorithms lead to scalable computations.

The organization of the material in this thesis is as follows. Chapter II gives an overview of the design of MCell and MCell-K with an emphasis on factors that can affect runtime performance. Chapter III describes the instantaneous and predictive performance models for MCell-K. Chapter IV describes dynamic load balancing algorithms based on these models. Chapter V deals with details concerning the implementation and validation of the parallel simulator used to design the models. Chapter VI discusses the accuracy, stability and efficiency of the performance model and load balancing techniques for a set of sample MCell-K runs. Chapter VII concludes with a summary of the results and a short description of how the production MCell-K code will incorporate the load balancer into its structure.

Chapter II

MCell and MCell-K

II.A MCell

MCell [2] is a cellular microphysiology simulator that uses Monte Carlo methods to trace the movement and behaviour of a large number of reactive molecules called ligands as they diffuse through space in a three dimensional computational domain. These ligands can undergo unimolecular or bimolecular state transitions as they interact with reactive elements called effector sites, located on surfaces and membranes that represent sub-cellular biophysical structures [1]. MCell uses a Model Description Language (MDL) as a high level user interface and link between the steps of model design, simulation and output of results.

II.A.1 Model Description Language

MCell's Model Description Language (MDL) is an extremely versatile way of representing the properties of diffusing molecules and arbitrary surface structures along with diffusion and reaction mechanisms typical of any desired physical system which involves particle diffusion. Parameterized descriptions of the micro-physiological environment along with diffusion and reaction mechanisms are used to drive an MCell simulation. Types of ligand molecules and their properties

[1] Currently ligands interact only with surfaces and not with each other. A future version of MCell will incorporate bi-ligand interactions.

are specified along with the locations of ligand release sites and corresponding release mechanisms. Complex reactive surfaces and membranes are represented using polygonal meshes. Mesh elements are provided with reactive (effector sites), reflective or absorptive properties and reaction mechanisms between the ligand molecules and reactive mesh elements are explicitly defined.

II.A.2 Simulation Characteristics

A typical MCell simulation begins with the release of numerous ligands from release sites according to specified release mechanisms. Computations are organized into iteration timesteps. Within an iteration timestep, each ligand performs a Brownian-dynamics random walk. A maximum diffusion distance specified in the MDL input controls the amount each ligand can move during the course of one iteration timestep. For each ligand the direction and distance to be moved are determined probabilistically, and a three dimensional ray trace operation is performed to compute the new location. In each iteration, ligands take several steps until they reach their maximum allowed diffusion distance. During each step the ray trace operation includes checks for intersections with all walls and mesh elements in the immediate vicinity of the ligand. If a mesh wall is intersected, the ligand may undergo a state change as dictated by the type of mesh element it has encountered. As iterations proceed, ligands diffuse outward from their release sites and can reflect off mesh walls, react with effector sites present on them, be absorbed by binding sites located on mesh walls, or interact in any other way defined in the MDL input. Ligand binding sites maintain ligand saturation counts and may undergo variations in reactive properties based on the number of bound ligands associated with them. Output can be obtained from MCell in various forms. Counts and statistics can be obtained with respect to any class of participating elements of the simulation. Output can also be generated in a format compatible with the DReAMM[2] three dimensional visualization tool (Figure II.1.A).

[2] http://www.mcell.psc.edu/DReAMM/about.htm

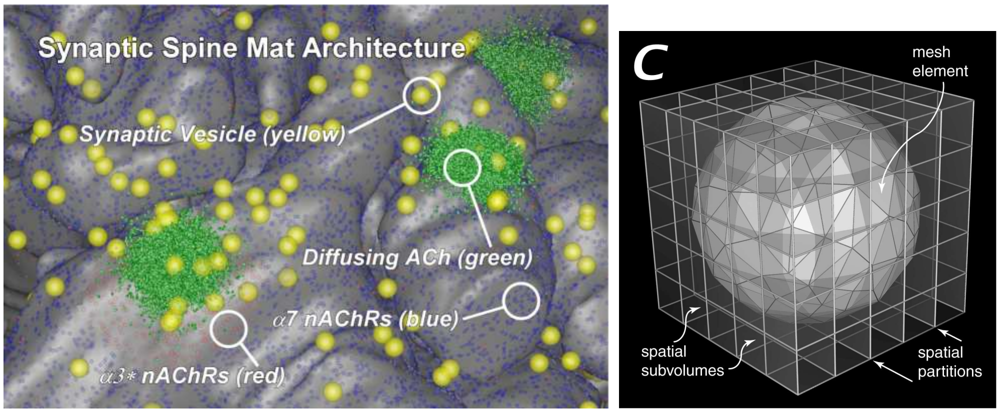

Figure II.1: (A (left)) Three dimensional visualization of ligands diffusing out from release sites in the chick ciliary ganglion MCell simulation (T. Bartol and T. Sejnowski) (B (right))Spatial subvolumes in MCell with planes along the X,Y and Z axes along with a spherical polygon mesh surface(J.Stiles and T. Bartol).

II.A.3 Implementation

The three dimensional computational domain is divided into a large number of spatial sub-volumes (SSVs) formed by the tensor product of a set of partitions along each dimension (Figure II.1.B). This limits the computation required to track individual ligands by requiring that a ligand check for intersections only with the mesh walls physically located within the SSV enclosing the ligand. During the ray trace operation, if a ligand encounters a subvolume wall, its subvolume identifier is updated and the ligand takes its next step [3].

Ligands are represented as data structures holding state and location information about a ligand. There is a global linked list of ligand structures that is traversed through during each iteration timestep and ligand movements and reactions are recorded as state changes. For the purpose of statistical accuracy there is no correlation between the ordering of ligands in this list and the location

[3] In the current implementation the SSVs are represented on a cartesian grid, whereas the mesh walls and surfaces are based on a triangulated three dimensional mesh. An alternate grid representation is being implemented in which both the SSVs and the mesh walls will be represented using a three dimensional tetrahedral mesh. This will allow some mesh tiles and SSV walls to coincide, thus reducing the number of intersections to be checked for during ray tracing.

of the ligands in the computational domain. Surface mesh walls are also represented as linked lists of mesh elements holding information of their location, orientation, reactive properties and saturation counts. During each ray trace step along a ligand's trajectory, the part of the mesh wall linked list representing mesh elements contained in the current SSV is traversed to check for intersections.

The radial direction and step length for the movements of each ligand are controlled by a set of random numbers that correspond to Monte Carlo probabilities derived from bulk solution rate constants of a Brownian-dynamics random walk algorithm. MCell uses a 64-bit random number generator and splits each number to obtain random numbers for the direction and length of a ligands step. MCell also allows for checkpointing wherein the state of the system at a particular iteration can be saved in a log file for the current state of the simulation to be examined. The simulation can then resume along with additional of changed MDL information.

II.A.4 Running Times

The main determinants of the running time of a typical serial MCell run are the number of release sites, ligands and mesh walls in the computational domain and the number of iterations required for the simulation to run to completion. A serial MCell run with 20 release sites and 100000 ligands, simulated with the micro structure of a chick ciliary ganglion takes approximately 2 hours to complete 3000 iterations on a single 2.2GHz processor, and uses approximately 500MB of memory. In a typical run the number of free ligands reduces considerably after about 2500 iteration timesteps as they bind to mesh walls and stop diffusing. A full production run involving all 550 release sites would run for over 2 days. These runs usually have to be repeated several times to ensure statistical accuracy of the results, and the total running time required soon becomes prohibitive.

II.B MCell-K

MCell simulations possess complex spatial and temporal dynamics that can potentially limit scalability in a parallel environment. Also due to the dynamic nature of migrating particles and the requirement of numerical and physical accuracy in the simulation, MCell is not embarrassingly parallel and requires careful synchronization. A parallel variant of MCell, called MCell-K[1], has been implemented using the KeLP parallel programming infrastructure. KeLP enables the management of distributed pointer-based structures, has mechanisms to handle data migration among processors, and can facilitate load balancing via overdecomposition. These parallel programming paradigms make KeLP a good choice for implementing a parallel variant of MCell.

II.B.1 Parallelization Strategy

A regular domain decomposition is used to partition the computational domain in a one-to-one mapping of partitions to processors. Initially, all processors read the input file and create local copies of geometry and other simulation data pertaining to their local regions of space in the computational domain. Mesh elements that cross processor boundaries are duplicated on multiple processors. Each processor maintains local linked lists for ligand structures and mesh elements. The ligands in the local processor lists are arbitrarily ordered with respect to their physical location within the processor's computational domain.

Processor boundaries are introduced into the MCell computational domain as walls of a particular type, and regular MCell ray trace operations are used to detect when a ligand steps across a processor boundary. Each iteration timestep is split into several sub-iteration timesteps. Ligands whose movements do not result in intersections with processor walls are completely updated in the first sub-iteration timestep. Ligands that record intersections with processor walls, are transfered to a communication list. At the end of each sub-iteration step, if any ligand that has not yet been completely updated has registered a hit with a processor boundary, there is a synchronization step where all ligands flagged for communication are sent to their destination processors. An iteration terminates only when all ligands in the entire computational domain have been updated.

II.B.2 Results

Speedups

MCell-K was tested with sample MDL input data and produced numerical and statistical results in agreement with those produced by the serial code [1]. A large MCell simulation of ligand diffusion through the chick ciliary ganglion mircostructure was performed with 192 release sites and 5000 ligands per release site on the NPACI Blue Horizon [1]. Under typical conditions the ligands decay as iterations proceed, but this run was modified to simulate the case of all ligands remaining active in the system at each timestep. The simulation using MCell-K was run for 2500 iteration timesteps and took 81.7 minutes to complete on 16 processors and 44.2 minutes on 32 processors. (A serial run of this size would take approximately 6 hours on a 2.2GHz processor.) Most ligands were completely updated in the first sub-iteration timestep and the additional time required for sub-iteration synchronization was found to be negligible compared to the total running time.

Communication Overhead

Communication events in MCell-K are frequent but involve message sizes of only a few kilobytes. Communication costs per iteration were found to be negligible in comparison to the running time per iteration. The maximum number of ligands (over all processors) communicated per timestep was found to be inversely proportional to the number of processors used. This, along with the small message sizes involved, suggests that communication costs may not be significant even in large runs with a large number of processors.

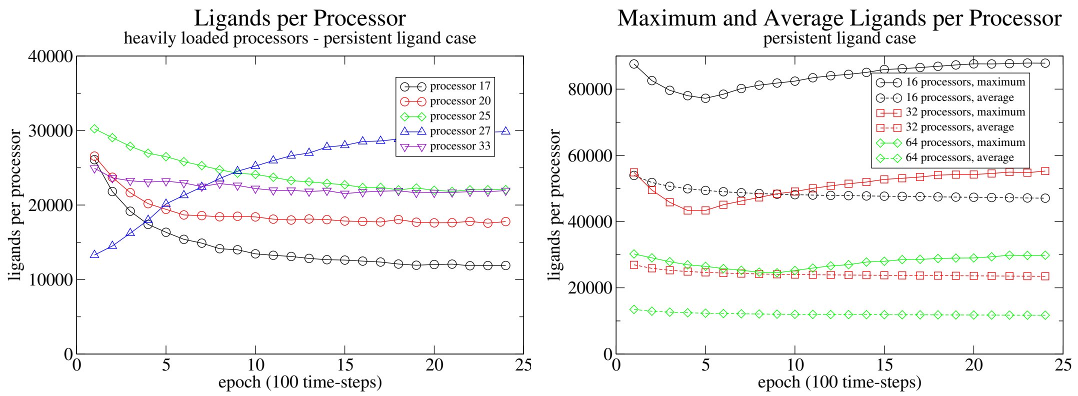

Figure II.2: Dynamic Load Imbalance in MCell-K (A (left)) Ligands per processor for each of the most heavily loaded processors of a 64 processor MCell-K run (Gregory T. Balls [1]). (B (right)) Maximum and average ligand counts over all processor for 16,32 and 64 processors (Gregory T. Balls [1]). Time is measured in epochs, each epoch corresponding to 100 iterations.

Load Imbalance

Load imbalance was found to be a major bottleneck in the system. In a run of the chick ciliary ganglion simulation the initial load imbalance was limited by initializing the simulation such that the distribution of ligand release sites over processors was within a factor of two of the average number of release sites per processor.

