Chapter #

SIMULATION-DRIVEN DYNAMIC CLAMPING OF NEURONS

G. T. BALLS[a], S. B. BADEN[a], T. M BARTOL[b], T. J. Sejnowski[b]
[a]*Department of Computer Science and Engineering, University of California, San Diego, La Jolla, California 92093-0114, USA*
[b]*The Salk Institute for Biological Studies, La Jolla, California 92186-5800 USA*

Abstract. *We describe an experimentation environment that enables large-scale numerical simulations of neural microphysiology to be fed back onto living neurons in-vitro via dynamic wholecell patch clamping – in effect making living neurons and simulated neurons part of the same neural circuit. Owing to high computational demands, the experimental testbed will be dispersed over a local area network comprising several high performance computing resources. Parallel execution, including feedback between the simulation components, will be managed by the Tarragon, a programming model and run time library that supports asynchronous data driven execution. Tarragon's execution model matches the underlying dynamics of Monte Carlo simulation of diffusive processes and it masks the long network latencies entailed in coupled dispersed simulations. We discuss Tarragon and show how its data driven execution model can be used to dynamically feed back the results of a neural circuit simulation onto living cells in order to better understand the underlying signaling pathways between and within living cells.*

Keywords. Data driven execution, dynamic clamping, cell microphysiology simulation

1. INTRODUCTION

Neurons in the brain connect with each other to form exceedingly complex networks of neural circuits which carry and transform information from neuron to neuron and circuit to circuit. At the same time, each individual neuron in the network is a living cell that contains within it complex biochemical signaling pathways which govern the behavior the cell. Since the information carried by the neural circuits physically exists in the form of the spatio-temporal state of these signaling pathways, the behavior of the circuits depends not only on the macroscopic neural network topology but also on the way information transforms and is itself transformed by the microscopic signaling pathways. That is, the neural circuits between cells and the signaling pathways within cells are coupled, spanning many length and time scales, to form a massive information processing system.

F. Darema (ed.), Dynamic.Data Driven Applications Systems, pp-pp
© 2004 *Kluwer Academic Publishers. Printed in the Netherlands.*

Given the range of these scales and accompanying phenomena, study of the brain involves many techniques and disciplines approaching the problem from the top down and from the bottom up. A major challenge is how to integrate the understanding gleaned from disparate experimental paradigms and levels of analysis into one coherent picture. One promising technique that attempts to bridge the gap between single neurons and small neural circuits involves the use of dynamic whole-cell patch clamping of living neurons and numerical simulations of neural circuits.

The dynamic whole-cell patch clamp technique involves using thin slices of living brain tissue placed in a special environmental chamber and viewed through microscope. The brain tissue is sliced in such a way that damage to the local neural circuitry within the slice is kept to a minimum. This permits study of the behavior of single neurons within the context of the neural circuits of which the neurons are integral components. Under the microscope, stimulating and recording electrodes are placed in the tissue. The recording electrodes are connected to electronic equipment which allow measurement of the voltage and current in a single neuron without disturbing the neighboring neurons. Early versions of this equipment also made it possible to hold the voltage or current of the cell at a constant command value, that is, to "clamp" the voltage or current at a constant value. Modern versions of this equipment allow voltage or current clamping to a dynamically changing command value—this is called a *dynamic clamp*. If the dynamically changing command value is computed in real-time based on the recent behavior of the clamped cell it becomes possible to dissect subtleties of neural circuit dynamics that would remain inaccessible by other means. Now the question becomes: how realistic and complex a computational model can one use to generate the dynamic command value in real-time on available hardware?

2. ASYNCHRONOUS SIMULATION

2.1 Simulation methodology

Our starting point is a general Monte Carlo simulator of cellular microphysiology, called MCell [2,45,42,44,43]. At the heart of the cell simulator is a 3D random walk that models diffusion using a Monte Carlo method. A highly scalable variant of MCell called MCell-K [8] has been implemented using the KeLP infrastructure [18,19], but a new variant is currently under investigation that employs asynchronous, data driven execution, and is described below. The complexity in implementing the random walk on a scalable platform comes as the result of three factors: (1) the unpredictable nature of molecular motion due to random walks and encounters with cell membranes, (2) time-dependent molecular concentrations, and (3) the need to enforce causality.

