Toward Petascale Simulation of Cellular Microphysiology

Scott B. Baden*, Terrence J. Sejnowski†, Thomas M. Bartol† and Joel Stiles‡
* Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California, 92093-0404
Email: baden@cs.ucsd.edu
† The Salk Institute
San Diego, California, 92186-5800
Email: {terry,bartol}@salk.edu
‡ Pittsburgh Supercomputing Center and Carnegie Mellon University
Pittsburgh, Pennsylvania, 15213
Email: stiles@psc.edu

***Abstract*—MCell is a Monte Carlo simulator of cell microphysiology, and the scalable variant can be used to study challenging problems of interest to the biological community. MCell can currently model a single synapse out of thousands on a single cell. Petascale technology will enable significant advances in the ability to treat larger structures involving many synapses, with correspondingly more complex behavior. However, there are significant challenges to scaling MCell across two orders of magnitude in performance: increased communication delays and uneven workload concentrations. We discuss software solutions currently under investigation that will accompany us on the path to Petascale cell microphysiology.**

I. 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 time scales (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 in which the number of active or diffusing molecules within a subcellular compartment is very small—which is often the case.

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], [2], [3], [4], [5]. In this setting, Monte Carlo (MC) methods are uniquely able to simulate realistic biological variability and stochastic "noise." MCell can model local ion depletion or accumulation in regions of densely packed ion channels. It copes well with fine structure and complex three-dimensional boundary/initial conditions. More details about MCell can be found at the MCell web sites[1].

The need to simulate larger systems has motivated a parallel version of MCell, called MCell-K[2] [6], which was implemented using the KeLP infrastructure [7], [8]. This version of MCell currently scales to about 200 processors. We are currently investigating new programming techniques for improving scalability which are based on a run time system called Tarragon [9]. Tarragon offers the opportunity of improved load balancing together with the ability to mask communication delays with computation.

II. Simulating Cellular Microphysiology

MCell uses rigorously validated MC algorithms [10], [3], [11], [5] to track the evolution of biochemical events in space and time involving individual diffusing or immobile molecules. Diffusing molecules move according to a 3D Brownian-dynamics random walk and encounter cell membrane boundaries, as well as transmembranous, scaffolded,

[1] http://www.mcell.cnl.salk.edu and http://www.mcell.psc.edu
[2] http://www.cs.ucsd.edu/groups/hpcl/scg/mcellk

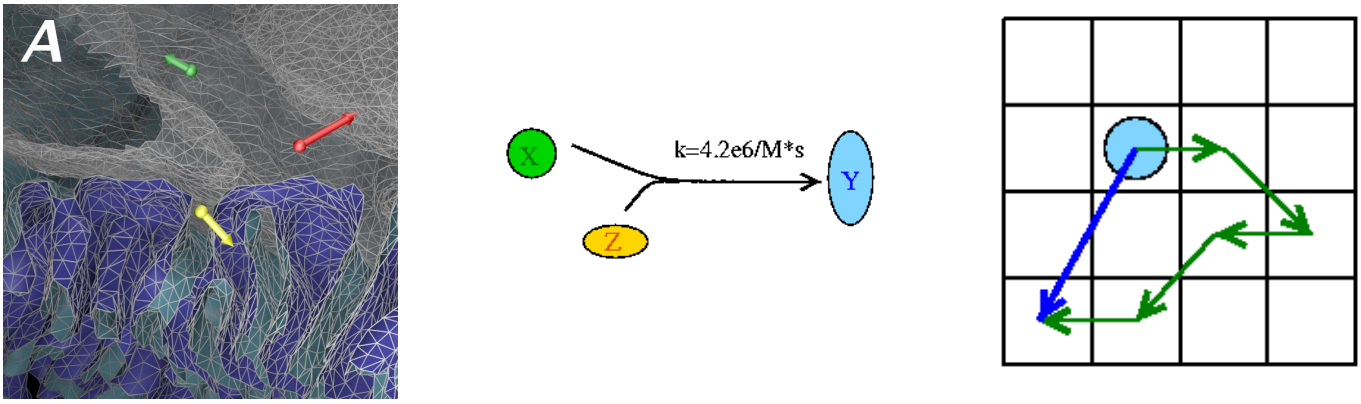

Fig. 1. a) Raytracing diffusion paths within a 3D reconstruction of nerve-muscle membrane represented as a triangular mesh. b) A biomolecular reaction. c) Diffusion entails taking a series of short random steps resulting in a net displacement. Part (a) was created with the assistance of P. Davidson. Serial electron micrographic sections for the reconstruction were cut by T. Deerinck in the laboratory of M. Ellisman (UCSD). Parts (b) and (c) courtesy R. Kerr.