Figure II.2.A shows the number of ligands per processor for the most heavily loaded processors as they vary as a function of iteration number. (Reported times are averaged over epochs, each corresponding to 100 iteration timesteps). Ligands were observed to rapidly diffuse across processor boundaries resulting in a high load imbalance that persisted throughout the rest of the simulation. Plots of the maximum load over all processors for each iteration timestep also showed that the total running time is affected by a load imbalance with the maximally loaded processor carrying approximately twice the average (ideal) load in the system (Figure II.2.B).

It can therefore be seen that the dynamic nature of the diffusing ligand distributions and the static but irregular concentrations of mesh elements results in a considerable load imbalance in a parallel implementation that uses a static regular domain decomposition. This necessitates the use of irregular domain decompositions that divide some regions of high workload more finely than others, as well as dynamic load balancing strategies that can detect changes or shifts in workload concentration and then perform an adaptive domain decomposition and workload redistribution.

Chapter III

Performance Model

The first step in analyzing the performance of an application for the purpose of designing a performance model, is to identify the operations that dominate the running time. These operation counts can then be used to obtain an estimate of the workload and predict running times. The running time of a parallel application is determined by the time taken by the most heavily loaded processor. In applications like MCell-K, since the work is divided among processors by partitioning the discrete three dimensional computational domain into subvolumes of space, the running time of the parallel simulation will depend on spatially dependent counts such as free ligand density and the number of mesh walls that ligands can intersect with, in each partition subvolume. An instantaneous model can use the operation counts of one iteration to predict the spatial workload distribution for the next iteration. This model can detect an increase in running time due to a load imbalance in the system only after it occurs, but it can be combined with a predictive model in which workload estimates that could indicate the need for load balancing, may be obtained prior to the actual computation that could cause the load imbalance.

This chapter describes the main computational components of a typical MCell run and discusses the formulation of instantaneous and supporting predictive models for a parallel MCell-K run.

III.A MCell workload statistics

The workload in a typical MCell run depends on the number and density of diffusing ligands and their proximity to walls and effector sites. An irregular distribution of effector sites on (irregular) mesh surfaces results in irregular workload concentrations. An irregular distribution of release sites results in highly concentrated workloads in some regions in the computational domain. This is especially significant during the first few hundred iterations which represent a large fraction of MCell's total running time (Figure III.2.B).

III.A.1 Workload Measure

To obtain a workload estimate, various counts and timing statistics are collected over a spatial grid of Workload Sample Boxes (WSB) in the computational domain. MCell uses the SSV grid to speed up searches for encounters with mesh surfaces. For the purpose of load balancing the WSB grid used is a coarsened version of the MCell SSV grid and load redistribution is carried out in terms of units of work determined by workload estimates per WSB.

A practical choice of the resolution of the WSB grid depends on the average diffusion rate of the ligands as specified in the MDL input. To ensure the numerical accuracy of the simulation and to minimize the movement of ligands across processor boundaries within an iteration, the distance along each edge of a sample box must be greater than the maximum distance a ligand can diffuse during one iteration timestep. The workload sample boxes chosen for the experiments described in this thesis each enclose $8 \times 8 \times 8$ SSVs.

The overhead of collecting these count statistics is low. For each ligand, a linear index mapping between the SSV index and the corresponding WSB index is used to accumulate counts for each sample box. The performance model is based on various counts and timings obtained by treating each sample box as a unit of work.

```
1.  for (each timestep) {
2.    for (each ligand) {
3.
4.        if isbound (ligand) give it a chance to unbind
5.
6.        if isfree (ligand) {
7.            call ray_trace (ligand) {
8.                while (ligand has not diffused far enough) {
9.                    move the ligand
10.                   check for intersection with nearest SSV walls
11.                   check for intersection with SSV's mesh wall list
12.                   }
13.           if (wall hit) {
14.               process subvol wall hit
15.               process mesh wall hit
16.               }
17.           }
18.       }
19.   }
20. }
```

Figure III.1: Code structure of the core MCell computation loop.

The core of the MCell computation consists of tracking each ligand in the system for a series of iteration timesteps. The processing of each ligand depends on its state (bound or free) and its proximity to mesh walls and effector sites (collisions and reactions). To identify the operations that dominate the running time, various operation counts were monitored and timings obtained for corresponding code segments. The code structure of the core MCell computation loop is shown in Figure III.1 and the counts and times measured are listed in Table III.1.

Counts	**Times**	**Operations**	**Lines**
c_{total}	t_{total}	Total Ligand Count and Running time	2-19
c_{free}	t_{free}	Free Ligand Count and Ray Trace time	7-17
c_{ssv_checks}	t_{ssv_checks}	Checking for SSV wall hits	10
c_{ssv_hits}	t_{ssv_hits}	Processing SSV wall hits	14
c_{mesh_checks}	t_{mesh_checks}	Checking for mesh wall hits	11
c_{mesh_hits}	t_{mesh_hits}	Processing mesh wall hits	15

Table III.1: Measured Operation Counts and Timings in Fig III.1. The design and analysis of the performance model was done in terms of these counts and timings.

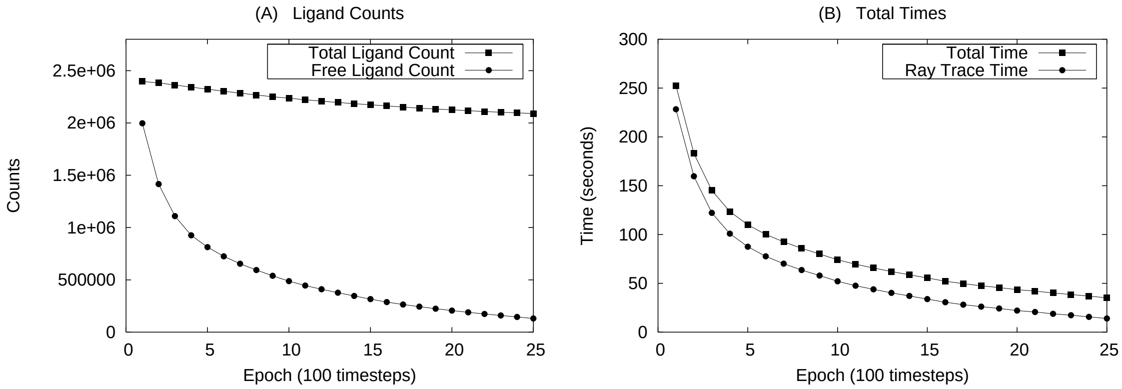

Figure III.2: (A) Total ligand count (c_{total}) and Free ligand count (c_{free}). (B) Total time (t_{total}) and Time spent on free ligands in ray trace (t_{free}).

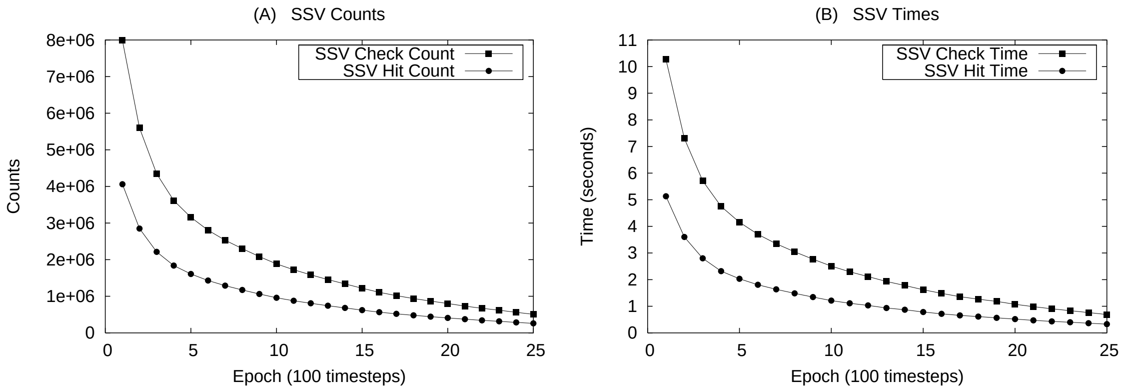

Figure III.3: (A) Number of SSV walls checked (c_{ssv_checks}) and hit(c_{ssv_hits}). (B) Time spent checking for (t_{ssv_checks}) and processing (t_{ssv_hits}) SSV wall hits.

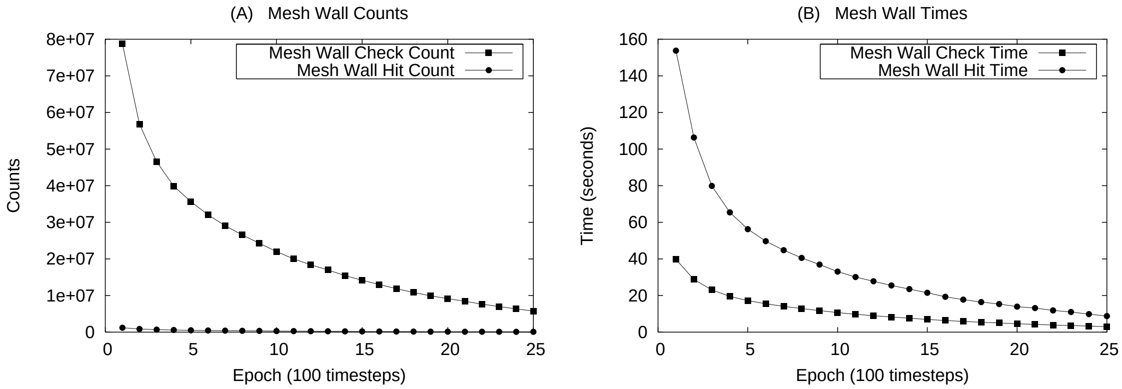

Figure III.4: (A) Number of Mesh walls checked (c_{mesh_checks}) and hit(c_{mesh_hits}). (B) Time spent checking for (t_{mesh_checks}) and processing (t_{mesh_hits}) mesh wall hits.

III.A.2 Observations

Timings and corresponding operation counts were measured for a sample serial run of a chick ciliary ganglion MCell simulation. This simulation had 32 release sites positioned near mesh walls, each releasing a train of 1000 ligands in the first iteration timestep. As the iterations proceeded the ligands diffused away from their release sites and began interacting with mesh walls and surfaces. Some ligands began to bind while others continued to diffuse.

Total Running Time

Figure III.2.A shows the total ligand count (c_{total}) and compares it to the free ligand count (c_{free}). The number of free ligands in the system is seen to drop rapidly as the ligands encounter binding sites on mesh elements. Figure III.2.B shows the total time spent in updating the ligands(t_{total}) and the time spent in the ray trace function call (t_{free}). It can be observed that the total running time of a serial MCell run is dominated by the computation done on free ligands in the ray trace function call.

Ray Trace operations

Within the ray trace function there are two main contributors to the running time - the checking and processing of intersections with SSV walls and the checking and processing of intersections with mesh walls. Figure III.3.A shows counts of the number of SSV walls that are checked for possible ligand encounters (c_{ssv_checks}) along with the number of SSV walls actually intersected by the diffusing ligands (c_{ssv_hits}). Figure III.3.B shows the corresponding times ($t_{ssv\_checks}, t_{ssv\_hits}$). The time spent checking for SSV walls is greater than that spent processing intersections. Regions of high free ligand concentrations will incur higher t_{ssv_checks} and c_{free} will be able to estimate the time spent checking for SSV wall intersections. Figure III.4.A shows counts of the number of surface mesh walls that are checked for intersections (c_{mesh_checks}) and counts of mesh elements

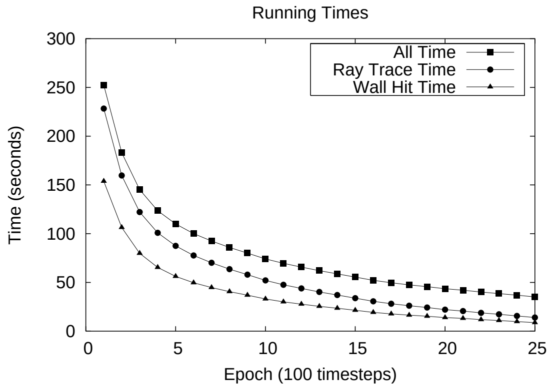

Figure III.5: Chick Ciliary Ganglion : Total running time(t_{total}), time spent in the Ray Trace function call (t_{free}) and the time spent processing intersections with mesh walls (t_{mesh_hits}).

that register ligand encounters(c_{mesh_hits}). Figure III.4.B shows the corresponding times ($t_{mesh\_checks}, t_{mesh\_checks}$). It can be observed that although the number of mesh walls checked for intersections greatly exceeds the number of mesh walls actually intersected, the running time in this segment of the code is dominated by the time spent processing intersections with mesh walls. Figure III.5 compares t_{total},t_{free} and t_{mesh_hits} and shows that the total running time is dominated by the time spent in processing intersections with mesh walls.