As molecules follow their random walk through space they may occasionally change processor owners and hence incur communication. This communication is currently handled in bulk by the KeLP implementation: migrant molecules are collected until the end of the current timestep and moved en masse to their destination. Since molecules may reflect off of cell membranes, or be released after becoming bound, termination of each timestep must be detected to avoid non-causal

behavior. If a processor were to begin the next timestep, and a "straggler" molecule arrived that had not completed its random walk in the previous timestep, then the computation would be in error. This is a classic problem in parallel discrete event simulation [21]. MCell-K employs a conservative strategy [14,34] for ensuring correctness, synchronizing several times each timestep in order to detect termination.

The behavior of neural circuits spans a multitude of length and time scales. For the reasons explained above, we envision computational models that must include not just the macroscopic behavior of neural circuits—such as arises from simple integrate-and-fire neurons as modeled by NEURON [23]—but also the microscopic behavior of individual synapses and downstream signal transduction cascades (e.g. as modeled by MCell). NEURON is widely-used modeling environment for simulation of the electrophysiology of neurons and neural circuits. The MCell simulations of microscopic behavior will be integrated with neural circuit-level simulations handled by NEURON. In turn these will be fed into the dynamic patch clamp assembly.

2.2 Requirements

Our simulations introduce two requirements: they are computationally demanding and they entail processing of asynchronous events. To meet our resource requirements we will run simulations on a collection of high performance computing resources dispersed over a local area network. To support asynchronous event processing we will employ a data driven execution model which is currently under investigation. This model is called *Tarragon* and it will be implemented as a run time library.

Asynchronous data driven execution actually plays two roles. First, it manages physical processes that are inherently asynchronous, such as the random walk process of the Monte Carlo simulation algorithm, and the coupling of the numerical model output to the dynamic patch clamp. Second, asynchronous execution is invaluable in managing the complexity of writing latency tolerant algorithms. Many scientific users lack the background or the inclination to manage asynchronous execution and are already burdened with the issues surrounding parallelization.

2.3 The Pitfalls of Masking Communication

During the course of making their random walks molecules will inevitably bounce off cell membrane surfaces. Their precise trajectory cannot be known in advance, though the maximum distance a molecule can diffuse in a single timestep is bounded. The only way to be certain that all walks have completed is to have processors exchange local completion information. The computation associated with detecting termination takes time to settle; on average 6 to 8 synchronization points are required per timestep in MCell-K.

While the cost of such synchronization is not significant on scalable platforms it is expected to grow significantly when computations are dispersed over a network. In order to meet real time constraints it is important that communication be kept out

of the critical path. We can avoid the unnecessary synchronization steps by permitting ligands to migrate instantaneously, i.e. using single-sided communication. However, the resultant communication costs would be unacceptable since such communication is inherently fine grained–each ligand consumes roughly 50 bytes. While techniques for realizing overlap can reduce communication costs e.g. via a communication proxy [5,33], significant and intrusive programmer intervention is required to reformulate the algorithm [6]. Many programmers lack the background to manage split-phase execution, computation re-ordering, and the bookkeeping needed to handle overlap effectively. A data driven execution model is inherently well matched to handling communication overlap since it can order tasks dynamically according to the availability of data, without requiring programmer intervention.

3. RUN TIME SUPPORT ISSUES

Two run time support issues arise in our application: load balancing and sceduling. A good load balancing strategy helps ensure that the work is assigned fairly over all the processors. This is important since any load imbalance will exacerbate communication wait times [18]. A good schedule helps ensure that the processors are making progress along the critical path which is essential in order to meet real time deadlines.

3.1 Partitioning

Due to uneven time-dependent concentrations of molecules, a load balancing problem arises. Molecules must periodically be shuffled among processing nodes without seriously disrupting locality. Domain specific load balancing utilities are generally used to evenly assign work to processors. Our current plant is use a space-filling curve to subdivide the workload, which will help conserve locality [41].

Under this strategy, we “over-partition” computations into chunks such that each processing node obtains several pieces of work [46]. (If the processing node has multiple CPUs, then the processors may share the using processors self-scheduling.) Chunks migrate gradually along the space filling curve in response to changes in the workload distribution [17, 20, 27, 40,37].

There are three desirable aspects of this strategy: it (1) preserves locality, (2) facilitates communication overlap via software pipelining, and (3) relies on workload migration in lieu of data repartitioning, obviating the need for empirically derived models to estimate workloads [4]. There is an advantage to removing the need for empirical models. They are data dependent and would continually change: MCell’s computational techniques are evolving in order to meet new simulator requirements.