and other diffusing molecules (Figure IIa). Encounters may result in chemical reactions governed by user specified reaction mechanisms (Figure IIB); as described below, molecular transitions are chosen according to a random selection process using probabilities derived from bulk solution rate constants [11]. 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 partial 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 [11]. Membrane boundaries are represented as triangulated meshes and may be arbitrarily complex. The defined structure and diffusion space is populated with molecules that interact probabilistically [5].

MCell simulations contain surface mesh objects, meshes composed of polygons, and 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, molecule unbinding, or transmembrane ion flux.

MCell employs unimolecular transition methods similar to the earlier Gillespie MC method [12], [13], [14]. The underlying computations for these transitions are relatively simple and readily parallelizable. In contrast, MCell's algorithms for Brownian-dynamics random walk and bimolecular events, which account for the bulk of computational cost, are unique to MCell and are considerably more difficult to parallelize, as will be discussed below. They cannot be treated as if it they were embarrassingly parallel.

The most common bimolecular event is *binding,* which may occur whenever diffusing (dimensionless) molecules encounter appropriate other molecules, such as diffusing signaling molecules or transmembranous receptor proteins on a surface. Each surface molecule is actually represented by a discrete "tile" on the surface. During every time-step each diffusing molecules must be traced along a random walk trajectory (i.e. a ray) to find potential intersections with mesh elements (triangles) of surfaces and collisions with other diffusing molecules in the volume (Figure IIC). Ray-triangle intersection tests are used to find the point of intersection with surface triangles. To detect collisions with other diffusing molecules the ray is used to sweep out a cylindrical volume along its length and all valid reaction partners that lie within this volume are considered to be collisions. 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 presence or absence of a reactive surface tile or molecule at the point of intersection, the properties of the mesh element, or both. 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 specified diffusion distance.

An acceleration structure is used to speed up the search for intersections between mesh elements and molecules, analogous to the chaining mesh structure described by Hockney and Eastwood [15]. The binning structure subdivides the computational volume into spatial subvolumes called *SSVs*. The SSV spacing is chosen to limit each SSV to no more than

a few mesh elements such that the search for intersections during ray marching can in turn be limited to the current and neighboring SSVs. The running time for searching 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. 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 the partitioning [11], [5].

III. Scalable Implementation

Using the serial MCell simulator it is reasonable to run moderate size simulations of about 10^5 diffusing molecule on a single processor workstation. The most costly portions of MCell's computations are the Brownian-dynamics random walk and binding events. These complicate 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, and give rise to a termination detection problem described below.

To enable MCell to scale to larger cell models, we implemented a parallel version, MCell-K [6]. We used the the KeLP infrastructure [7], [8], a C++ class library that manages distributed pointer-based data structures, including molecule migration among processors. The philosophy behind KeLP is to separate concerns surrounding code correctness from optimizations that affect performance. KeLP enabled us to work with a production application 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 with a legacy code.

MCell-K splits the problem space into regions, assigning one region to each processor. Each region contains many spatial subvolumes, including surface and release site information and 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.) The regions can have non-uniform sizes according to to the spatial distribution of diffusion molecules, but the geometry of the regions is restricted to tensor products of orthogonal cutting planes (Figure IIIc).

As the simulation progresses, each processor releases molecules within its spatial subvolumes, updates their positions, checks for interactions between molecules and surfaces, and performs reactions as necessary. Any molecules that reach a processor boundary are communicated by KeLP. Since molecules are fine grained entities (roughly 50 bytes each), it is not feasible communicate individual molecules as they cross a boundary. Thus, KeLP aggregates molecules and transmits them in toto at synchronization points called *sub-iterations.* While aggregation amortizes communication overheads, it has the side effect of introducing potential non-causal behavior. This non-causal behavior is eliminated by a termination detection protocol described 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 appears in Figure III, where each pass through the `do until` loop corresponds to a sub-iteration. Every molecule needs to completely traverse its path during each time-step. As a result, if any molecules cross processor boundaries, all processors need to pass through the "`do until ...`" loop at least twice: once for molecules originating on the processor and a second time for any incoming molecules. Since molecules may bounce off surfaces and return to their original processor or continue to a third processor, the number of sub-iterations is unpredictable and can be determined only at run time.