III.A.3 Analysis

Absence of mesh walls

In the absence of mesh walls all ligands are free and all the time is spent in the ray trace function call, checking for and processing intersections of ligands against SSV walls. The time taken per ligand to check for and process intersections with SSV walls does not vary much with location since there is no spatial variation in the density of SSV walls (the SSVs form a regular grid). Therefore in the absence of mesh walls, the total time spent in the ray trace function call is directly proportional to the number of free ligands.

Presence of mesh walls

The presence of mesh surfaces affects the running time depending on their spatial distribution as well as local ligand densities. The time taken per ligand to check for intersections with mesh walls depends on the number of mesh walls in the current SSV, and this shows a large spatial variation. Upon intersection with a mesh wall, a ligand can undergo several different state changes and the accompanying computations dominate the running time. Figure III.5.A shows that the total running time follows a trend that closely matches that of the time spent processing intersections with mesh walls. Intersections with mesh walls and surfaces are spatially dependent operations and the corresponding computational load varies with location in the computational domain. Since the running time of a parallel application involving the spatial sub-division of the computational domain is decided by the maximally loaded processor, these spatially dependent operation counts can be used to estimate the running time of a parallel implementation.

III.B Instantaneous model

This model assumes that in a typical MCell run, the state of the system varies smoothly between successive iterations and that there are no abrupt changes in workload on the timescale of one iteration. This assumption is based on the observation of running times and operation counts which vary smoothly from iteration to iteration (Figure III.2.B). A workload estimate used to predict the running time at any given iteration timestep $(t + 1)$ can be based on a weighted average of spatially dependent operation counts obtained during the previous iteration timestep (t).

We must note that the perfect instantaneous workload estimate for an iteration timestep would be the measured running times on each processor for the previous step. This however is not practical[1]. The use of operation counts

[1] The timer resolution places a lower limit on the number of operations that can be accurately timed. In certain systems and simulations there can be noticeable overheads of using a timer function call.

to generate a workload estimate is efficient, and can be used along with prior information to predict load imbalances and take corrective measures before they occur.

Workload Estimate: An empirical workload estimate can be constructed from counts corresponding to operations whose running times dominate the total running time. The total running time (t_{total}) is dominated by the number of free ligands (c_{free}) and the processing of their intersections with mesh walls (c_{mesh_hits}). There is also a finite computation time associated with each ligand, bound or free (c_{total}). These counts vary considerably with location in the computational domain and can be used to obtain an estimate of the running time of a parallel MCell run. A workload estimate WL^{t+1} at the iteration timestep $t+1$ can be given as a weighted average of c^t_{total}, c^t_{free}, and $c^t_{mesh\_hits}$ measured during the t^{th} iteration timestep.

$$WL^{t+1} = a * c^t_{total} + b * c^t_{free} + c * c^t_{mesh\_hits} \tag{III.1}$$

where the model parameters a,b and c are to be chosen such that this weighted sum of spatially varying operation counts predicts trends observed in the overall running time over all WSBs.

Model Parameters: Running times of all spatially varying computations were totaled over the entire computational domain and used to compute values of a, b and c via a linear least squares fit over all iterations of a test run. Since the times and counts used in the model are those of spatially varying computations (ray trace function calls and mesh wall intersections) this model will also be able to estimate the workload per spatial subdomain (sample box (WSB) or processor).

Now, it is not possible to use this method to obtain parameter values for the model function during an actual parallel MCell-K run that requires dynamic

Running time profiles sometimes include spurious spikes caused by factors like memory and disk I/O and other system processes. In addition, a load imbalance and corresponding change in running time can be detected only after it occurs, and it may take a few iterations to recover. Using the running times as the workload estimate does not allow the use of prior information for the model to be predictive. Also, the variations in running time across iterations are inherently noisy and the use of individual iteration timings in load balancing decisions may result in frequent unnecessary load balancing steps.

load balancing. Also, note that it is only required that the profile of the running time be estimated and not the actual absolute running times. Any scaling of the total workload estimate will be valid as an estimate. Therefore, values of a,b and c that hold the same ratios with each other as those computed using a representative MCell-K run would suffice.

In the absence of mesh walls the running time depends on the number of free ligands within each sample box, with the operations done on the free ligands deciding the running time. Since all free ligands constitute an equal amount of work, and there are no further spatial effects that need to be factored in, the ligand count per WSB is a good estimator of the running time profile of such a parallel run.

The presence of mesh walls in the computational domain adds additional spatially varying workloads. The computation per sample box now depends on both the spatial ligand distribution as well as the density of mesh walls within the sample box. The model was trained on an actual run which simulated the sub-cellular dynamics in a chick ciliary ganglion with 32 release sites each releasing 1000 ligands during the first iteration. The simulation was allowed to run for 2500 iterations. It was observed that the following ratio was a good estimator of the running time profile of a parallel run in which the diffusing ligands interacted with mesh walls and effector sites (Figure III.6).

$$a : b : c = 1 : 7 : 10 \tag{III.2}$$

In the absence of mesh walls, only the first two terms remain, but since $c_{total} = c_{free}$, the estimate from Equation III.1 is still valid. In practice, for a general MCell-K run a scaled down serial version of the simulation can first be run to obtain the counts required for the model along with measured total times. Values of a, b and c can then be found via a least square fit. As long as these parameters are obtained from spatially varying counts and times, the model will be representative of an actual parallel run.

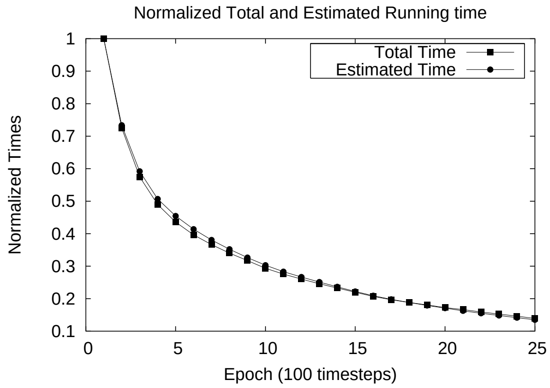

Figure III.6: Chick Ciliary Ganglion : Total measured running time (t_{total}) and estimated running time using the instantaneous model (WL). The times are normalized to the same scale.

The following is a possible strategy of obtaining reliable workload estimates from the performance model. The performance model can be trained for each individual data set. Typically, MCell runs are repeated several times to ensure statistical accuracy. One such run can be used as a sample run in which a generic model is applied, and operation counts and timings are gathered. This can then be used to find more accurate model parameters (Equation III.1) that are tuned to the actual simulation. This more accurate model can then be applied to the remaining simulation runs.

III.C Predictive model

In an MCell run, it is sometimes possible to predict the occurrence of a simulation event that may adversely affect the running time. A predictive workload estimate obtained from such information can complement the instantaneous model and allow for a potential load imbalance to be predicted before it actually occurs. This section describes some predictable simulation events and characteristics that can affect the parallel running time of MCell-K.

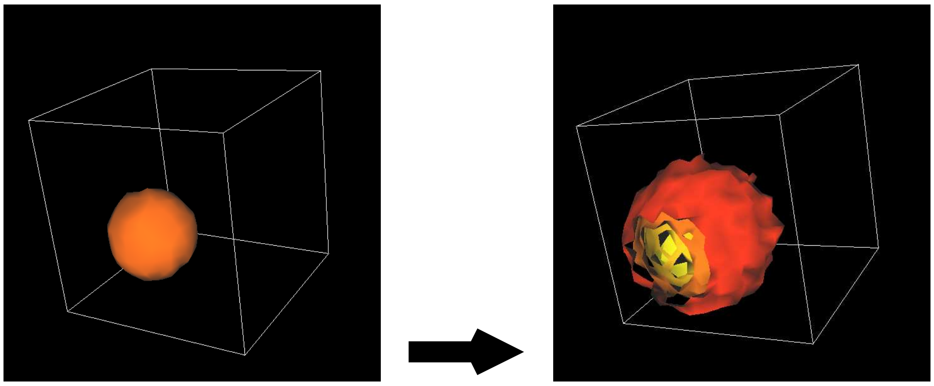

Figure III.7: Unbounded Diffusion - Gaussian Ligand Density Profiles. Three dimensional visualization of ligand clouds at iteration 25(A) and 50(B).

III.C.1 Temporal spatial load variations

In the absence of obstructing mesh walls, clouds of ligands diffuse spherically outward from release sites with a density profile similar to that of an expanding three dimensional Gaussian. It is this behaviour that MCell is meant to model. Once the ligands are released, they are allowed to diffuse out in all directions according to a three dimensional random walk. The ligand diffusion rates specified in the MDL input can be related to the observation of Gaussian density profiles, to form a basis for a model that can predict spatial workload distributions, and hence establish whether a load balancing step is needed at a future iteration.

In the presence of obstructing mesh walls however, this is not practical. The computational overhead of precisely predicting the spatial distribution would be more than the overhead due to allowing the simulation to become imbalanced for a few of iterations until it is detected by the load balancer.

III.C.2 Overall running time trends

Most of the total MCell running time is spent in the first few hundred iteration timesteps. This is because most released ligands begin life as active and free and later either bind with effector sites or are destroyed. As the iterations proceed, there are generally fewer free ligands in the entire system, thus reducing

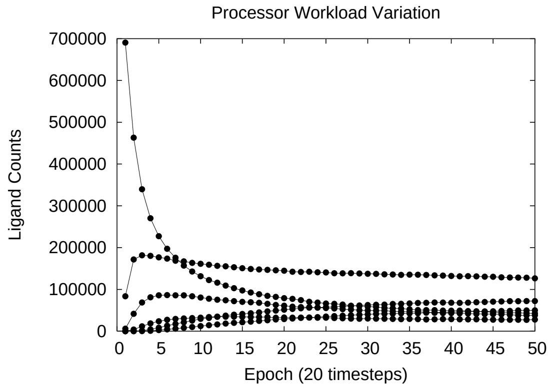

Figure III.8: Processor workload variation for Unbounded Diffusion and a static regular decomposition.

the total parallel running time. This observation suggests that emphasis must be placed on being able to predict trends accurately during the crucial early timesteps of the simulations.

Usually, MCell simulations begin with very high ligand densities around release sites. With a static initial decomposition (regular or adapted to the initial workload distribution based on release site locations) ligands diffuse out and soon cross sample boxes and processor boundaries. Once the particles diffuse out considerably, the workload is closer to being naturally balanced (Figure III.8). This observation also suggests that load balancing may be required more frequently in the first few hundred iterations and can be relaxed later.

The amount of computation performed in each sample box over time can also be predicted from the initial ligand distributions within each sample box. A workload sample box (WSB) with an initially high density of free ligands is likely to see a decrease in free ligand count and the associated running time as some ligands begin to bind and some diffuse out. In a WSB that begins empty, ligands can diffuse into it and begin reacting with mesh walls, resulting in an initial increase in running time as ligands diffuse into the WSB, followed by a leveling off and decrease in workload as the ligands begin to diffuse out or bind (Figure III.9).

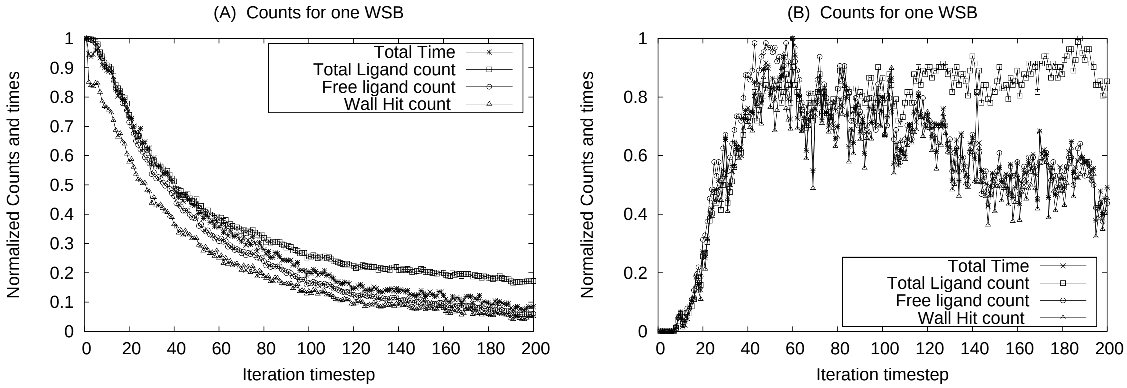

Figure III.9: (A) Normalized running times and counts for a sample box containing a ligand release site. (B) Normalized running times and counts for a sample box that is initially empty and contains mesh walls with effector sites.