Dynamic load balancing of task graphs has been employed in the SCIRun [25] and Uintah [36] programming environments. While SCIRun relies on shared memory, some progress has been made in a hierarchical load balancer for clustered SMP systems. Taylor and co-workers [31] manage load balancing of structured adaptive mesh refinement [11,10] on distributed systems. The systems consist of a

small number of machines—two Origin 2000 systems. The scheduler carries out local and global load balancing, and also takes into account network delays. Parashar and others have treated a similar problem in a structured setting [38,39]. SMARTS [47] supports integrated task and data parallelism for MIMD, and provides an API for coarse-grain macro-dataflow. It relies on work-stealing[35], and has been demonstrated only on shared memory. Related work has primarily treated functional parallelism, e.g. CILK[32], Mentat[22], and others [3]. OSCAR[28] has similar goals to SMARTS, but operates on static graphs.

3.2 Scheduling

In a latency tolerant formulation, each processor will communicate with others to exchange migrating molecules and to perform termination tests. The precise order in which a processor executes its assigned work and carries out communication can dramatically affect performance. Concerns surrounding locality, dynamically varying workloads, and communication performance are at issue. For example, it may be advantageous to preferentially schedule random walks involving nearby molecules in order to enhance memory locality in accesses to the data structures that represent surfaces the molecules react with. This in turn implies that a separate scheduling algorithm is needed to manage communication: molecules that are migrating to the same processor should be communicated nearby in time, so that the preferential scheduling algorithm will have the opportunity to schedule the events as intended. A good schedule can also enhance communication overlap so that there is sufficient available computational work to overlap with communication. Since scheduling policies may differ among applications (and even the initial data), it s important to separate scheduling and algorithm correctness concerns in order to improve application performance robustness.

4. TARRAGON

To meet our requirements, we are investigating a programming model with data driven [24] execution semantics. Under such a model, data motion triggers computation and vice versa. The execution model is fundamentally different from Bulk Synchronous parallelism, which divides communication and computation into distinct phases, and provides a cleaner way of handling ligands that cross processor boundaries during the course of making their random walks. The flow of the data rather than the success of heroic programming determines the ability to overlap communication with computation and the scheduler can ensure that the processor makes timely progress along the critical path with respect to communication and real time deadlines. Owing to the use of overdecomposition, there will be plenty of random walk computations available to overlap with the communication and workloads can be migrated automatically to ensure that workloads can be evenly balanced.

We are implementing our experimental software testbed to support the data driven execution model along with new scheduling algorithms and load balancing

strategies. These will be implemented as a set of run time libraries called *Tarragon*. We are also investigating *parameterized scheduling algorithms* such that an application can communicate performance hints in the form of *performance*. These metadata have the capacity to articulate scheduling changes that can improve performance without affecting correctness.

4.1 Theory of operation

Tarragon supports task parallelism in which computations are described by a directed graph constructed at run time. The vertices correspond to tasks to be executed, the edges correspond to data dependences between the tasks. A task graph is distributed across processing modules, and each module is assumed to comprise multiple CPUs sharing a common address space. Memory is not shared across modules.

The Tarragon Model operates with the assistance of an entity known as the Mover-Dispatcher. The Mover-dispatcher runs concurrently with the application and is hidden from the view of the casual programmer. As with traditional data flow [16,26,1], parallelism arises among independent tasks. Dependent tasks are enabled according to the flow of data among them. Task firing rules are non-strict in Tarragon. First, a task may fire as the result of the flow of data across an individual edge. Second, data may be treated as a stream and a task may fire upon arrival of a subset of a stream. The Mover-Dispatcher is in charge of moving data along the edges of a TaskGraph and handling task enablement

The Tarragon philosophy is to support data motion with operations that have an intuitive cost model rather than to hide the activity. It provides a simple data motion primitive: push( ). A call to push( ) may or may not cause information to be moved immediately. The Mover/Dispatcher decides when to actually carry out the required communication under the advice of the Scheduler. The specific mechanism that the Mover-Dispatcher uses to move data among tasks is hidden from the user. It may involve a call to MPI, TCP/IP, or a simple memory copy if the source and destination tasks occupy the same address space.

Incoming data is processed by the Mover/Dispatcher, which makes a callback to a user-defined handler to deserialize the incoming data into the user data structures associated with the receiving TaskGraph node. There is no need for programmer intervention to invoke the handler and the Tarragon run time system will incorporate incoming data without interference into data structures that are involved in running computations. A task can become enabled when data has arrived on one or more input edges. The decision to make a task runnable is made by the scheduler on the Mover-Dispatcher's behalf. Scheduling will be discussed in detail below (§4.3).