We detect termination of the "`do until ...`" loop using a collective reduction call (all-reduce), to see if any processor is holding any molecules for any other processor. If there are none, 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 simple to implement.

Communication events in MCell are intermittent, with message lengths on the order of tens of kilobytes. Under KeLP's control, migrating molecules are linearized into message buffers. The buffers are uniform in size over all processors for a single communication event, but sizes are adaptively adjusted according to run-time dynamics. We found that our adaptive technique was faster than any fixed buffer size. We combine communication buffer adjustment with termination detection, which was previously described.

IV. Results

We obtained results from DataStar, an IBM SP system located at the San Diego Supercomputer Center. We used DataStar's 8-way p655+ nodes, which contain 1.5GHz Power4+ CPUs and 16 Gigabytes of shared memory. We ran with KeLP version 1.4[3] and used the installed IBM C++ compiler, mpCC. C++ code was compiled with compiler options `-O2 -qarch=pwr4 -qtune=pwr4`.

We ran a problem obtained from a realistic structure reconstructed from serial electron microscopic tomography of a chick ciliary ganglion. The simulations ran for 2500 time-steps, or 2.5 ms of real time. Molecules are dispersed in groups of 5000 from the release sites, and gradually disperse into the outlying space, reacting and possibly becoming absorbed. During the time periods immediately after the release, the workload is concentrated near the release site. (Figure IIIa-d). The number of release sites effectively determines the maximum amount of parallelism that can be effectively realized in

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

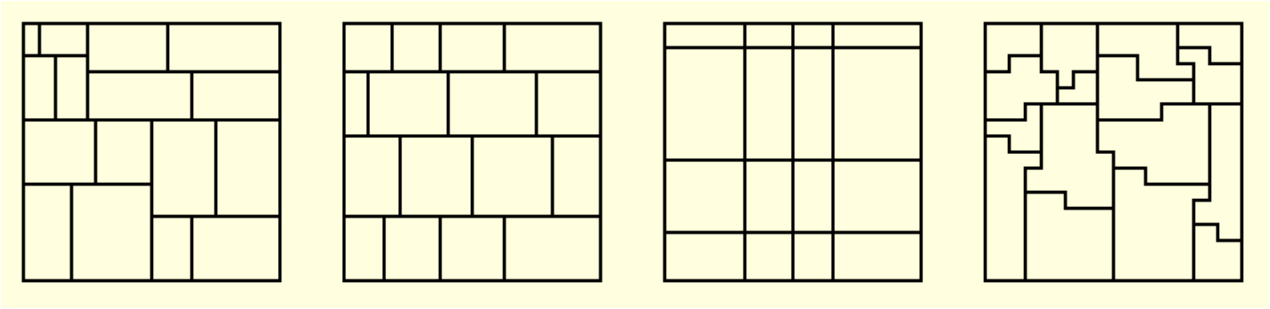

Fig. 2. Irregular partitionings in two dimensions, rendered with (a-b) Recursive coordinate bisection, (c) Rectilinear partitioning, and (d) and an Inverse Spacefilling Curve.

```
foreach timestep t
    do until there are no more molecules to update
      do while there are molecules to update on this processor
        update the next molecule
        if the molecule crosses a processor boundary:
          add the molecule to the communication list
      end do while
      communicate molecules in the communication list,
        exchanging with other processors
    end do until
end foreach
```

Fig. 3. The MCell timestepping loop.

Fig. 4. (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. Figures courtesy U. Rao[16].

a simulation. Thus, when we perform scaling studies, we vary the number of processors and release sites together.

The running time of an MCell simulation is a function of the geometry and characteristics of the surface membranes, the chemical reactions, and the number of released molecules. In the chick ciliary ganglia simulation, molecules diffuse, react with receptors on the membrane, and are destroyed by enzymes on the membrane. With normal enzyme activity, diffusing molecules are destroyed fairly rapidly. For our test case, we reduce the rate at which molecules are destroyed by the enzyme. We are thus able to simulate the effect of experimentally applied drugs that inhibit the enzyme, thereby allowing molecules to remain active for much longer periods of time. When the function of enzymes within the chick ciliary ganglion is inhibited, most molecules survive for the 2500 time-steps of our simulation. The total number of molecules in existence is nearly identical throughout the course of the runs.

