A large scale Monte Carlo simulator for cellular microphysiology

Gregory T. Balls and Scott B. Baden
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive
La Jolla, CA 92093-0114 USA
gballs@cs.ucsd.edu, baden@cs.ucsd.edu

Tilman Kispersky, Thomas M. Bartol, and Terrence J. Sejnowski
The Salk Institute
10010 North Torrey Pines Road
La Jolla, CA USA
tilman@salk.edu, bartol@salk.edu, terry@salk.edu

Abstract

Biological structures are extremely complex at the cellular level. The MCell project has been highly successful in simulating the microphysiology of systems of modest size, but many larger problems require too much storage and computation time to be simulated on a single workstation. MCell-K, a new parallel variant of MCell, has been implemented using the KeLP framework and is running on NPACI's Blue Horizon. MCell-K not only produces validated results consistent with the serial version of MCell but does so with unprecedented scalability. We have thus found a level of description and a way to simulate cellular systems that can approach the complexity of nature on its own terms. At the heart of MCell is a 3D random walk that models diffusion using a Monte Carlo method. We discuss two challenging issues that arose in parallelizing the diffusion process – detecting time-step termination efficiently and performing parallel diffusion of particles in a biophysically accurate way. We explore the scalability limits of the present parallel algorithm and discuss ways to improve upon these limits.

1 Introduction

The computational challenge of biological modeling stems largely from the wide range of space- and time-scales encompassed by molecular and cellular processes. To date, a variety of theoretical and computational methods have been developed independently for different problems at different scales. At the atomic/molecular level, quantum and molecular mechanics (QM/MM) simulations require femtosecond time resolution and massively parallel computation, and thus generally cannot be used at spatial and temporal scales beyond small or partial protein structures and nanosecond time frames. Cellular/multicellular studies, on the other hand, have mostly focused on higher-level physiological processes, e.g. biochemical signaling, that occur on much longer timescales (milliseconds to days), and for simplicity and computational efficiency have mostly used methods based on systems of ordinary differential equations. With this approach, cell structure and spatial features are limited or lacking, as are stochastic effects, which in-vivo may contribute to the robust nature of the organism and may also account for switching into disease states.

Much of functional cell physiology lies between these two spatio-temporal extremes, i.e. at the microphysiological level, where finite groups of molecules, subject to complex structural and spatial arrangements, are coupled via stochastic and/or directed interactions that drive cellular biochemistry and machinery. A major challenge is to develop modeling and simulation methods that allow integration of mechanisms, kinetics, and stochastic behaviors at the molecular level with structural organization and function at the cellular level.

A number of non-Monte Carlo simulators can be used to study diffusion and computational biochemistry in cells (e.g. FIDAP [8], Kaskade [4], Virtual Cell [21], E-Cell [29]), but are not generally suitable for large-scale, high-resolution three-dimensional simulation of realistic ultrastructure. Furthermore, these simulators, which are based on finite-difference or finite-element methods, do not re-

Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

alistically simulate cases where the number of active or diffusing molecules within a subcellular compartment is very small—which is often the case. For example, while Smart and McCammon [22] present a simulation of a simplified, planar, vertebrate neuromuscular junction (NMJ) using Kaskade, it would be exceedingly difficult to use such a finite-element simulator to study realistically reconstructed synaptic architecture (see Fig. 1A) where symmetries in the reaction space do not exist and cannot be simplified. In addition, while programs like NEURON [14, 15] or GENESIS [6] may be used to simulate some aspects of the electrophysiological properties of cells, these tools were not designed to simulate the chemical properties of cells and do not explicitly consider three-dimensional properties of intra- and extracellular microdomains. Thus, they cannot model local ion depletion or accumulation in regions of densely packed ion channels. Finally, the numerical methods used in these non-Monte Carlo simulators do not cope well with fine structure and complex three-dimensional boundary/initial conditions, and would consume vast computational resources in the effort.

The need for spatially realistic models in simulations of cellular microphysiology motivated development of a general Monte Carlo simulator of cellular microphysiology called MCell originally in the context of synaptic transmission [1, 27, 24, 26, 25]. In this setting, Monte Carlo (MC) methods are superior to the methods discussed above and are uniquely able to simulate realistic biological variability and stochastic "noise". The desire to simulate larger systems in shorter times has motivated a parallel version of MCell. We give a brief overview of MCell here and report on algorithms, run-time support, and software infrastructure for parallel, scalable, realistic cellular modeling based on MCell and KeLP.

2 Simulating Cellular Microphysiology

2.1 Overview of MCell

MCell uses rigorously validated MC algorithms [3, 24, 23, 25] to track the evolution of biochemical events in space and time for individual molecules called *ligands* and *effectors*. Ligands move according to a 3D Brownian-dynamics random walk and encounter cell membrane boundaries and effector molecules as they diffuse. Encounters may result in chemical reactions governed by user specified reaction mechanisms; as described below, molecular transitions are chosen using a random selection process using probabilities derived from bulk solution rate constants [23]. The diffusion algorithms are completely grid-free (i.e. no spatial lattice) and the reaction algorithms are at the level of interactions between individual molecules and thus do not involve solving systems of differential equations in space and time. Details about individual molecular structures are ignored, which distinguishes MCell from quantum and molecular mechanics (QM/MM) simulations. Since time-steps are generally on the scale of microseconds, simulations are feasible on the scale of milliseconds to seconds in duration. Importantly, MCell simulations explicitly include the functional impact of individual stochastic state transitions, molecular positions, and density fluctuations within realistic subcellular geometries.