III.C.3 MDL input

The MDL input describes the initial state of the system and the mechanisms that drive the MCell simulation. Therefore, these specifications hold considerable information that can be used to predict some dynamic workload trends.

1. The distribution of release sites along with counts of ligands to be released during the first few iterations, provides a good estimate of the initial spatial workload distribution. This can guard against high load imbalances during the first few (up to 5) iterations, when the workload concentrations are inherently high and irregular (Figure VI.9).

2. MDL parameters that specify the binding and reaction probabilities, effector site saturation limits, ligand destruction probability and other such factors, control the number of free ligands in the system. These parameters can be monitored during the simulation, to predict workload concentrations before they actually occur [2].

[2] A new version of MCell is being developed in which a scheduler will be used as a performance optimization to avoid a large number of conditional tests that are guaranteed to be false under certain local or global conditions. The scheduler will monitor the state of the system and decide when to execute these operations. The operations include checks for encounters with mesh walls and effector sites as well as ligand releases.

3. Ligand diffusion rates specified in the MDL input can be used with the Gaussian diffusion model to predict how workloads are likely to change with time. This however is possible and practical only in regions of the computational domain without obstructing mesh walls.

4. Some MCell simulations involve periodic ligand releases, specified via release site object parameters in the MDL input. A mapping between the release train interval and actual iteration timestep number can be used to predict an imbalance that can occur if a large number of ligands are suddenly released into previously sparse locations in the computational domain. The load balancer can then be called just before the releases with an artificial workload distribution and this can prepare the system for the sudden influx of new work. This kind of prediction can also be done if sudden releases depend on dynamic parameters of the simulation. The load balancer can monitor these parameters and use a tighter threshold than that used for the releases, to predict a large imbalance before it occurs. Simulation results with prediction will be discussed in Chapter VI.

An accurate prediction is not always possible or helpful using information from the MDL input. For instance, some MCell simulations involve ligand releases at random points in time, from random locations in space. This cannot be predicted and the load balancer will have to resort to an instantaneous workload estimate based on an imbalance sensor that is sensitive enough to detect the imbalance within an iteration of it occurring. Also, ligand releases that do not significantly change the workload density over parts of the computational domain are not disruptive to the workload balance and can be ignored with regard to the predictive model.

Chapter IV

Load Balancing

The performance model described in Chapter III was validated by using the workload estimates it generated as a basis for load balancing. This chapter describes the adaptive domain decomposition algorithms used, and discusses some heuristics used to make the load balancing dynamic.

IV.A Domain decomposition

Irregular spatial workload concentrations in MCell necessitate the use of an adaptive irregular domain decomposition, in which areas with high workload concentration are divided relatively finely compared to areas with lower workloads. Two domain decomposition methods explored were Recursive Co-ordinate Bisection (RCB) and a one dimensional partitioning along a 3D Hilbert Space Filling Curve (SFC). Both methods were applied to a computational domain gridded in three dimensions at the scale of the sample boxes (WSB), with each grid cell holding the current workload estimate for the corresponding region of space.

IV.A.1 Recursive Co-ordinate Bisection

Recursive co-ordinate bisection involves a hierarchical partitioning of space in which the domain is first split into two subdomains of equivalent computational load by the geometric division of the enclosed volume along a co-ordinate axis.

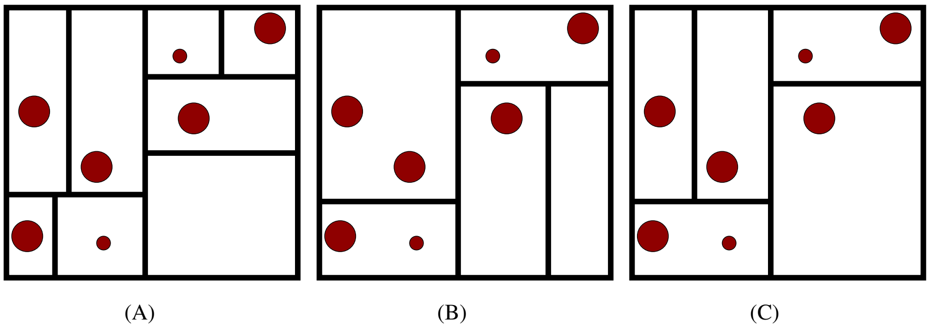

Figure IV.1: Schematic diagram for Recursive Co-ordinate Bisection with different termination criteria : (A)Height of the tree, (B)Total number of partitions, (C)Maximum workload density. The dots represent spatially concentrated workloads with the size of the dot corresponding to the amount of work at that spatial location.

The two subdomains may not enclose volumes of the same size, but will contain approximately equal workloads. Each subdomain is then further divided into two partitions, each with approximately half the total workload of the parent domain. This is a recursive process which continues until terminated by one of the following heuristics.

Recursion can be terminated on the basis of (A) recursion tree depth, (B) a limit on the total number of partitions formed in the system, or (C) a predetermined estimate of maximum allowable workload per partition (Figure IV.1 A,B,C). In the first case, the number of partitions will always be a power of two, but partitions are not guaranteed to have approximately equal workloads and might even be empty, especially if workloads are highly concentrated and irregular (Figure IV.1.A). The second approach is sometimes useful when trying to generate partitions that carry a one-to-one mapping with available processors but is implementation dependent and can also result in unevenly distributed workloads. One way of avoiding this is to overdecompose the domain and then perform an in-order traversal through the leaves of the recursion tree and create a many-to-one map-

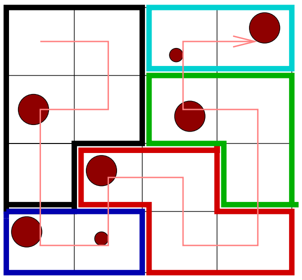

Figure IV.2: Schematic diagram for Hilbert Space Filling Curve Partitioning.

ping between partitions and processors. The third method is generally the most practical in which a sub domain is always further partitioned unless its workload count is below a certain fraction of the total workload. This allows regions with higher workload densities to be partitioned more finely than others and is the method chosen for the simulations.

IV.A.2 Hilbert Space Filling Curve

A three dimensional Hilbert space filling curve passes through the cells of a three dimensional grid following a one dimensional path. Such a traversal ensures locality, with every adjacent pair of cells in the one dimensional traversal being physically adjacent to each other in three dimensional space. The space filling curve is generated via a recursive, fractal based algorithm where the base traversal path on a cube follows a gray code ordering of its eight vertex co-ordinates. The one dimensional traversal along the space filling curve corresponds to a linear traversal through the leaves of the fractal recursion tree.

Once a mapping from three dimensional co-ordinates to the one dimensional index has been computed, a partitioning algorithm can traverse the one dimensional path and assign grid points, in this case sample boxes, to processors. A one dimensional recursive co-ordinate bisection routine or a simple average based binning algorithm can be used to assign WSBs to processors (Figure IV.2).

IV.A.3 Optimality

A comparison between the two methods in terms of practicality and optimality of a good domain decomposition technique for MCell-K is given below.

Optimal Load Division

A load balancing step ideally results in each processor holding a workload equal to the average load on the system. The average workload can be computed from global information about the total load and number of available processors. A recursive co-ordinate bisection partitioner is vulnerable to the generation of empty partitions, and requires overdecomposition and a many-to-one mapping between partitions and processors to achieve an optimal decomposition (Figure IV.1). A many-to-one mapping however introduces an additional level of partitioning and mapping in an already overdecomposed partitioning at the scale of the workload sample boxes (WSBs), and this may result in a computational overhead.

A space filling curve partitioner on the other hand directly uses the overdecomposition at the scale of the WSBs and is able to generate partitions with a one-to-one mapping, with each processor holding a workload as near to the average as possible.

Locality

It is desired that sample boxes adjacent to each other in three dimensional space, be grouped onto the same processor. This is based on the observation that ligand populations move gradually and workload shifts per iteration are mainly between nearby sample boxes. Both methods are capable of preserving locality.

Irregularly Shaped Partitions and Load Redistribution

If processor partitions are restricted to regular shapes as in recursive co-ordinate bisection, the workload balance is limited by the average size of the partitions per processor. This is because the workload redistribution involves the transfer of slices of space corresponding to groups of sample boxes between processors rather than individual sample boxes. A one dimensional decomposition on the space filling curve however allows for irregularly shaped partitions where load balancing can be done by distributing workload at the scale of single WSBs. This results in a finer load distribution over processors (Figure IV.2).

Communication

Communication between processors occurs either due to ligand diffusion across processor boundaries or workload redistribution during load balancing.

Ligand diffusion : The surface to volume ratio of the processor partitions will affect the amount of communication between processors. Regularly shaped partitions have a smaller surface to volume ratio than irregularly shaped partitions, and are hence subject to less communication due to ligand diffusion. The space filling curve based partitioning results in irregularly shaped partitions and more communication may be required. This however does not result in a drastic communication overhead because of the locality inherent in the space filling curve. Also, the MCell-K results [1] described in Chapter II show that the cost of ligand communication in a typical MCell-K run is negligible in comparison to the total running time. We therefore ignore communication due to ligand diffusion, in our performance analysis and load balancing algorithms.

Load Balancing Workload Redistribution : Whenever the load balancer is invoked and a load redistribution is performed, WSBs and the associated ligand lists are to be communicated between processors. To minimize communication of sample boxes between processors, changes must be based on the current load distribution rather than recomputing over the entire domain. The recursive co-ordinate bisection algorithm can be made incremental by combining pairs of leaves and redividing them based on the projected workload. However, this process may require that the tree be unraveled up to higher levels or to the root, and this will increase communication. The space filling curve algorithm is a one time computation. A map is created from three dimensional cartesian co-ordinates to a one dimensional traversal. This map is then used to do a one dimensional workload distribution between nearest neighbour processors based on global information. Since the map is computed only once, there is very little overhead of the workload redistribution during a load balance call.

The space filling curve algorithm is therefore a more efficient approach than recursive co-ordinate bisection and would be a better choice for an adaptive load balancing algorithm.

Limitation due to size of sample boxes

One point to note is that both methods are limited by the granularity of the computational domain at the scale of the sample boxes. It is entirely possible that a single WSB may contain an extremely large workload that exceeds the target average workload per processor. Since this sample box cannot be divided further there is no way to distribute the load among different processors, and a few iterations will have to pass before a sufficient number of ligands diffuse out of the sample box and can be moved to another processor (Figure V.2.B).

IV.B Adaptive Load Balancing

Temporal variations in workload concentrations as described in Chapter III suggest that adaptive load balancing is required. A scheduler can constantly monitor the state of the system and be able to detect conditions that may indicate a need for immediate load balancing. Information can also be obtained from the initial state of the system to predict certain events that may require a load balance step in the future.

Workload Imbalance

The overall workload imbalance W_{im} can be estimated at each iteration by computing the fraction of the average workload by which the maximally loaded processor exceeds the average workload.

$$W_{im} = \frac{Maximum\ Load\ -\ Average\ Load}{Average\ Load} \tag{IV.1}$$

W_{im} can be monitored and a load balancing step is performed only if it crosses a predetermined threshold. This method requires global information about current workloads over all processors. Since this is a collective operation it will introduce synchronization delays if performed at every iteration timestep. This can partially be alleviated by checking for load imbalance only at regular intervals of time, with the assumption that the load distribution does not change drastically within the interval[1].

MDL input

In some simulations the MDL input provides information about the occurrence of certain events that can disrupt the workload balance in the system. If sudden ligand releases are to occur at regular intervals, it is usually possible to translate this information from the MDL input, into a set of specific iteration numbers at which the releases will occur. The releases can thus be predicted and the load balancer can be called with an artificial additional load, just before the release actually happens. This is particularly useful if the load imbalance is not being monitored at every iteration timestep. Simulation results with prediction are discussed in Chapter VI.

[1] In MCell-K, the sub iteration timesteps can be ignored with regard to monitoring load imbalance since the number of ligands diffusing between processors within one iteration is usually not large enough to cause a load imbalance.