The chick ciliary ganglia model has a total of 550 molecule release sites, distributed non-uniformly throughout the simulation volume. Our earlier results were obtained using static uniform partitioning [6], [17]. To mitigate the effects of the non-uniform distribution on load imbalance, we randomly sampled 384 of the 550 release sites. This strategy reduces the load imbalance, and enables us to obtain useful information about communication overheads which would otherwise be

obscured by processor idleness due to severely imbalanced workloads. (We'll return to the subject of load balancing shortly.)

Table I summarizes results obtained on 48, 96, and 192 processors. The parallel efficiency in is 89% going from 48 to 192 processors. Beyond 192 processors, load imbalances hampers scalability.

We have obtained recent results with non-uniform partitioning by taking an advantage of a feature of the original serial MCell code. This feature enables us to partition the simulation volume by taking the tensor product of sets of orthogonal cutting planes, each set spaced non-uniformly along its respective axis (Figure IIIc). This partitioning is known as rectilinear partitioning[18] and has enabled us to reduce load imbalance significantly. However this strategy is not sufficient to scale to the petaflop level, owing to the geometric restrictions on how the partitions are constructed.

To scale to the petaflop level, we need a finer grained partitioning strategy which has more flexibility in accommodating unevenly concentrated loads. U. Rao [16] performed trace driven simulations and found that dynamic decomposition based on spacefilling curves [19], [20], [21], [22] (Figure IIId) can improve load imbalance significantly for a reasonable cost. However, she also found that communication costs would increase significantly, perhaps as much as a factor of 4 or 5. This comes as the side effect of improved load balancing. In order to better balance the workloads, processors must evenly share regions of space that are heavily populated by molecules in transition, and this implies that some cuts will pass directly through the heavily populated regions, disrupting locality. The net effect is to increase the frequency that molecules move across processor boundaries, and hence raise communication overheads.

We had previously estimated that communication costs account for between 7% to 17% of the overall running time in the chick ciliary ganglion problem [17]. However, these results were obtained with uniform static partitionings. Recent results with rectilinear partitionings obtained by Jose Garcia Moreno-Torres have validated our projections of significant increases in communication overhead, as a side effect of load balancing.

At this point the calculation becomes communication bound. A strategy for tolerating communication delays is vital to scaling performance to thousands of processors.

Reformulating an algorithm to tolerate communication delays generally involves a split phase design using asynchronous, non-blocking communication. Policy decisions such as scheduling tend to be embedded in the application software, with the result that correctness and performance concerns become intertwined. The decisions depend on the hardware, resulting in non-robust performance. Software development will tax even the expert programmer.

To this end we are investigating a graph-based execution model inspired by dataflow, called Tarragon. Tarragon is a programming model and C++ class library implementing the model [9]. As in classic data flow [23], [24], [25], the Tarragon model captures parallelism among independent tasks, with interdependent tasks enabled according to the flow of data among them. The effect is to couple data motion with computation —a model that matches the behavior of migrating molecules in MCell simulations. Run time services manage the data flow semantics and task scheduling, freeing the user from having to manage the low level details. In studies with a synthetic benchmark that mimics the essential behavior of MCell, we have successfully overlapped significant fractions of communication overhead, thus enhancing scalability[26], [9]. Tarragon supports slackness, also known virtualization or overdecomposition [27], [28], [29], [30], whereby each processor is assigned many pieces of work. There are two benefits obtained by using slackness. Slackness increases the amount of potential concurrency of computation and data motion, improving the ability to pipeline the two activities. It also facilitates load balancing by reducing the granularity of work units.

V. Discussion

Using current technology we can simulate realistic cell structures on up to about 200 processors, perhaps as many as a thousand with improved load balancing. Load balancing will remain a challenge in problems that generate concentrated workloads impulsively. The best we can hope for is that the workloads disperse if given sufficient time. Efforts to improve load imbalance will inevitably impact communication bottlenecks, but the effect of load imbalance and communication overheads will be heavily problem dependent.

Alternative programming models such as Tarragon provide the support for latency tolerance that will be vital to scaling to Petascale class systems with many thousands of processors. Most production programming on scalable systems tends to be done with MPI, but research is needed to explore alternatives that support latency tolerant formulation and deep parallel memory hierarchies based on multicore processor building blocks interconnected in novel ways.