MCell is very general because it includes a high-level model description language (MDL) which allows the user to build subcellular structures and signaling pathways of virtually any configuration [23]. Membrane boundaries are represented as triangle meshes and may be of arbitrary complexity (Fig. 1A). The defined structure and diffusion space is populated with molecules that interact probabilistically (Fig. 1B, [25]).

2.2 Chemical Reaction Algorithms

MCell simulations can contain hundreds to thousands of surface mesh objects, meshes composed of millions of polygons, and millions of molecules. During each time-step, the current list of molecules must be traversed and the range of possible downstream events depends on each molecule's initial state. The chemical reaction algorithms encompass both space-independent *unimolecular transitions* and space-dependent *bimolecular transitions.* Unimolecular events are first order processes, in which a given molecule simply changes state or gives rise to one or more diffusing molecules, e.g. a conformational change, ligand unbinding, or transmembrane ion flux.

MCell employs unimolecular transition methods similar to the earlier Gillespie MC method [12, 13, 17]. The underlying computations for these transitions are relatively simple, inexpensive, and are readily parallelized. In contrast, MCell's unique algorithms for Brownian-dynamics random walk and bimolecular events account for the bulk of computational cost and are considerably more difficult to parallelize, as will be discussed below. Thus, the Monte Carlo algorithm used by MCell cannot be treated as if it were embarrassingly parallel.

The most common bimolecular event is *binding,* which may occur whenever diffusing (dimensionless) molecules encounter appropriate other molecules (e.g. receptor proteins) on a surface. Each such receptor is actually represented by a discrete "tile" (Fig. 1B). During every time-step each diffusing molecule must be traced along a random walk trajectory (i.e. a ray) to find potential intersections with mesh elements of surfaces (Fig. 1A). If no intersection occurs, the molecule is simply placed at the endpoint of the ray and remains there for the duration of the time-step. If intersection does occur, the final result depends on the pres-

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

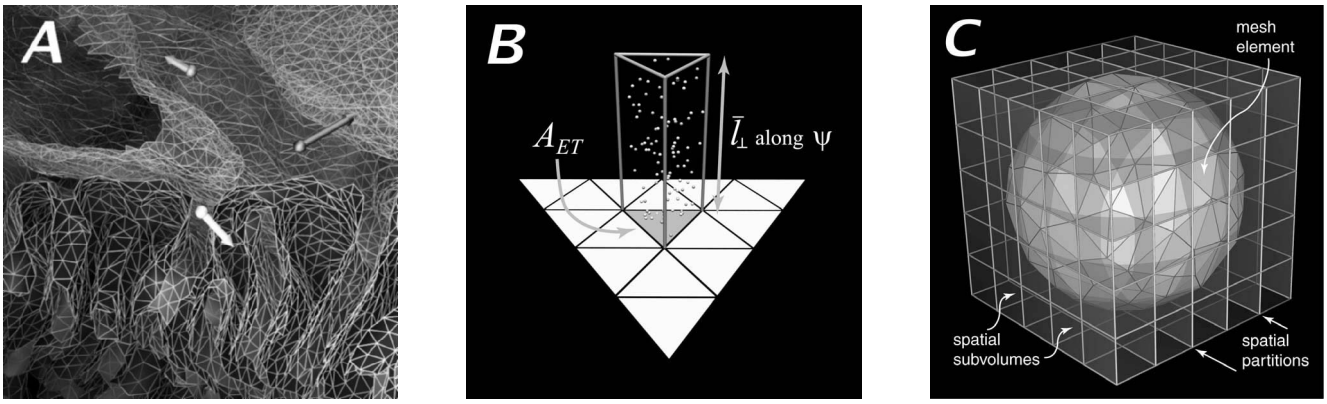

Figure 1. A) Raytracing diffusion paths within a 3D reconstruction of nerve-muscle membrane represented as a triangle mesh. Model by J. Stiles and T. Bartol, with assistance from P. Davidson. DReAMM visualization by J. Stiles. Serial electron micrographic sections for the reconstruction by T. Deerinck in the laboratory of M. Ellisman (UCSD). B) Mapping of triangular effector tiles (A_{ET}) onto a single triangle mesh element from A. Extruded volume above A_{ET} represents the domain from which diffusing molecules (shown as dots) may collide with and bind to an effector. C) Example of spatial subvolumes. A spherical polygon mesh surface is shown with spatial partitions (transparent planes) along the X, Y, and Z axes. Spatial subvolumes (cuboids) are created between the partitions. Under optimal conditions each subvolume includes (wholly or partly) no more than a small number of mesh elements. The search for collisions between diffusing ligand molecules and mesh elements in A and B (ray tracing and marching) can then be restricted to individual subvolumes, dramatically increasing execution speed. Partitions may be placed anywhere along each axis.

ence or absence of a reactive tile at the point of intersection, and/or the properties of the mesh element (Fig. 1B). If the molecule is neither absorbed by the surface nor retained at the point of intersection because of binding, then its movement must be continued. After passing through or reflecting from the mesh element, the search for an intersection begins again, and this process of ray marching continues until the molecule is absorbed, binds, or travels the total original diffusion distance.