Monitoring simulation variables

In some MCell simulations the operations that dominate the running time change as the iterations proceed. For example, as increasing numbers of ligands bind and saturate the binding effector sites, the fraction of the total running time spent on bound ligands may increase. The workload estimate can then be altered, and load balancing decisions can be made based on the new estimate. Also, if large ligand releases are triggered when certain simulation parameter values cross thresholds, the resulting workload imbalance can be predicted by monitoring the critical variables. A decision of whether to load balance or not, can depend on such monitored variables. There is no overhead in monitoring these parameters, since the MCell simulation itself requires it. If the workload on any processor is predicted to drastically exceed the average at any iteration timestep, the load balancer can be invoked to redistribute sample boxes among processors based on an artificial estimated load, before the load imbalance actually occurs.

Chapter V

MCell Parallel Simulator

The current version of MCell-K supports a regular domain decomposition with a one-to-one mapping between partitions and processors. It does not yet support overdecomposition and each processor maintains a single list of ligands randomized with respect to the ligands' location in the processors computational domain. Currently, load balancing algorithms that employ overdecomposition at the scale of the workload sample boxes cannot be implemented and tested with MCell-K[1].

In order to develop and test the performance model and load balancing algorithms described in Chapters III and IV, the serial MCell code was instrumented to simulate a parallel run. The parallel simulator was validated by demonstrating that it could produce spatial-temporal workload trends similar to those observed in actual MCell-K runs using a regular domain decomposition. It was then used to test the performance model and load balancing techniques. This chapter describes the design of this simulator, the assumptions it uses, and its validation.

[1] Work is currently in progress to implement an overdecomposition of the computational domain and support a many-to-one mapping between subdomains and processors. KeLP distributed data structures [4] are being used to overdecompose the computational domain. The overdecomposed sub-domains can be configured to be of the size of the sample boxes required for load balancing. This will facilitate the implementation of the dynamic load balancer described in Chapter IV, into MCell-K.

V.A Implementation

Sample Boxes

Since the performance model and the load balancer are based on statistics obtained at the scale of the sample boxes, the first step in constructing the simulator was to map all the SSVs in the computational domain to WSBs. The domain partitions whose tensor product formed the sample box partitions were aligned with SSV partitions to ensure that no SSV was only partially accommodated in any sample box. A direct linear mapping from the one dimensional SSV index carried in each ligand's data structure, to a one dimensional sample box index was used to assign ligands to sample boxes. This mapping is done dynamically at each iteration based on each ligand's current location. Counts and timings are accumulated into an array of structures with each element holding counter and timer statistics of one sample box[2].

Ligand list reordering

In order to simulate the processing of ligands per WSB the linked list of ligands is re-ordered to form sublists for each WSB. Each sublist contains all ligands located within the corresponding WSB and are arbitrarily ordered with respect to ligand location. These sublists are then concatenated in increasing order of WSB index. The linked list re-ordering is done using only pointer manipulation, leaving the locations of the ligand structures in the memory unchanged. Since a linear traversal through this list during the MCell computation sequence would be inefficient with regard to locality, the entire reordered list is copied into a contiguous chunk of memory at the beginning of each iteration.

[2]In a later version of MCell when the SSVs are to be represented on a tetrahedral mesh, an alternate mapping mechanism will be required to translate the SSV index into a corresponding sample box index that retains a cartesian grid structure for the sample boxes.

Timing and Communication

During the execution of the core MCell computation loop (Figure III.1) in which the entire linked list of ligands is linearly traversed, timers are placed around the processing of each sublist, to obtain total execution times per sample box. Timings required for the performance model (Chapter III) are thus accumulated per WSB. Given that the observed costs for individual ligand communication in MCell-K were negligible in comparison to the total running time [1], ligand communication between processors within and after each iteration, is ignored with respect to timing. Ligand communication is implicit in the ligand linked list reordering at the beginning of each iteration. Any ligands that have crossed sample box boundaries in the previous iteration will be mapped to their new sample boxes. The copying of the list into a contiguous chunk of memory emulates the storage of each processor's portion of the list on separate contiguous chunks of memory in a parallel run.

Load Balancing

Once all counter and timer statistics have been obtained per sample box, a mapping from sample box index to processor index is required to be able to compute the total workload and running time per processor. This mapping depends on the domain decomposition used. The load balancing algorithms are implemented as functions that take in the three dimensional grid of sample box counts and times and return a one dimensional mapping from sample box index to processor index. Statistics are then accumulated over all sample boxes per processor and used for further analysis.

A load imbalance estimator given by Equation IV.1 was used to compute W_{im}, the load imbalance over all processors at all or a specific number of iteration timesteps. This was used to make the load balancer adaptive by choosing to perform a load balancing step only if the computed load imbalance W_{im} crossed a predetermined threshold.

Load balancing via recursive co-ordinate bisection and a Hilbert space filling curve were implemented. The recursive co-ordinate bisection method overdecomposed the computational domain and used a many-to-one mapping between partitions and processors. Recursion was terminated by placing an upper limit on the allowed workload density per partition. Successive calls to this load balancer involved a re-computation over the entire domain decomposition. The decomposition based on the Hilbert space filling curve used the overdecomposition provided by the sample boxes and created a one-to-one mapping between partitions and processors. The partitions were created using an average workload based binning strategy that resulted in each processor holding a workload as near to the average workload as allowed by the granularity of the sample boxes. Successive calls to the space filling curve load balancer involved a one dimensional redistribution of sample boxes between neighbouring processors.

The initial domain decomposition was computed based on information specified in the MDL input. An artificial workload was generated across the grid of sample boxes based on the location of release sites and the number of ligands to be released from them in the first iteration.

Diagnostic Output

Counts and timings per processor for regular and adaptive domain decompositions, along with the number of sample boxes per processor were written to a log file at each iteration. The number of sample boxes moved between processors at each load balance step was monitored by comparing the old and new sample box to processor mappings. The workload estimates over all sample boxes were written to a file at specified iterations to be later loaded into ChomboVis[3], a three dimensional visualization application to monitor the spread of workload concentration over the entire computational domain.

[3] http://seesar.lbl.gov/anag/chombo

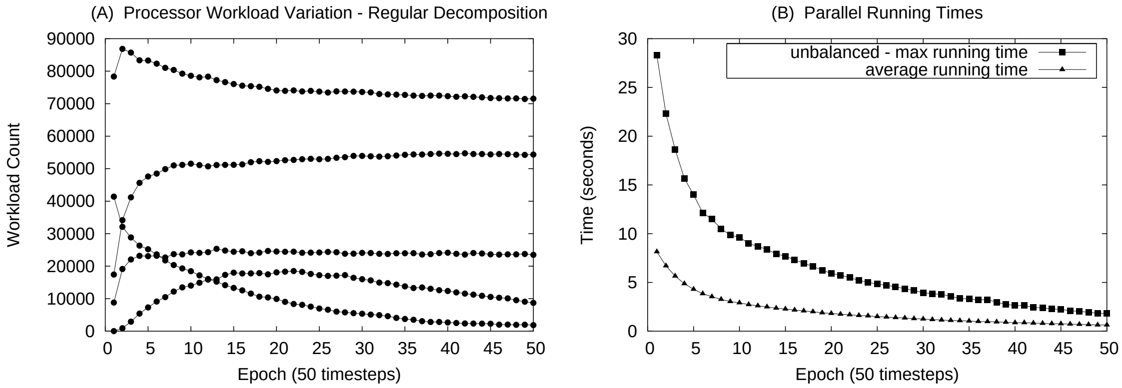

Figure V.1: Validation of the parallel simulator : (A) Workload count per processor for each of the most heavily loaded processors using a regular decomposition. (B)Maximum and average running times over all processors using a regular decomposition.

V.B Verification

Verification tests were done with a static regular decomposition similar to that used in MCell-K on the Chick Ciliary Ganglion simulation. The simulator was run on a 2.2GHz processor.

Test with a static regular decomposition

Figure V.1.A shows the load per processor for the most heavily loaded processors, as a function of iteration timestep for a simulated parallel run of the Chick Ciliary Ganglion simulation. These trends show that the initial workload distribution across processors rapidly changes as the ligands diffuse across processor boundaries resulting in a persistent high workload imbalance across processors. These trends are similar to those observed in the MCell-K runs described in Chapter II and Figure II.2.A in which a rapid movement of ligands between processors quickly resulted in a high load imbalance that persisted for a long time.

Figure V.1.B shows the maximum running time over all processors as a function of iteration timestep as compared to the average (ideal) running time per processor. The most heavily loaded processor ran three times slower than the

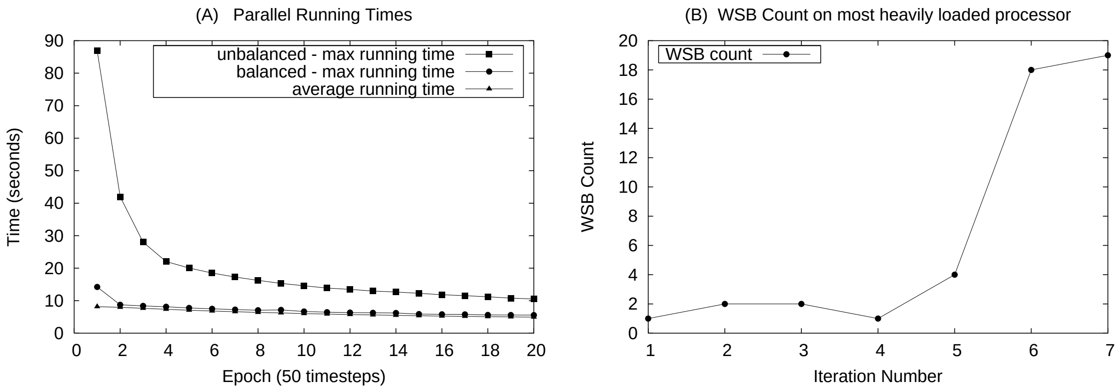

Figure V.2: Validation of the load balancer : (A) Maximum and average running times over all processors for unbounded diffusion with a regular decomposition and load balancing. (B)Number of WSBs on the most heavily loaded processor for the first 7 iterations of the unbounded diffusion simulation with load balancing.

average time per processor, indicating a persistent load imbalance. The parallel running time obtained was 375 seconds with the regular decomposition and 114 seconds as the average running time over all processors. This trend is similar to that observed in MCell-K (Figure II.2.B).

Load Balancer Test

The load balancing algorithms were then tested with the perfect instantaneous workload estimator, namely the actual measured running time per sample box per iteration. The MCell simulation used MDL input corresponding to a single release site located in the middle of the computational domain releasing 40,000 ligands that then undergo unbounded diffusion in free space.

The correctness of the load balancer can be judged via the plot in Figure V.2.A which shows the maximum running time over all processors with and without load balancing, along with the average (ideal) load. The running times for a run without load balancing were obtained based on a static regular domain decomposition. The running times obtained with load balancing are seen to closely track the target ideal curve. Since the actual measured running time per workload sample box (WSB) was used to estimate the workload distribution for the load balancer, this trend shows that the load balancer is capable of achieving a near ideal workload balance across processors. The total unbalanced running time was 402 seconds, the (ideal) average time was 124 seconds and the total balanced running time (without load balancing costs) was 142 seconds resulting in a computational speedup of 2.8.

The curve does not track the target ideal curve in the first few iterations. This is because the size of the sample boxes limits the granularity of workload redistribution. In the first few iterations most of the workload is concentrated within one or two sample boxes, and cannot be distributed across all available processors. Figure V.2.B shows the number of WSBs on the most heavily loaded processor during the first few iterations. The total workload on this processor greatly exceeded the average but there was no way to further divide and distribute it among other processors.

These tests show that in the case of a perfect instantaneous workload estimator, the load balancer was accurate and effective up to the granularity of the sample boxes. The accuracy of the performance model under various simulation conditions and the accuracy and efficiency of some adaptive load balancing techniques are discussed in Chapter VI.

Chapter VI

Simulations and Results

The performance models discussed in Chapter III and the load balancing algorithms discussed in Chapter IV were applied to some test runs of MCell. One test case was that of Simple Unbounded Diffusion and the other included periodic ligand releases that disrupt the load balance. The results were analyzed in terms of the accuracy of the performance models and the efficiency of the load balancing algorithms. A simulation run was then done on MDL input for a real data set corresponding to the Chick Ciliary Ganglion simulation and its results were analyzed to estimate the performance and accuracy of these methods in the general case of a real parallel MCell-K simulation.