What we can accomplish now is to examine a single synapse out of thousands on a single cell. There are hundreds of synapses on a single branch of a dendrite and single axons make hundreds of synapses on neighboring neurons. By scaling up to petascale performance levels we will be able to explore the interactions between these structures in a larger volume of the neuropil, including the glial cells that ensheath the axons, dendrites and synapses.

The MCell simulator is computationally intensive but owing to the complex dynamics of chemical reactions, performance may not be linear in the number of computational elements. Thus, petaflop rates are not as meaningful as quantities such as: the rate at molecular chemical reactions can be updated or how large a cellular volume can be updated in a given amount of time. We have found that while performance can be modeled approximately, precise estimates are elusive: chemical reactions varying running times, and there can can be many types of reactions in a given simulation.

We are currently running on about 1/1000 the number of processors anticipated on a Petaflops-scale system. We

TABLE I

RUNNING TIME, PARALLEL EFFICIENCY, AND COMMUNICATION COSTS FOR THE THE CHICK CILIARY GANGLION SIMULATION. THE COMMUNICATION TIMES DO NOT INCLUDE THE COST OF THE GLOBAL REDUCTIONS AND BARRIERS, WHICH ARE NEGLIGIBLE. THE INFERRED COST WAS OBTAINED BY DOUBLING THE SIZE OF THE MESSAGE BUFFERS, AND IS PROBABLY MORE ACCURATE.

Processors	Time	Eff v. 48 Proc	Communication (measured)	Communication (inferred)
48	1600s	—	50s	8s
96	808s	99%	47s	10s
192	469s	89%	81s	32s

estimate that we will be able to fully engage such a system by running simulations with O(10^8) diffusing molecules. Examples of candidate projects include calcium-mediated synaptic plasticity in brain and neuromuscular synapses, excitation-contraction coupling in cardiac muscle cells, and biochemical signaling cascades and metabolic networks in other non-neuronal cells. At this stage, simulations of synaptic plasticity are already under development, using either brain tissue (hippocampus) reconstructed from large-scale electron microscopic datasets, or neuromuscular architecture created in silico with computer-aided design software.

All of these candidate projects will be much more complex than the ciliary ganglion model described above and will involve multiple signaling ligands, calcium ions, ion channels, receptors, pumps, enzymes, kinases, etc., in large cellular volumes (on the order of 10^4 μm^3). Furthermore, each will require simulation of interactions between diffusing molecules in solution as well as diffusing molecules in membranes. The recent serial version of MCell (MCell3) introduces these new diffusion-reaction algorithms, as well as event scheduling and adaptive time and space steps for moving molecules. These new requirements and capabilities introduce additional challenges for efficient parallelization, and overlapping of communication and computation will be critical. Another challenge will be to handle on-chip and off-chip communication efficiently for petascale architectures, i.e., massive collections of relatively large multicore units. Looking farther into the future, the next generation of biological simulations will couple existing Monte Carlo diffusion-reaction methods to finite element or finite volume computations in adaptive volume meshes. At that point cellular electrical activity and shape changes (e.g., nerve impulses, blood cell motility, muscle cell contraction) will be coupled quantitatively to subcellular biochemistry. To some extent load imbalance issues may be attenuated by the addition of finite element or finite volume computation throughout the entire volume mesh subdivided across processors. On the other hand, mesh adaptation at different spatial and temporal scales in different spatial regions will reintroduce additional complexities for efficient large-scale parallelization. Given these expected difficulties and the scale of future biological computation, perhaps the most useful performance metric will become biological success rather than conventional measures of parallel scalability. In other words, will we be able to effectively simulate the behavior of a neural network embedded in a 3D model of brain tissue, or the electromechanical behavior of cardiac muscle cells embedded in the ventricular wall, regardless of how "efficiently" a petascale machine is being utilized? If the ultimate measure is human health, chances are an individual who gains benefit from realistic physiological simulations will not be too concerned about "inefficiency" as we develop the necessary computational infrastructure.

ACKNOWLEDGMENTS

Scott Baden was supported NSF NPACI ACI9619020 and NSF ACI0326013. Tom Bartol and Terry Sejnowski were supported by NSF NPACI ACI9619020, NSF IBN-9985964, NIH R01 GM069630 and the Howard Hughes Medical Institute. Joel Stiles was supported by NIH R01 GM069630 and P41 RR06009.

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*, vol. 12, pp. 783–94, 1994.

[2] 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*, vol. 93, pp. 5747–5752, 1996.