2.3 Optimization and Scaling of Random Walk

To accelerate the search for intersections between mesh elements and molecules, MCell employs a geographical data structure to place the particles into bins, analogous to the chaining mesh structure described by Hockney and Eastwood [16]. The binning structure subdivides the computational volume into spatial subvolumes called *SSVs*, as shown in Fig. 1C. SSVs are constructed by forming a tensor product of 1-D decompositions placed at irregular positions along the X, Y, and/or Z axes, resulting in a rectilinear partitioning [18].[1]

The partitioning is adjusted so that each SSV contains no more than a few mesh elements and the the search for intersections during ray marching can be limited to the current and neighboring SSVs. Execution time scales as O(N), where N is the number of diffusing molecules. Whenever a molecule passes through an SSV boundary, ray marching simply continues in the adjacent SSV.[2] Since SSVs influence only the simulation's execution speed and do not affect the random number stream, net molecule displacements, or reaction decisions, simulation results are identical regardless of partitioning [23, 25].

3 Parallel Implementation: MCell-K

With the serial MCell simulator it is possible to handle moderate size simulations such as those illustrated in Figs. 2A and 2B as long as only a few release sites are activated at time. Under these conditions there would be 10^4 to 10^5 diffusing particles, and simulations on a single processor workstation might last minutes to hours. However, we are interested in larger scale simulations, for example Fig. 2B under conditions where most of the 550 release sites might be activated, releasing millions of diffusing particles, and where run-times on a single processor worksta-

[1] This approach consumes memory inefficiently, and an improved implementation that employs 3D tetrahedral meshes is under construction.

[2] In the case of a parallel implementation some SSV boundaries coincide with processor boundaries, and molecules that cross such boundaries must be migrated accordingly.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

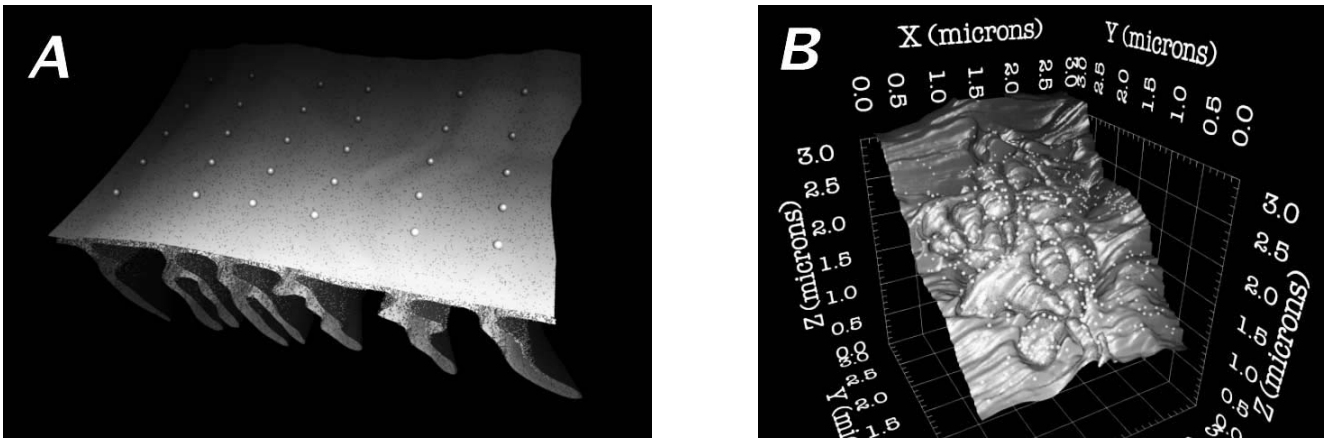

Figure 2. Simulations of typical complexity and size. A) Reconstruction of a rat diaphragm synapse contains around 70000 polygons and 30 neurotransmitter release sites (large white spheres). Model courtesy T. Bartol and J. Stiles. B) Serial EM tomographic reconstruction of a chick ciliary ganglion synapse composed of 300000 triangles and 550 neurotransmitter release sites (white spheres). Model courtesy J. Coggan, T. Bartol, E. Esquenazi, D. Berg, M. Ellisman, T. Sejnowski.

tion would range from days to months. Moreover, simulations often must be conducted at a reduced resolution in order to fit in the memory of a single processor.

The most costly portions of MCell's computations are the Brownian-dynamics random walk and binding events. These introduce considerable complexity into the parallelization process owing to the spatial and temporal dynamics of realistic microphysiological simulations which cause molecules to become distributed unevenly in space and time.

To enable MCell to scale to such large models under these conditions, we have implemented a parallel variant, called MCell-K, using the KeLP infrastructure [9, 10]. KeLP manages distributed pointer-based data structures, including molecule migration among processors, and facilitates load balancing.

We next discuss the major issues involved in parallelizing MCell, including data decomposition, detecting termination, ligand migration, and KeLP programming.

3.1 General Parallelization Strategy

The philosophy behind KeLP is to separate concerns surrounding code correctness from optimizations that affect performance. KeLP enabled us to work with a large production code while confining most changes to small regions of the code. We used existing data structures whenever possible and limited ourselves to a single parallel data type. This data type provided an impedance match between the data representation used in the MCell simulation algorithms, which was hidden from KeLP, and the representation required by KeLP, which was hidden from MCell. Given sufficient time, one might make more extensive and comprehensive changes to MCell, but such an approach is rarely realistic.

MCell-K divides the problem space into $N_x \times N_y \times N_z$ regions, assigning one region to each processor. Given an input with $S_x N_x \times S_y N_y \times S_z N_z$ spatial subvolumes and the decomposition described above, each processor is assigned $S_x \times S_y \times S_z$ SSVs. All processors read the geometry data from the input file, including surface and release site information, constructing the parts of the simulation geometry lying at least partially with the region of space defined by the set of SSVs owned. A surface triangle which crosses processor boundaries is instantiated on all processors containing at least part of the triangle's surface.