There is one other issue that must be handled: termination. When a task runs out of molecules it may or may not have completed execution for the current timestep. The reason is that at some later time during the timestep a molecule may enter the region of space owned by the task. At this time, the task is made runnable. The task will eventually execute at a time determined by the Scheduler. Eventually the task will complete the current timestep and may proceed to the next one. Tasks communicate with other tasks when they exhaust their workload and this information is used to detect termination.

4.2 Coupling

The Mover/Dispatcher effectively isolates an application from policy decisions concerning scheduling and data motion. Thus, the activities may be customized to the application and system configuration in order to meet real time requirements, and to tolerate the multiple scales of latency inherent to a hierarchically constructed computing platform.

The TaskGraph may be used not only to represent the MCell simulations, but also to represent the coupled program structure that feeds the result of the simulations to the *in-vitro* patch clamp assembly. Results from the simulations are fed into the assembly at the time they become available, avoiding the need for polling. The component subtasks may vary in their computational requirements but the Tarragon run time system will allocate an appropriate amount of resources to each simulation invocation under direction of the Scheduler, which is in turn invoked by the Mover-Dispatcher.

4.3 Scheduling

Scheduling plays an important role in optimizing performance as its goal is to enable progress along the critical path of outstanding communication *and* computation. A good scheduler must address competing concerns surrounding locality, communication latency and real time concerns, and it is important that all concerns be balanced. Scheduling has received considerable attention in recent years. Beaumont et al. [9] advocate bandwidth centric scheduling of equal sized tasks on heterogeneous processors. Such scheduling may be useful in allocating tasks to the same processing node based on the carried workload and communication costs. Affinity hints have been employed by others to support locality (e.g. COOL [13]). SMARTS used affinity information to enhance memory locality by scheduling related tasks “back to back” at run time [47]. Kohn and Baden have used affinity in co-locating structured adaptive meshes in order to reduce communication costs [30]. Others have proposed application level schedulers [12].

In addition to performance, real time concerns are also at issue. For instance, it is important to ensure that information flowing between the living tissue and the simulations are processed in a timely manner. Flexible scheduling enables Tarragon to meet these requirements and its task graph representation readily accommodates both the MCell simulator as well as the clamping devices.

To support flexible scheduling we will apply a new technique called *parameterized scheduling*. Parameterized scheduling has the property that it can read attributes decorating the task graph to help guide the scheduler. These attributes come in the form of performance metadata [29], which can represent a variety of quantities, e.g. affinity, priority or other metrics. The Tarragon programmer is free to interpret the meaning of the metadata, while the scheduler examines their relative magnitudes in order to make scheduling decisions. The flexibility offered by parameterized scheduling significantly enhances the ability to explore alternative scheduling policies and metrics.

4.4 Implementation

To support data driven execution, each processing module will run one or more Mover-Dispatcher threads to coordinate communication and scheduling. The Mover-Dispatcher listens for data motion activity and runs concurrently with computation. It routes outgoing data to other tasks as specified in the TaskGraph and also senses the arrival of data coming from other tasks. The arrival of incoming data causes the Mover-Dispatcher to enable a suspended task for execution. Although the Mover-Dispatcher consumes resources, past work with communication proxies revealed the cost to be reasonable so long as the proxy does not utilize the processor as much as the computational threads [5,18,6,7]. We expect this to be the case of the present application.

Some aspects of Tarragon are similar to those of the Charm++ run time system[40,27]. Like Charm, work is "overdecomposed" onto processing nodes, that is, tasks are assigned many-to-one to processing modules. However, there are some important differences. Charm++ supports shared objects, and asynchronous remote method invocation on general C++ objects. Tarragon exposes a different API to the programmer. There are no shared objects, and methods may be invoked only locally. Data must be moved explicitly and before a method may be applied to it. The Tarragon philosophy is to expose communication, which is assumed to be an expensive operation. DMCS [15] has some similarities to Tarragon. It supports single sided communication and active messages.