VI.A Test Case : Simple Unbounded Diffusion

The Simple Unbounded Diffusion simulation consists of a single release site located in the middle of the computational domain and releasing 40000 ligands in the first iteration. The ligands are allowed to diffuse outward via a three dimensional Brownian-dynamics random walk. This is the simplest type of MCell simulation and has a spatial workload distribution that is initially concentrated in space and then gradually spreads out.

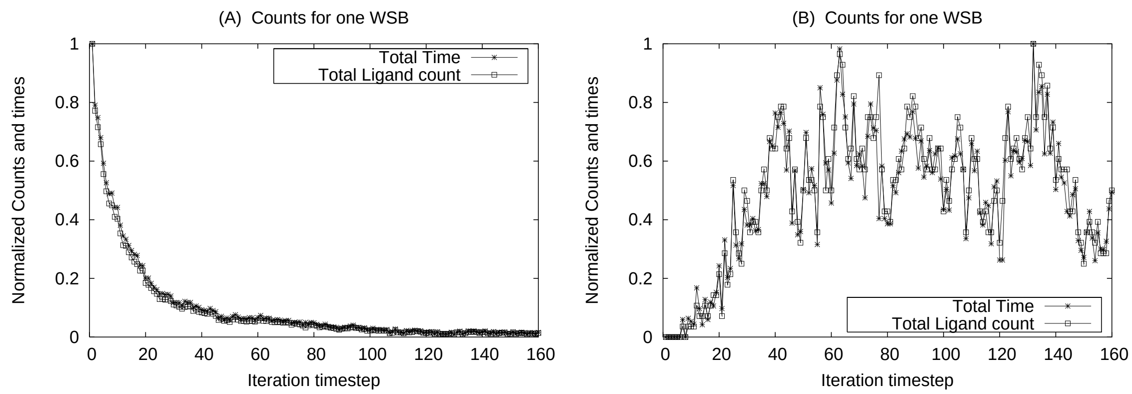

Figure VI.1: Running time and workload estimate for the unbounded diffusion simulation measured within (A) a WSB enclosing a release site and (B) an initially empty WSB that ligands diffuse into. The instantaneous performance model using the free ligand count to estimate the running time per WSB is seen to accurately estimate the running time.

VI.A.1 Performance model

Since there are no mesh walls, the instantaneous model estimates the running time from the number of free ligands. Figure VI.A.1 shows the measured running time and the workload estimate for two WSBs, one containing a release site and one that is initially empty. It can be seen that the free ligand count is an accurate estimator of the total running time of this simple run.

VI.A.2 Load Balancing

The performance model was also tested via load balancing by using the model to obtain workload estimates. A comparison was also made between the Recursive Co-ordinate Bisection (RCB) and Space Filling Curve (SFC) algorithms used for domain decomposition in the Unbounded Diffusion simulation. Runs were performed with load balancing performed at each iteration, and compared to runs with adaptive load balancing using W_{im}, the workload imbalance estimator. The parallel running times were obtained as the maximum running time over all

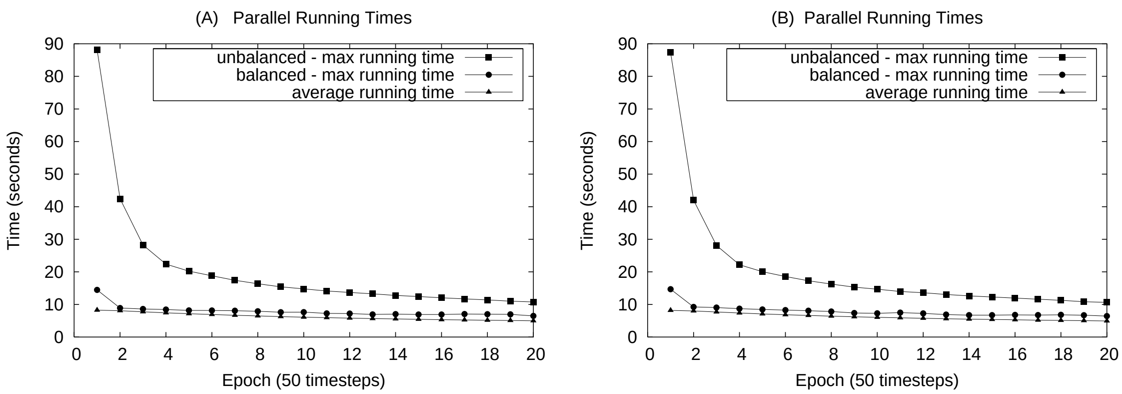

Figure VI.2: Unbounded diffusion : Maximum and average running times over all processors with a regular decomposition and with (A) SFC load balancing (B) SFC Adaptive load balancing using W_{im}.

processors. The unbalanced running time was based on a static regular domain decomposition. The balanced running time was based on dynamic and adaptive irregular domain decompositions. The ideal running time was computed as the global average running time over all processors and served as a reference to ascertain how accurate the load balancing was. The ideal parallel running time is however not always attainable given the granularity of the WSBs. A WSB holding more than the average workload of the system cannot be further partitioned to distribute its workload.

Computation

Figure VI.2.A shows that the balanced parallel running time closely tracks the ideal curve when load balancing was performed at every iteration timestep. The instantaneous model used here is thus a valid estimate of the parallel running time and leads to load balancing results similar to that obtained using the actual running time as the workload estimate (Figure V.2).

The accuracy of the load imbalance estimator was tested by making the load balancing adaptive. After each load balancing step, as the ligands diffused out and the load imbalance increased, the running time of the maximally loaded

	SFC continuous	SFC adaptive	RCB continuous	RCB adaptive
Unbalanced running time (sec)	407.45	403.71	401.69	403.99
Ideal running time (sec)	125.3	124.2	123.5	124.204
Balanced running time (sec) (Only Computation)	157.78	157.59	154.67	155.52
Computational Speedup	2.5824	2.5618	2.5971	2.5977
Ratio of Balanced to Ideal running time	1.2592	1.2688	1.2524	1.2522
Number of Load Balancing Steps	1000	158	1000	29
Time spent per load balance step (sec)	0.062	0.061	0.127	0.104
Average number of WSBs to be communicated per load balance step	51.074	292.68	84.458	2105.9

Table VI.1: Unbounded Diffusion - Parallel Running times and speedups on 16 processors : The Space Filling Curve (SFC) and Recursive Co-ordinate Bisection (RCB) domain decompositions resulted in similar ratios of total balanced to ideal running time. Adaptive load balancing using W_{im} required a small number of load balancing steps to achieve computational speedups and balanced to ideal running time ratios comparable to the runs with load balancing performed at every iteration. RCB based load redistribution was more expensive in terms of computation and communication.

processor began to increase. The load balancer was called whenever the workload imbalance estimator W_{im} (given by equation IV.1) crossed 1.2. This allows a workload imbalance of up to 20% of the average load to be tolerated. Figure VI.2.B shows that the balanced running time as a result of adaptive load balancing also closely tracks the ideal curve. Table VI.1 compares total parallel running times for simulation runs with and without load balancing using both RCB and SFC based algorithms[1].

[1]The timings were obtained from separate simulation runs. The difference of 3-6 seconds in the total unbalanced running time is due to noise in the timings arising from external effects like memory and disk accesses.

The following observations can be made.

1. The Space Filling Curve (SFC) and Recursive Co-ordinate Bisection (RCB) domain decompositions resulted in similar ratios of total balanced to ideal running time showing that they were both able to achieve a comparable workload balance.

2. Adaptive load balancing using W_{im} required a small number of load balancing steps to achieve computational speedups and balanced to ideal running time ratios comparable to the runs with load balancing performed at every iteration.

3. RCB load balancing per iteration was more expensive than load balancing based on the SFC algorithm. The dynamic RCB domain decomposition however was more stable and required fewer load balancing steps than the SFC.

4. The RCB decomposition required much more communication than the SFC, even in the dynamic case where the RCB performed fewer load balancing steps. This could be improved by making successive RCB decompositions incremental. Also, the average number of WSBs communicated per load balancing step was larger for the dynamic load balancing run. This is because in the dynamic load balancing runs a larger imbalance has to be corrected for at each step.

Communication

During each load balancing step communication between processors is carried out in terms of WSBs. Communication counts for the Unbounded Diffusion simulation using SFC and RCB domain decompositions were obtained. Figure VI.3 shows the communication counts for the first 160 iterations with load balancing performed at every iteration and with adaptive load balancing. The entire simulation ran for 1000 iteration timesteps, but in both cases, all significant WSB

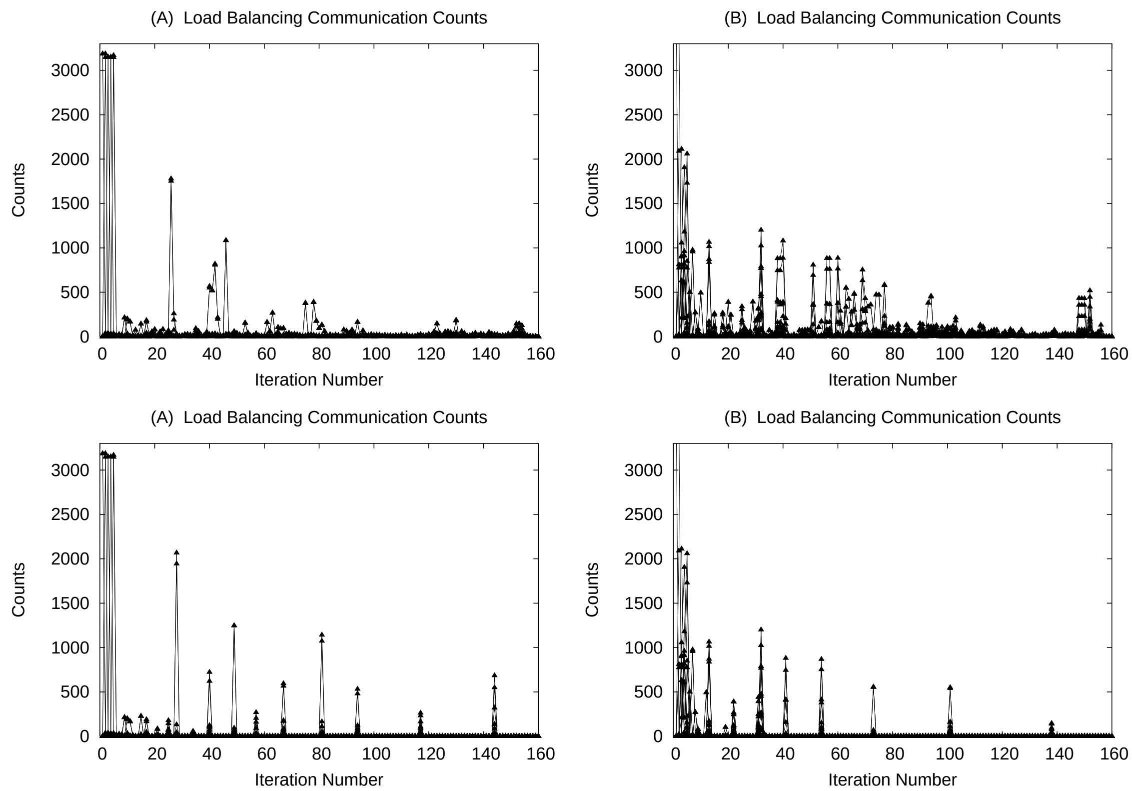

Figure VI.3: Load Balancing Communication Counts - (top) Unbounded Diffusion with load balancing at every iteration: (A) SFC, (B) RCB and (bottom) Unbounded Diffusion with Adaptive load balancing:(A) SFC, (B) RCB.

communication during load balancing occurred within the first 160 iterations. Also the maximum number of WSBs are communicated during the first few load balancing steps.

This demonstrates that the workload is initially very irregularly distributed and is highly dynamic in nature during the first 100 iterations after which it is more naturally balanced and WSB communication becomes negligible. This shows that load balancing communication costs could be large during the first few hundred iteration timesteps during which load balancing is not only required more frequently, but the number of WSBs and ligand lists to be communicated per load balance step is much more than during the later iterations.