As the simulation progresses, each processor releases ligands within its spatial subvolumes, updates ligand positions, checks for interactions between ligands and surfaces, and performs reactions as necessary. Any particles that reach processor boundaries are communicated through a special-purpose KeLP data structure, which is described further below. The inner loop of MCell-K is the diffusion step, which is called once per time-step. The parallel algorithm for the diffusion step looks like this:

```
do until no processor has ligands to update:
  while there are local ligands to update:
    update next ligand
    if ligand crosses processor boundary:
      add ligand to communication list
  communicate ligands in communication list
```

We refer to each pass through the "`do until ...`" loop as one *sub-iteration*.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

In MCell-K, we implement processor boundaries as a special type of wall, similar to the way MCell treats SSV boundaries. Processor boundary walls are initialized at the outset of the simulation along with all other wall types. During the course of an MCell-K simulation, if the ray tracing algorithm detects that a ligand's trajectory intersects a processor boundary wall, the ligand is earmarked from migration.

3.2 Data Migration

The KeLP framework provides a distributed container class called an *XArray*, representing a parallel array of containers. Each container within an XArray is owned by a single processor, and all processors know the owners of each container. The container is defined by the user. The user need only define a few member functions to instantiate, marshall, and unmarshall the data structure which KeLP invokes to create an XArray and to move selected contents among processors [2].

We developed a KeLP container class for MCell-K to allow the standard MCell data structure, a ligand, to be communicated among processors. The KeLP class, *kelp_ligand_list*, is used only when dealing with ligands crossing processor boundaries. Each processor owns an array of kelp_ligand_lists containing one element for itself and one element for each of its neighbors. When ligands cross processor boundaries, they are appended to the kelp_ligand_list corresponding to the appropriate neighboring processor. When a KeLP data motion event is invoked, the ligands within each kelp_ligand_list are migrated to the respective neighboring process. Upon receipt, the ligands are appended to the processor's kelp_ligand_list element for local ligands. KeLP uses a simple geometric region calculus to determine relationships among neighbors from the information used to construct the XArray. This information will prove highly useful when we implement non-uniform partitionings to handle load balancing, as the logic behind data motion is the same in both the uniform and non-uniform cases.

Communication events in MCell are frequent (on the order of 5-15 times per second) but rather small, generally on the order of a few thousand bytes. Under KeLP's control, communication buffers are packed and unpacked by virtual functions defined by KeLP, which in turn invoke member functions defined by the user's container class. These buffers are uniform in size over all processors for a single communication event, but sizes are adaptively adjusted according to run-time dynamics. We combine communication buffer adjustment with termination detection (described below). We set the ligand buffer size to the maximum requested size across all processors, rounded to the next power of two. We found that our adaptive technique was faster than any fixed buffer size. A more efficient scheme might adjust buffer sizes according to the needs of individual pairs of communicating processors, but at present, communication is not a significant bottleneck.

3.3 Detecting Termination

Every ligand needs to completely traverse its path during each time-step. As a result, if any ligands cross processor boundaries, all processors need to pass through the "`do until ...`" loop described above at least twice: once for ligands originating on the processor and a second time for any incoming ligands. Since ligands may bounce off surfaces and return to their original processor or continue to a third processor, the number of sub-iterations cannot be known before run-time.

We detect termination of the "`do until ...`" loop through an all-reduce call, taking a maximum across processors. Processors report zero if they have no ligands left to update after the communication step; they send a non-zero value otherwise. If the result of the all-reduce is zero, we exit the "`do until ...`" loop. A more efficient approach might use nearest neighbors only, however, the global call is currently not a bottleneck and is far simpler to implement.

4 Experiments

As stated earlier, the goal of MCell-K is to offer improved capabilities for engaging in scientific discovery by solving problems more quickly and/or on a larger scale than would be possible with the serial implementation. In this section we will demonstrate that MCell-K produces results that are in agreement with those produced by the serial code and discuss scalability and performance.

4.1 Test Case

Our test case involves the simulation of a chick ciliary ganglion (Fig. 2B). The input to this simulation is a realistic structure reconstructed using serial electron microscopic tomography of a chick ciliary ganglion. The ciliary ganglion has a total of 550 release sites. Previous simulations, using the serial MCell code, have been able to compute a small fraction of this problem, releasing ligands from a total of only about 20 sites at a time. In our simulation we release 5000 ligands from each of 192 release sites, for a total of 960,000 ligands. The simulation runs for 2500 time-steps, or 2.5 ms of real time. We perform the simulation on 16, 32, and 64 processors and examine speedup, load balance, and communication requirements.

We have selected the 192 release sites (an average of 3 sites per processor in the 64-processor tests) at random, but

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

we have limited the maximum number of sites per processor to 6 in the 64-processor case. We thereby limit the load imbalance to a factor of 2 when running on 64 processors. A simulation involving all 550 release sites would have an average of 8.6 release sites per processor and a 25 release sites on the most heavily loaded processor, resulting in a load imbalance near 3. We note that, under our current uniform decomposition of the space, several release sites lie very near to processor boundaries. This situation is unavoidable without a more flexible decomposition method, which is currently under investigation.