[3] 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 *Computational Neuroscience*, J. M. Bower, Ed. New York, NY: Plenum Press, 1998, pp. 279–284.

[4] 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.*, vol. 77, pp. 1177–1187, 1999.

[5] 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 *Synapses*, W. Cowan, T. Sudhof, and C. Stevens, Eds. Johns Hopkins University Press, 2001.

[6] G. T. Balls, S. B. Baden, T. Kispersky, T. Bartol, and T. J. Sejnowski, "A large scale monte carlo simulator for cellular microphysiology," in *Proceedings 18th International Parallel and Distributed Processing Symposium*, Santa Fe, NM, April 2004, pp. 42–51.

[7] S. J. Fink, "Hierarchical programming for block–structured scientific calculations," Ph.D. dissertation, Department of Computer Science and Engineering, University of California, San Diego, 1998.

[8] S. J. Fink, S. B. Baden, and S. R. Kohn, "Efficient run-time support for irregular block-structured applications," *J. Parallel Distrib. Comput.*, vol. 50, no. 1-2, pp. 61–82, April-May 1998.

[9] P. Cicotti and S. B. Baden, "Asynchronous programming with tarragon," in *Proc. 15th IEEE International Symposium on High Performance Distributed Computing, 2006*, Paris, France, June 2006, pp. 375–376.

[10] 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.*, vol. 59, no. 6, pp. 1290–1307, 1991.

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

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

[13] ——, "A rigorous derivation of the chemical master equation," *Physica A*, vol. 188, pp. 404–425, 1992.

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

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

[16] U. R. Venkata, "A performance model and load balancer for a parallel monte-carlo cellular microphysiology simulator," Master's thesis, Department of Computer Science and Engineering, University of California, San Diego, 2004.

[17] G. T. Balls, S. B. Baden, T. Bartol, and T. J. Sejnowski, "Mcell-k: A highly scalable cell microphysiology simulator on blue gene l," in *Proc. 12 SIAM Conf. Parallel Processing for Scientific Computing*, San Francisco, CA, Feb. 2006.

[18] D. M. Nicol, "Rectilinear partitioning of irregular data parallel computations," *J. Parallel Distrib. Comput.*, vol. 23, no. 2, pp. 119–134, 1994.

[19] M. S. Warren and J. K. Salmon, "A parallel hashed oct-tree n-body algorithm," in *Supercomputing*, 1993, pp. 12–21.

[20] J. R. Pilkington and S. B. Baden, "Dynamic partitioning of non-uniform structured workloads with spacefillin g curves," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, pp. 288–300, March 1996.

[21] S. Aluru and F. Sevilgen, "Parallel domain decomposition and load balancing using space-filling curves," in *International Conference on High-Performance Computing (1997)*, 1997, pp. 230–235.

[22] M. Parashar, J. C. Browne, C. Edwards, and K. Klimkowski, "A common data management infrastructure for adaptive algorithms for pde solutions," in *Supercomputing '97: Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*. ACM Press, 1997.

[23] J. Dennis, "Data flow supercomputers," *IEEE Computer*, vol. 13, no. 11, pp. 48–56, 1980.

[24] J.R. Gurd and C. C. Kirkham and I. Watson, "The Manchester prototype dataflow computer," *Communications of the ACM*, vol. 28, no. 1, pp. 34–52, January 1985.

[25] "Executing a program on the mit tagged-token dataflow architecture," *IEEE Transactions on Computers*, vol. 39, no. 3, pp. 300–318, March 1990.

[26] P. Cicotti and S. B. Baden, "Asynchronous cellular microphysiology simulation," in *Proc. 12 SIAM Conf. Parallel Processing for Scientific Computing*, San Francisco, CA, Feb. 2006.

[27] 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.*, vol. 184, pp. 269–285, 2000.

[28] L. V. Kalé, "The virtualization model of parallel programming : Runtime optimizations and the state of art," in *LACSI 2002*, Albuquerque, October 2002.

[29] K. Devine, J. Flaherty, R. Loy, and S. Wheat, "Parallel partitioning strategies for the adaptive solution of conservation laws," in *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, I. Babuška, J.E. Flaherty, W.D. Henshaw, J.E. Hopcroft, J.E. Oliger, and T. Tezduyar, Ed., vol. 75. Springer-Verlag, Berlin-Heidelberg, 1995, pp. 215–242.

[30] 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.*, vol. 26, pp. 241–263, 1998.