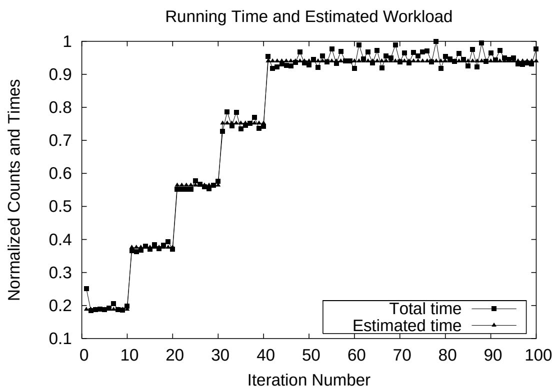

Figure VI.4: Unbounded Diffusion with Periodic Ligand Releases: (A)Total measured running time and estimated running time using the instantaneous model.

Use of MDL input information

Information about periodic ligand releases obtained from the MDL input can be used to predict when a load balancing step may be required, by using an artificial workload estimate. The unbounded diffusion simulation was run with periodic ligand releases (Figure VI.4) and an artificial workload was used to perform a load balancing step before each release. A small test run included 100 iterations with two release sites, each releasing 1000 ligands every 10 iteration timesteps up to iteration 40. The periodic releases result in the total serial running time profile of Figure VI.4. Since there are no mesh walls, the free ligand count is an accurate estimator of the total running time.

Figure VI.5 shows the maximum and average running times obtained for a run with dynamic load balancing with and without prediction. Without prediction, the instantaneous model can detect the releases only after they occur, and there are periodic instances of high load imbalance at iterations 10,20,30 and 40 (almost equal to that of the unbalanced case) corresponding to ligand releases. With prediction of the ligand releases, the balanced running times for iterations 10,20 and 30 reduce to half the balanced running time without prediction at those iterations. The granularity of the WSBs prevented a better workload balance at

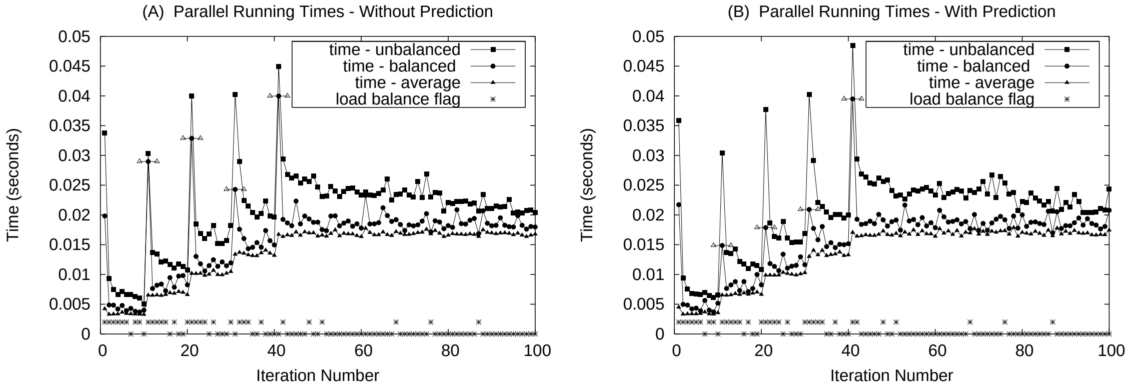

Figure VI.5: Unbounded Diffusion with Periodic Ligand Releases - Adaptive Load Balancing: (A) Maximum and average running times over all processors without prediction. The balanced running time has high peaks at iterations 10,20,30 and 40 that correspond to ligand releases that result in a high load imbalance for one iteration. (B) Maximum and average running times over all processors with prediction. The balanced running time at iterations 10,20 and 30 reduce to half the balanced running time of (A) at those iterations.

those iterations. Without prediction, load balancing was performed 31 times and with prediction it was performed 33 times. The total running time reduced and the ratio of the total balanced running time to the ideal running time decreased from 1.2012 to 1.1732 with prediction, showing that a better overall load balance was achieved.

Based on this simple run, it can be seen that prediction of ligand releases can reduce the parallel running time. The effect of prediction will be more pronounced if the load balancing is not limited by the granularity of the WSB grid. Periodic releases described in the MDL input can be predicted by computing the iteration numbers at which the releases are to occur. Releases that are triggered by certain simulation variables can be predicted by monitoring the values of the concerned variables. Ligand releases can also be designed to occur randomly, in which case, prediction is almost always not possible and one iteration with a high load imbalance must pass before being detected by the instantaneous model.

VI.B Test Case : Chick Ciliary Ganglion

The Chick Ciliary Ganglion MCell simulation was used as the main test case in this analysis. The input to this simulation is a realistic structure reconstructed using serial electron microscope tomography of a chick ciliary ganglion. It consists of a complex set of mesh surfaces with several effector tiles and ligand binding sites. Ligands released in the first iteration diffuse outward and can react with mesh walls and change state, bind with effector sites, or decay. The simulations were run for 2500 iterations on a 2.2GHz processor. There were 32 release sites located near mesh walls, each releasing 1000 ligands during the first iteration. A complete run would involve over 500 release sites, each releasing 5000 ligands and would be repeated several times to ensure statistical accuracy of the results. The simulations and results discussed in this section are based on one simulation run. Repetitions of the runs gave almost identical results. The space filling curve and recursive co-ordinate bisection based dynamic load balancers were both tested in these runs.

VI.B.1 Performance Model

During the simulation, ligands released from sites located near mesh surfaces diffused out and began to interact with the mesh walls almost immediately. Ligands began to bind to the effector tiles and the number of free ligands dropped rapidly.

Parameters of the instantaneous model given by Equations III.1 and IV.1 were used to construct an estimate of the running time. Figure VI.6 shows the running time profile over the entire computational domain along with the computed workload estimate, scaled to the same magnitude as the observed running time. It is seen that this workload estimate gives an accurate estimate of the total running time, using spatially varying operation counts.

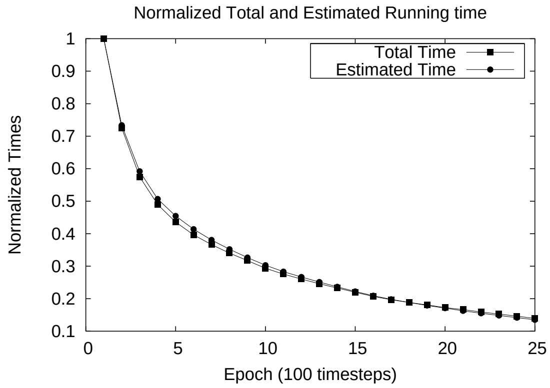

Figure VI.6: Normalized running time profiles and workload estimates for the chick ciliary ganglion simulation with release sites located near mesh walls.

VI.B.2 Load Balancing

The performance model was also tested via load balancing simulations that used the model to obtain workload estimates for the Chick Ciliary Ganglion simulation. A comparison was made between RCB and SFC domain decomposition algorithms on 16 processors. Comparisons were made between runs with load balancing at each iteration, adaptive load balancing based on W_{im} the workload imbalance estimator, and load balancing performed at every 10 iterations.

Computation

The parallel running time was obtained as before, as the maximum observed running time over all processors per iteration. The ideal running time was obtained as the average of the running times over all processors at each iteration. The unbalanced parallel running time using a static decomposition, along with the balanced and average (ideal) parallel running times observed with SFC and RCB based load balancing performed at every iteration are shown in Figure VI.7. Parallel running times using SFC and RCB load balancing at every iteration, adaptive load balancing using W_{im} and load balancing at every 10 iterations are summarized in Table VI.2.

	SFC conti- -nuous	SFC adaptive	SFC regular	RCB conti- -nuous	RCB adaptive	RCB regular
Unbalanced running time (sec)	335.78	336.71	381.24	335.27	335.7	381.4
Ideal running time (sec)	99.32	99.6	111.97	99.12	99.39	111.8
Balanced running time (sec) (Only Computation)	141.47	141.21	164.32	150.6	151.03	176.67
Ratio of Balanced time to Ideal time	1.424	1.41	1.46	1.52	1.51	1.57
Computational Speedup	2.37	2.38	2.32	2.23	2.22	2.16
Number of Load Balancing Steps	2500	2443	250	2500	2454	250
Time spent per load balancing step (sec)	0.062	0.061	0.062	0.102	0.102	0.101
Average number of WSBs communicated per load balancing step	20.7	15.310	75.42	69.4	51.235	127.59

Table VI.2: Parallel running time and costs for the chick ciliary ganglion test case using SFC and RCB based load balancing. Adaptive SFC and RCB based load balancing were done by performing load balancing steps whenever W_{im} crossed 0.3. Regular load balancing was done by performing load balancing every 10 iterations. The SFC domain decomposition resulted in a more balanced workload and hence better speedups than the RCB. Load balancing at every 10 iterations achieved a slightly poorer speedup, but given the number of load balancing steps performed, would incur a lower total load balancing cost compared to the other methods. The SFC based load balancer incurred less computational and communication costs.

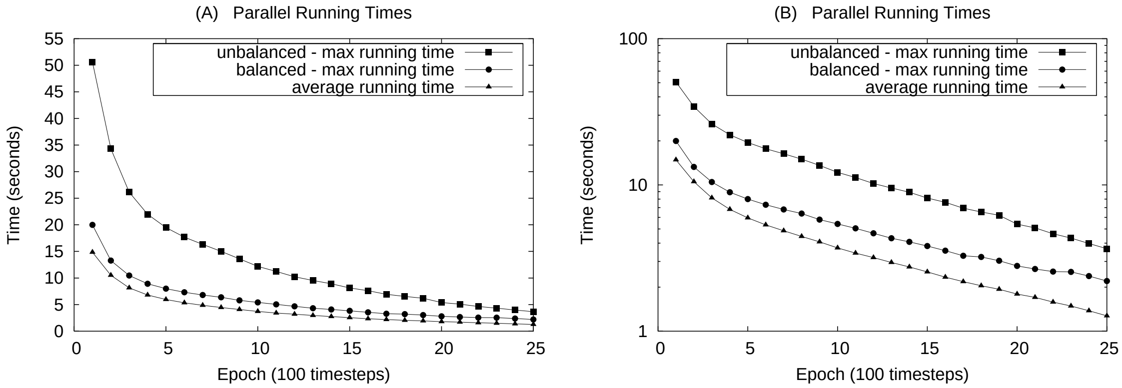

Figure VI.7: Maximum and average running times with a static regular decomposition and with SFC based load balancing at each iteration for the ciliary ganglion simulation with release sites located near mesh walls. A computational speedup of 2.37 was obtained. (A) linear scale, (B) logarithmic scale.

The following observations can be made.

1. Figure VI.7.A shows that load balancing was most effective during the first few hundred iterations during which the regular decomposition resulted in a very high load imbalance. With a logarithmic scale (Figure VI.7.B) it was seen that the accuracy of the performance model degraded slightly after the first few hundred iterations, but this did not adversely affect the running time during the later iterations which were inherently less compute intensive than the initial iterations.

2. Table VI.2 shows that the SFC domain decomposition resulted in a better computational speedup than the RCB domain decomposition. The runs with the SFC domain decomposition gave lower ratios of the balanced to ideal running times, showing that the workload distribution obtained using the SFC algorithm was better balanced than the load distribution obtained with the RCB algorithm. This can be explained by the fact that the SFC decomposition is more fine grained and because it allows irregularly shaped partitions per processor.

3. Runs with load balancing at every iteration were compared to runs with dynamic load balancing where load balancing steps were performed based on a W_{im} threshold. The tight threshold of 1.3 on W_{im} did not result in any benefit since the load balancer was invoked at over 2400 out of 2500 iterations. The runs with load balancing performed every iteration incurred lower total load balancing costs, but showed a slight degradation in load balancing accuracy and computational speedup. There were fewer load balancing steps, but the number of WSBs to be communicated per load balance step was higher.

4. The time spent per iteration on computing the new workload distribution across processors was an order of magnitude higher for the RCB as compared to the SFC. Communication costs were also observed to be higher for the RCB decomposition.

It can therefore be seen that the Space Filling Curve based irregular domain decomposition is a more efficient choice for dynamic load balancing. The SFC based load redistribution is more efficient than the RCB with regard to both computation and communication costs. The decomposition is more fine grained and results in a more balanced workload at the end of each load balance step. Higher computational speedups are observed as a consequence.

Communication

Load balancing communication counts in terms of the number of WSBs to be communicated between processors for the chick ciliary ganglion simulation using SFC load balancing are shown in Figure VI.8. The number of WSBs to be communicated in the run with load balancing performed at every iteration is 66143 as compared to 56203 for the dynamic load balancing case. Also, the simulation ran for 2500 iteration timesteps but all significant load balancing communication was required only within the first 200 iterations.