Over the course of the simulation, ligands diffuse, react with receptors on the membrane, and are destroyed by enzymes on the membrane. In a healthy chick ciliary ganglion, ligands are destroyed fairly rapidly. For our test case, we reduce the rate at which ligands are destroyed by the enzyme on the membrane. We are thus able to simulate the effect of drugs which inhibit the enzyme, thereby allowing ligands to remain active for much longer periods of time, and, in the course of laboratory experiments, allowing additional properties of the synapse to be studied. Since many ligands survive throughout the the length of this simulation, we expect that ligands will be able to diffuse deep into the narrow crevices in the surface. Ligands trapped in these crevices may bounce off several walls per time-step, some crossing processor boundaries multiple times per time-step. This "persistent ligand" test case represents near the maximum amount of ligand bouncing that we expect from a biologically realistic simulation.

4.2 Experimental Setup

We ran on NPACI's Blue Horizon IBM SP system[3], located at the San Diego Supercomputer Center. Blue Horizon contains 144 POWER3 SMP High Nodes (model number 9076-260) interconnected with a "Colony" switch. Each node is an 8-way Symmetric Multiprocessor (SMP) based on 375 MHz Power-3 processors[4], sharing 4 Gigabytes of memory, and running AIX 5L. Each processor has 8 MB of 4-way set associative L2 cache, and 64 KB of 128-way set associative L1 cache. Both caches have a 128-byte line size. Each CPU has 1.5 GB/sec bandwidth to memory.

We ran with KeLP version 1.4[5] and used the installed IBM C++ compiler, mpCC. C++ code was compiled with compiler options `-O2 -qarch=pwr3 -qtune=pwr3`. We used the standard environment variable settings on Blue Horizon, and collected timings in batch mode using *loadleveler*. KeLP invokes MPI to handle communication, and we used the native IBM installation. Timings reported are based on wall-clock times obtained with `MPI_Wtime()`. Each simulation was performed 3 times. Variation among the runs was small. The times reported are averages for the 3 runs.

[3] `http://www.npaci.edu/BlueHorizon/`

[4] `http://www.rs6000.ibm.com/resource/technology-/sppw3tec.html`

[5] `http://www-cse.ucsd.edu/groups/hpcl/scg/kelp/`

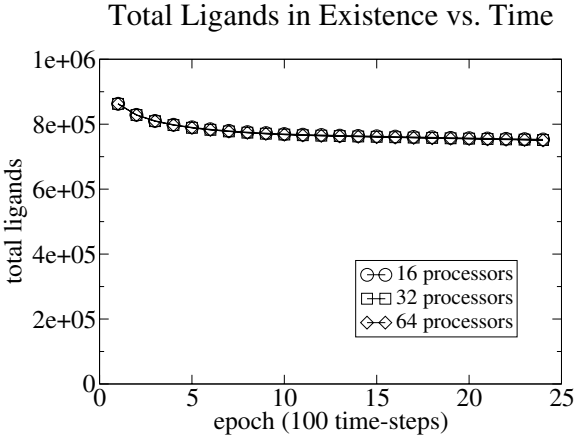

Figure 3. Ligands in existence throughout the simulation on 16, 32, and 64 processors. Each simulation releases 5000 ligands from each of the 192 release sites at time-step 0.

4.3 Results

When the function of enzymes within the chick ciliary ganglion is inhibited, most ligands survive for the 2500 time-steps of our simulation. The total number of ligands in existence is nearly identical throughout the course of the 16, 32, and 64 processor runs (Fig. 3). The fact that the total number of ligands in existence is so similar among the various runs is an indication that the parallel algorithm provides consistent results, independent of the number of processors.

In order to understand performance in greater detail, we report observables like running times over *epochs* comprising 100 time-steps. For each of the 16, 32, and 64 processor cases, we recorded the running times for every 100 time-steps during one simulation (Fig. 4). The total running times of the 16, 32, and 64 processor cases are shown in Table 1. The parallel efficiency in going from 16 to 64 processors is 85%. The table also shows both the number of ligand updates per processor and maximum number of ligand updates per second. (For the maximum number of ligand updates, we use the sum of the maximum at each epoch.)

Even with persistent ligands given a fair amount of time to diffuse, significant load imbalance remains throughout the calculation. The ratio of the maximum to the average number of ligands per processor is shown in Fig. 5. Since enzymes, like ligands, are non-uniformly distributed in the

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

Processors	Running Time	Efficiency vs. 16	Efficiency vs. 32	Ligand Updates Average	Ligand Updates Maximum	Max. Lig. Updates per Second
16	4903 s	—	—	121.1×10^6	210.5×10^6	42900
32	2652 s	92%	—	60.4×10^6	127.4×10^6	48000
64	1450 s	85%	91%	30.2×10^6	69.4×10^6	47900

Table 1. Running time, parallel efficiency, and ligand updates for the the chick ciliary ganglion simulation. The maximum number of ligand updates per processor is determined by summing the maximum per processor over all time-steps.

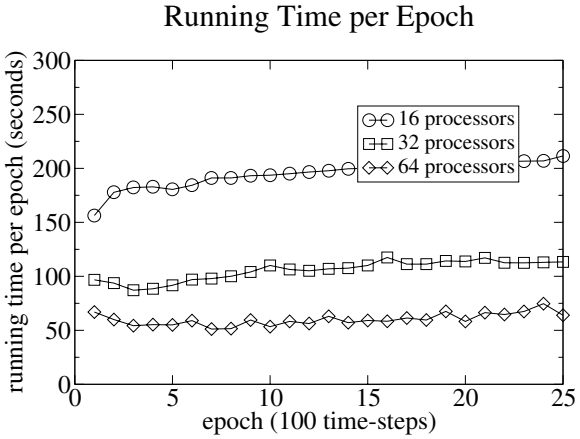

Figure 4. Running times for every 100 time-steps of the simulation on each of 16, 32, and 64 processors. Times are in wall-clock seconds.

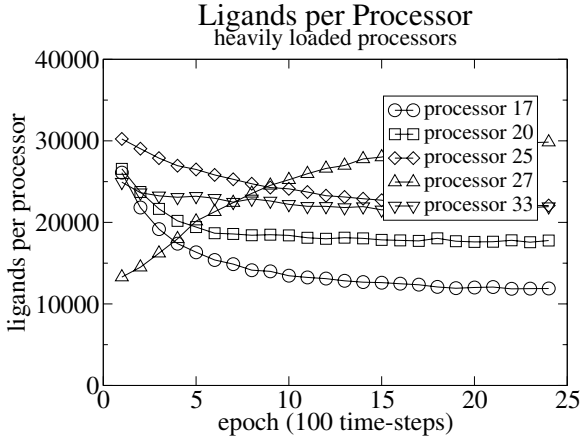

Figure 6. Ligands per processor for each of the most heavily loaded processors of the 64-processor simulation.

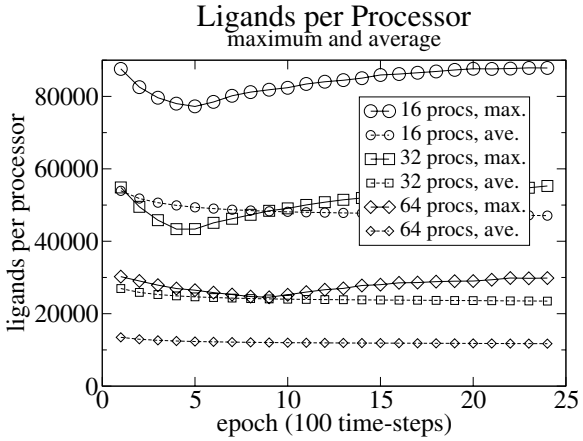

Figure 5. Maximum and average ligands per processor for every 100 time-steps of the simulation on each of 16, 32, and 64 processors.

ciliary ganglion, the population of ligands decreases at different rates on different processors, as seen in Fig. 6. Since no ligands are released after time-step 0, the increase in ligands seen on processor 27 is due entirely to diffusion from neighboring, heavily loaded processors.

Fig. 7 compares actual and predicted epoch running times, using the estimate of 42900 ligand updates per second (from Table 1). The predictions are in good agreement with the actual running times, though it appears that the estimate is a little high. We are currently investigating refinements to our model.

The number of ligands communicated in this simulation is shown in Fig. 8. The maximum number of ligands communicated appears to be inversely proportional to the number of processors used, as expected, since the length of each processor boundary is shorter as more processors are added. The number of sub-iterations, shown in Fig. 9, increases with the number of processors. We believe that the increased number of sub-iterations required, as well as some of the unevenness in the running time, may be due to

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

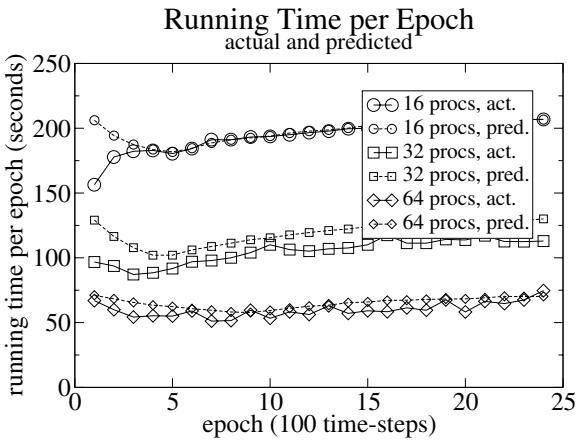

Figure 7. Actual and predicted running times for every 100 time-steps of the simulation on each of 16, 32, and 64 processors. Predicted running times are calculated by estimating 42900 ligand updates per second on the most heavily loaded processor.

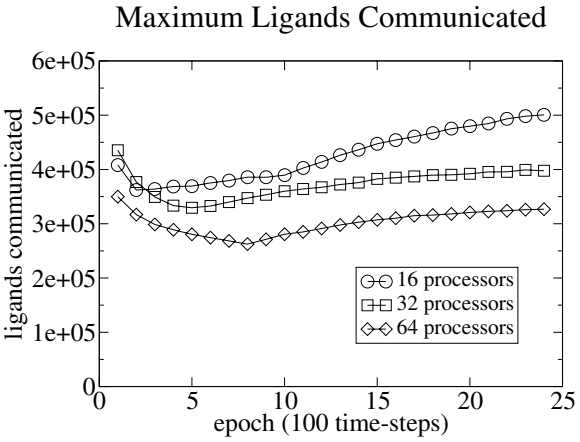

Figure 8. The maximum number of ligands communicated by a processor per 100 time-steps for each of the 16, 32, and 64 processor simulations.