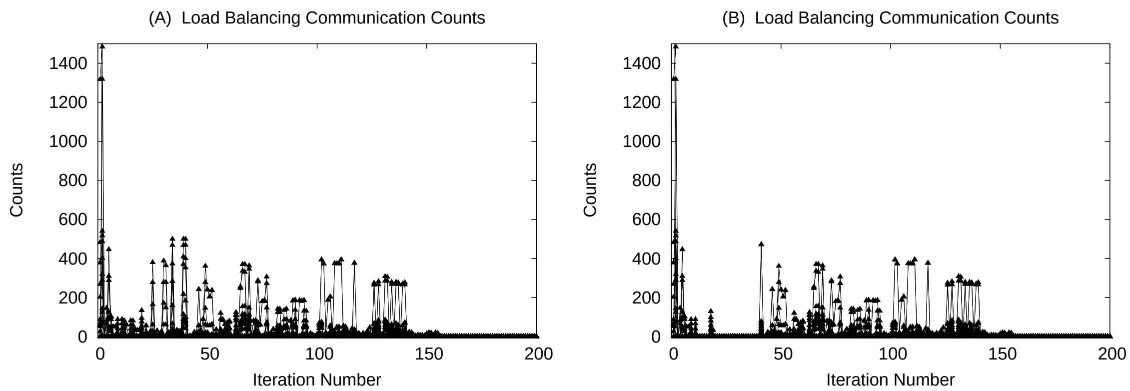

Figure VI.8: Load Balancing Communication Counts for the first 200 iterations: (A) SFC load balancing at all iterations (B) SFC Adaptive load balancing.

These observations suggest a trade-off between the computational versus communication costs associated with load balancing while deciding when a load balance step is to be performed. Runs with frequent load balancing incur more computational costs but involve less communication. Runs with load balancing performed every 10 iterations had considerably less computational overheads but significantly larger communication costs. Also, most load balancing communication is required during the first few hundred iterations, which already comprises a large fraction of the total computation and this may limit the speedup.

Use of MDL Input

Figure VI.9.A shows the workload distribution at the 20^{th} iteration. It can be seen that the workload is concentrated and irregularly distributed during the first few iteration timesteps. Figure VI.9.B depicts a cross section of the workload distribution at the 20^{th} iteration that shows the varying workload density profile within regions of high workload. The MDL description of the release site locations and the number of ligands to be released from each release site during the first iteration, was used to generate the initial domain decomposition. This optimization resulted in a decrease in the time spent during the first few iterations as compared to a run that began with a regular decomposition.

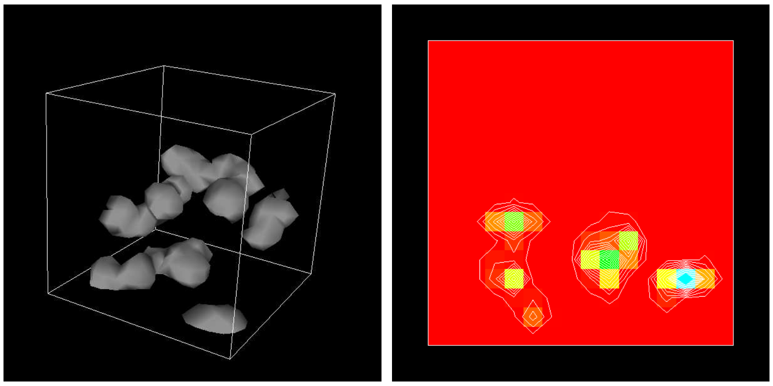

Figure VI.9: (A)Three dimensional plot of the workload distribution at the end of the first 20 iterations (B) A Cross Section of the workload to display the irregular density distribution.

Parallel Speedups

Parallel running times were obtained using a regular domain decomposition and SFC based load balancing on the chick ciliary ganglion simulation simulated for 8,16,32 and 64 processors. Figure VI.10 shows the balanced parallel running times per iteration for simulations of 8,16,32 and 64 processors. The observed speedups with the parallel running times obtained with regular decompositions can be compared to the ideal speedup and that obtained after load balancing. The running time on 8 processors was taken as the base case for speedup calculations. Figure VI.10.B shows speedup plots for the ideal, unbalanced and balanced cases and Table VI.3 summarizes the observed timings.

The following conclusions can be made.

1. The balanced simulation scales better than the unbalanced case, but does not scale ideally. The precision of the performance model becomes more important as the number of processors is increased (note that load balancing communication was not timed and the speedups have been computed based only on the computational load per processor attributed to MCell computations.)

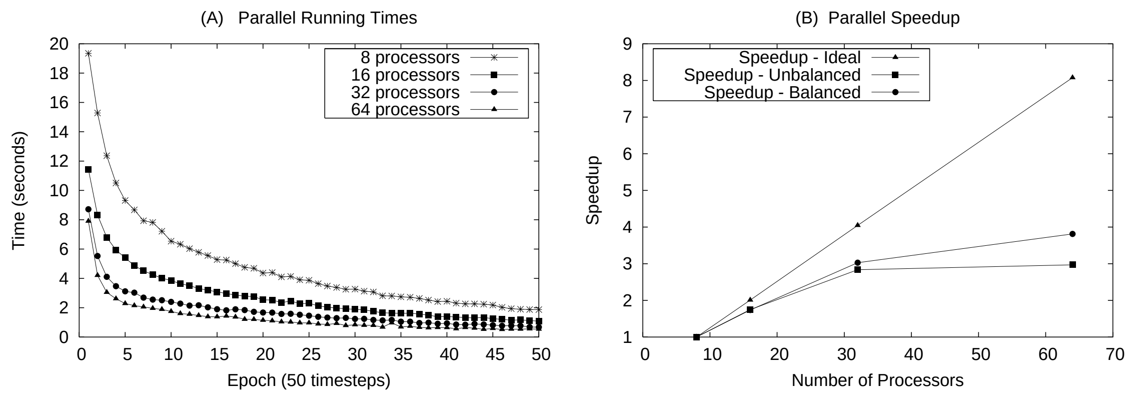

Figure VI.10: (A)Load Balanced Parallel Running times on 8,16,32,64 processors (B) Speedup Curves with unbalanced, balanced and ideal running times. Speedups are relative to the running time on 8 processors.

	8 procs	16 procs	32 procs	64 procs
Unbalanced Running Time	582.84	333.35	205.55	196.29
Speedup - Unbalanced	1.00	1.748	2.83	2.96
Ideal Running Time	198.32	98.78	49.02	24.56
Speedup - Ideal	1.00	2.01	4.04	8.07
Balanced Running Time (Only Computation)	243.61	140.24	80.47	63.88
Computational Speedup	1.00	1.73	3.02	3.81
Balanced/Ideal	1.2284	1.4197	1.6416	2.6010

Table VI.3: Running times on 8,16,32 and 64 simulated processors. The 8 processor timings are taken as the base case for computing parallel speedups

2. One factor that may limit the speedup on a large number of processors is overdecomposition becoming an overhead during the latter half of the simulation when the workload is rarified.

3. If the cost for load balancing is also factored in, as the number of processors increases the load balancing communication becomes a global overhead and the problem will not scale as well. This shows that efficient and scalable load balancing operations will be required.

Chapter VII

Conclusion

This thesis involved a performance analysis of the MCell Monte-Carlo Cellular Microphysiology simulator and the design and evaluation of an accompanying performance model for MCell-K, a parallel implementation of MCell. The model was validated by applying it to the problem of dynamic load balancing in a parallel environment. A parallel simulator was implemented and used to test the performance models on dynamic load balancing algorithms based on adaptive irregular domain decomposition techniques.

Two components of the hybrid performance model were explored. An instantaneous component that estimated the running time for an iteration based on operation counts gathered during the previous iteration was derived from the running time profile of all spatially varying computations. The instantaneous component can also be obtained from actual running time measurements during the previous iteration but will be subject to timer overheads and inaccuracies due to limited timer resolution and noisy running time variations. A predictive component was added to use prior information about the state of the system and the mechanisms that drive it, to predict the running time of a future iteration. A hybrid performance model that combined the instantaneous and predictive components was used to estimate the dynamic spatial workload distribution in a typical MCell-K run. An adaptive load balancer that used this workload estimate was able to dynamically detect a load imbalance if it occured and trigger the redistribution of workload among processors. The predictive component of the hybrid model was effective at iterations where an accurate estimate could not be obtained from retrospective information alone. The use of the hybrid performance model along with dynamic load balancing algorithms resulted in a factor of two performance improvement in simulated parallel runs on 16 processors. The resulting computation was also found to be scalable.

Load balancing algorithms explored were based on two adaptive domain decomposition techniques, recursive co-ordinate bisection and a one dimensional partitioning along a 3D Hilbert space filling curve. A few comparative performance tests were performed to evaluate the two techniques in terms of accuracy, stability and computational and communication costs. The load balancing tests revealed that the space filling curve algorithm resulted in a more balanced workload distribution and incurred less load balancing computation and communication costs.

Tests were performed on simulations of unbounded diffusion and periodic ligand releases as well as a realistic sample MCell simulation with the complete mesh microstructure and ligand diffusion and reactive properties of a chick ciliary ganglion. These tests demonstrated the accuracy of the performance model and load balancing techniques on the problems tested. The results showed that it is possible to derive an accurate performance model for an MCell-K run. Although a generic model may not be optimal for all types of simulations, custom performance models can be constructed using information from the input model description.

MCell-K is currently being modified to handle an over-decomposition of the computational domain and support a many-to-one mapping between sub-domains and processors. The sub-domains that are created can be configured to be the size of the workload sample boxes used to obtain the workload estimate in the performance model. This will facilitate the implementation of a dynamic load balancer into the framework of MCell-K.

Bibliography

[1] G.T. Balls., S.B. Baden. Kispersky, Tilman, Thomas M. Bartol, Terrence J. Sejnowski. *A large scale Monte Carlo simulator for cellular microphysiology*. Proceedings of the 18th International Parallel and Distributed Processing Symposium. April 1982

[2] T.M. Bartol, J.M. Stiles. *Monte Carlo Methods for Simulating Realistic Synaptic Microphysiology using MCell*. E. DeSchutter,editor, Computational Neuroscience: Realistic Modeling for Experimentalists,CRC Press 2001.

[3] J.R.Pilkington, S.B. Baden. *Dynamic Partitioning of Non-Uniform Structured Workloads with Space Filling Curves*. IEEE Transactions on Parallel and Distributed Systems, 7(3):288–300, 1996.

[4] S.B. Baden, P.Collela, D. Shalit, B.V.Straalen. *Abstract KeLP*. Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, Virginia, March 2001.

[5] S.B.Baden, S.R.Kohn. *A comparison of load balancing strategies for particle methods running on mimd multiprocessors.*. In Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing, March 1991.

[6] F.D. Sacerdoti. *A Cache-Friendly Liquid Load Balancer*. M.S. Dissertation, June 2002.

[7] S.B. Baden. *Programming Abstractions for Dynamically Partitioning and Coordinating Localized Scientific Calculations Running on Multiprocessors* . SIAM J. on Scientific and Statistical Computing, (pp. 145-157) January 1991.

[8] L.V. Kale *The Virtualization Approach to Parallel Programming: Runtime Optimizations and the State of the Art*. LASCI, October 2002.

[9] J.E. Flaherty et al. *Parallel Structures and Dynamic Load Balancing for Adaptive Finite Element Computation*. Applied Numerical Mathematics, Volume 26, pp 241-263, January 1998.

[10] H. Casanova, T.M. Bartol, J.R. Stiles, F.Berman. *Distributing MCell simulations on the grid*. International Journal of High Performance Computing Applications, 15:243-257, 2001.

[11] J. C. Phillips et al. *NAMD:Biomolecular Simulation on Thousands of Processors*. IEEE 2002

[12] C. Xu, Francis C.M.Lau. *Load Balancing in Parallel Computers - Theory and Practice*. Kluwer Academic Publishers.

[13] M.J.Berger, S.H.Bokhari. *A partitioning strategy for non-uniform problems on multiprocessors*. IEEE Transactions on Computers, vol C36,pp 570-580,May 1987.

[14] J.P.Singh, J.Holt, J.L Hennessy, A.Gupta. *A parallel adaptive fast multipole method*. Supercomputing, pp 54-65, 1993.

[15] Office of Graduate Studies and Research. *Instructions For Preparation and Submission of Doctoral Dissertations and Masters' Theses*. University of California, San Diego. 1991.