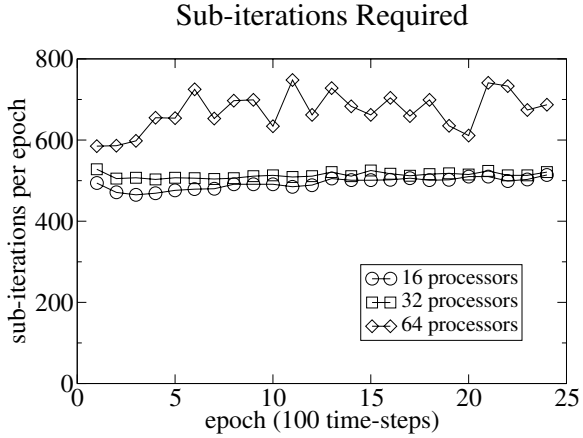

Figure 9. The number of sub-iterations required per 100 time-steps for each of the 16, 32, and 64 processor simulations.

frequent ligand bouncing, but we have not yet gathered sufficiently fine-grained data to be certain. The aggregate effects of ligand bouncing appear not to be severe, however.

5 Discussion

Load imbalance is currently a performance bottleneck and is costing us approximately a factor of two in running time. We are implementing a non-uniform decomposition (recursive coordinate bisection [5]) to ameliorate the problem and to make larger-scale runs feasible. We will report on the outcome in the future.

At present we collected summary performance data over epochs. We are in the process of collecting more detailed performance information, on the level of individual sub-iterations. This will help us learn more about communication costs, which are estimated to be small.

The added sub-iterations needed to detect termination are modest, but we need more detailed statistics to determine if there are larger variations masked by the current sampling technique. However, based on the current results, the costs are small. An all-reduce operation on Blue Horizon costs a few hundred microseconds on 64 processors. Even if there were 10 such calls per time-step, the cost would be on the order of a few milliseconds per time-step, and no more than a few seconds over the total length of the run. By comparison, runs take hundreds to thousands of seconds to complete, so we do not anticipate termination detection to be a problem even on thousands of processors.

The one remaining cost due to parallelization is ligand migration. Measurements have shown that message lengths progressively decrease over the course of a time-step. From

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

our statistics, we know that the most costly communication steps transmit a few thousand ligands per time-step, at roughly 50 bytes per ligand. Thus, a few hundred thousand bytes are transmitted per time-step. We may safely ignore message start time (about 20 microseconds averaging the differing on-node and off-node transmission costs) since it is small compared with the cost of the global synchronization. What remains is bandwidth-limited communication. Peak communication bandwidth is 450 MB/sec averaging on-node and off-node communication rates. Total time per time-step spent migrating ligands is therefore on the order of 1 millisecond, which is small compared with running times of 1/2 to 2 seconds per time-step, and is of the same rough order of magnitude as synchronization. Thus, we do not anticipate that communication will be an issue in even the largest scale simulations anticipated to run on thousands of processors.

Although our current scheme for estimating computation costs appears reasonable, we will soon be adding a new capability to MCell-K that permits bimolecular reactions among ligands. The resulting non-linearities in workload estimation will challenge our ability to balance workloads, and we are investigating better methods to estimate workloads.

6 Conclusions

We have built a parallel implementation of MCell with minimal modifications to the original serial code. We were able to achieve this result by using the KeLP programming system, which allowed us to manage distributed pointer-based data structures.

By parallelizing MCell, we were able to run simulations significantly more ambitious than could previously be performed with the serial code. We have demonstrated good parallel efficiency on up to 64 processors of NPACI's Blue Horizon system, and we plan to scale the application to thousands of processors after we have installed dynamic load balancing.

We believe that dynamic load balancing is critical to obtaining the maximum performance since ligand populations and rates of reactions vary so widely in realistic biological simulations. A first step toward load balancing MCell-K will be to use irregular decompositions, but we ultimately plan to over-decompose the problem domain, assigning multiple problem subregions per processor. This approach has previously been investigated in the context of KeLP [20], and by others [19, 28, 11] as well.

We believe that as we achieve better load balance within MCell-K, the importance of fast communication will increase. We are beginning to investigate ways to communicate ligands asynchronously, which we believe will reduce the amount of time that processors are idle due to load imbalance.

Planned enhancements to the MCell code, such as moving boundaries and ligand-ligand interactions, will further complicate our parallelization efforts.

Owing to statistical variations in Monte Carlo simulations, it is necessary to run multiple simulation invocations to obtain statistically meaningful results. An interesting possibility is to run MCell-K simulations on the grid as is done with the serial implementation of MCell [7].

7 Acknowledgments

Greg Balls and Scott Baden were supported by the National Partnership for Advanced Computational Infrastructure (NPACI) under NSF contract ACI9619020. Tilman Kispersky, Tom Bartol, and Terry Sejnowski were supported by NSF NPACI ACI9619020, NSF IBN-9985964, and the Howard Hughes Medical Institute. Scott Baden dedicates his portion of the work performed on this paper to Linda Dorfman (1938-2003). The MCell-K website is located at `http://www.cnl.salk.edu/~tilman`.

References

[1] L. Anglister, J. R. Stiles, and M. M. Salpeter. Acetylcholinesterase density and turnover number at frog neuromuscular junctions, with modeling of their role in synaptic function. *Neuron*, 12:783–94, 1994.

[2] S. B. Baden, P. Colella, D. Shalit, and B. V. Straalen. Abstract kelp. In *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*, Portsmouth, Virginia, March 2001.

[3] T. M. Bartol, B. R. Land, E. E. Salpeter, and M. M. Salpeter. Monte Carlo simulation of miniature endplate current generation in the vertebrate neuromuscular junction. *Biophys. J.*, 59(6):1290–1307, 1991.

[4] R. Beck, B. Erdmann, and R. Roitzsch. Kaskade 3.0: An object oriented adaptive finite element code. Technical Report TR 95-4, Konrad-Zuse-Zentrum fur Informationstechnik, Berlin, 1995.

[5] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. on Comput.*, C-36(5):570–580, May 1987.

[6] J. M. Bower and D. Beeman. *The book of GENESIS : exploring realistic neural models with the GEneral NEural SImulation System.* TELOS, Santa Clara, CA, 1995.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE

[7] H. Casanova, T. M. Bartol, J. R. Stiles, and F. Berman. Distributing MCell simulations on the grid. *Int'l. J. High Perf. Comp. Appl.*, 15:243–257, 2001.

[8] M. Engelman. *FIDAP Theoretical Manual, Version 6:02*. FDI, Evanston, IL, 1991.

[9] S. J. Fink. *Hierarchical Programming for Block–Structured Scientific Calculations*. PhD thesis, Department of Computer Science and Engineering, University of California, San Diego, 1998.

[10] S. J. Fink, S. B. Baden, and S. R. Kohn. Efficient run-time support for irregular block-structured applications. *J. Parallel Distrib. Comput.*, 50(1-2):61–82, April-May 1998.

[11] J. E. Flaherty, R. M. Loy, C. Özturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Parallel structures and dynamic load balancing for adaptive finite element computation. *Applied Numerical Maths.*, 26:241–263, 1998.

[12] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *J. Phys. Chem.*, 81:2340–2361, 1977.

[13] D. T. Gillespie. A rigorous derivation of the chemical master equation. *Physica A*, 188:404–425, 1992.

[14] M. Hines. A program for simulation of nerve equations with branching geometries. *Int. J. Biomed. Comput.*, 24:55–68, 1989.

[15] M. Hines. NEURON – a program for simulation of nerve equations. In F. H. Eeckman, editor, *Neural Systems: Analysis and Modeling*, pages 127–136. Kluwer Academic Publishers, Boston, MA, 1993.

[16] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. McGraw-Hill, 1981.

[17] H. H. McAdams and A. Arkin. Simulation of prokaryotic genetic circuits. *Annu. Rev. Biophys. Biomol. Struct.*, 27:199–224, 1998.

[18] D. M. Nicol. Rectilinear partitioning of irregular data parallel computations. *J. Parallel Distrib. Comput.*, 23(2):119–134, 1994.

[19] J. C. Phillips, G. Zheng, S. Kumar, and L. V. V. Kalé. NAMD: Biomolecular simulation on thousands of processors. In *Proceedings of SC 2002*, 2002.

[20] F. D. Sacerdoti. A cache-friendly liquid load balancer. Master's thesis, Department of Computer Science and Engineering, University of California at San Diego, June 2002.

[21] J. Schaff, C. Fink, B. Slepchenko, J. Carson, and L. Loew. A general computational framework for modeling cellular structure and function. *Biophys. J.*, 73:1135–1146, 1997.

[22] J. L. Smart and J. A. McCammon. Analysis of synaptic transmission in the neuromuscular junction using a continuum finite element model. *Biophys. J.*, 75(4):1679–1688, 1998.

[23] J. R. Stiles and T. M. Bartol. Monte Carlo methods for simulating realistic synaptic microphysiology using MCell. In E. DeSchutter, editor, *Computational Neuroscience: Realistic Modeling for Experimentalists*. CRC Press, 2001.

[24] J. R. Stiles, T. M. Bartol, E. E. Salpeter, and M. M. Salpeter. Monte Carlo simulation of neurotransmitter release using MCell, a general simulator of cellular physiological processes. In J. M. Bower, editor, *Computational Neuroscience*, pages 279–284, New York, NY, 1998. Plenum Press.

[25] J. R. Stiles, T. M. Bartol, M. M. Salpeter, E. E. Salpeter, and T. J. Sejnowski. Synaptic variability: new insights from reconstructions and Monte Carlo simulations with MCell. In W. Cowan, T. Sudhof, and C. Stevens, editors, *Synapses*. Johns Hopkins University Press, 2001.

[26] J. R. Stiles, I. V. Kovyazina, E. E. Salpeter, and M. M. Salpeter. The temperature sensitivity of miniature endplate currents is mostly governed by channel gating: evidence from optimized recordings and Monte Carlo simulations. *Biophys. J.*, 77:1177–1187, 1999.

[27] J. R. Stiles, D. van Helden, T. M. Bartol, Jr., E. E. Salpeter, and M. M. Salpeter. Miniature endplate current rise times less than 100 microseconds from improved dual recordings can be modeled with passive acetylcholine diffusion from a synaptic vesicle. *Proc Natl Acad Sci USA*, 93:5747–5752, 1996.

[28] J. D. Teresco, M. W. Beall, J. E. Flaherty, and M. S. Shephard. A hierarchical partition model for adaptive finite element computation. *Comput. Methods. Appl. Mech. Engrg.*, 184:269–285, 2000.

[29] M. Tomita, K. Hashimoto, K. Takahashi, T. Shimizu, Y. Matsuzaki, F. Miyoshi, K. Saito, S. Tanida, K. Yugi, J. Venter, and C. Hutchison. E-CELL: Software environment for whole cell simulation. *Bioinformatics*, 15(1):72–84, 1999.

0-7695-2132-0/04/$17.00 (C) 2004 IEEE