5. DISCUSSION AND CONCLUSIONS

An experimentation environment has been described for coupling large-scale numerical simulations of neural microphysiology to living neurons *in-vitro*. The environment is coordinated asynchronously using a run time library called Tarragon. Tarragon supports data driven execution. It masks communication latency and balances workloads automatically. Tarragon makes two contributions: (1) parameterized scheduling, which includes performance meta data used to guide scheduling decisions, and (2) a uniform model for expressing asynchronous parallelism involving a mixture of physical devices on a "wet lab" work bench and hierarchically organized computational resources coupled over local area networks. Tarragon separates the concerns surrounding policy from decisions affecting performance, i.e. scheduling, from the expression of a correct algorithm. It therefore supports the implementation of highly scalable cell physiology simulators that offer performance and coding advantages compared with simulators implemented under bulk synchronous parallelism, thereby enabling new capabilities for making scientific discovery.

ACKNOWLEDGMENTS

Greg Balls and Scott Baden are supported by NSF contract ACI-0326013 and by the National Partnership for Advanced Computational Infrastructure (NPACI) under NSF contract ACI9619020. Tom Bartol and Terry Sejnowski are supported by NSF NPACI ACI9619020, NSF IBN-9985964, and the Howard Hughes Medical

Institute. The MCell-K website is accessible via the world wide web at the following URL: http://www-cse.ucsd.edu/groups/hpcl/scg/.

REFERENCES

1. Arvimd, Executing a program on the mit tagged-token dataflow architecture, *IEEE Trans. Computers.*, **39**(1990), 300-318
2. L. Anglister, J. R. Stiles, and M. M. Salpeter. Acetylcholinesterase density and turnover number at frog neuromuscular junctions, with modelling of their role in synaptic function. *Neuron*, **1**(1994),783–94
3. R. G. Babb, Parallel processing with large-grain data flow technique. *Computer*, **17**(1984), 55–61
4. S. B. Baden. Programming abstractions for dynamically partitioning and coordinating localized scientific calculations running on multiprocessors. *SIAM J. Scientific and Statistical Computing.*, **12**(1991):145–157
5. S. B. Baden and S. J. Fink. Communication overlap in multi-tier parallel algorithms, *Proc. SC '98*, (1998).
6. S. B. Baden and S. J. Fink. A programming methodology for dual-tier multicomputers. *IEEE Trans. Software Engineering*, **26**(2000), 212–26
7. S. B. Baden and D. Shalit. Performance tradeoffs in multi-tier formulation of a finite difference method. *Proc. 2001 International Conf. on Computational Science*, (2001).
8. G. T. Balls, S. B. Baden, T. Kispersky, T. M. Bartol, and T. J. Sejnowski. A large scale monte carlo simulator for cellular microphysiology.*Proc.18th Intl. Parallel Distributed Proessing. Symp.*, (2004).
9. O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Bandwidth-centric allocation of Independent tasks on heterogeneous platforms. *Proc. 16th Intl. Parallel Distributed. Processing Symp.*, (2002)
10. M. J. Berger and P. Colella. Local adaptive mesh refinement for shock hydrodynamics. *J. Computational Physics.*, **82**(1989), 64–84
11. M. J. Berger and J. Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *J. Computational Physics*, **53**(1984), 484–512
12. F. Berman. High-performance schedulers. In I. Foster and C. Kesselman (Eds.) *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.
13. R. Chandra, A. Gupta, and J. L. Hennessy. Data locality and load balancing in cool. *ACM SIGPLAN 1993 Symp. on Principles and Practice of Parallel Programming*. (1993), 249-259.
14. M. K. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communcations ACM*, **24**(1981), 198–206
15. N. Chrisochoides, K. Barker, J. Dobbelaere, D. Nave, and K. Pingali. Data movement and control substrate for parallel adaptive applications. *Concurrency Practice and Experience*, **14**(2002), 77-101
16. J. Dennis. Data flow supercomputers. *IEEE Computer*, **13**(1980), 48-56.
17. K. Devine, J. Flaherty, R. Loy, and S. Wheat. Parallel partitioning strategies for the adaptive solution of conservation laws. In I. Babuška et al. (Eds.), *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, Springer-Verlag, Berlin, **75**(1995), 215-242
18. S. J. Fink. *Hierarchical Programming for Block-Structured Scientific Calculations*. PhD thesis, Department of Computer Science and Engineering, University of California, San Diego (1998)
19. S. J. Fink, S. B. Baden, and S. R. Kohn. Efficient run-time support for irregular block-structured applications. *J. Parallel Distributed. Computing.*, **50**(1998), 61-82
20. 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, *Appied. Numerical Math..*, 26(1998, 241-263
21. R. Fujimoto. *Parallel and Distributed Simulation Systems*. Wiley, (2000)
22 A. S. Grimshaw, J. B. Weissman, and W. T. Strayer. Portable run-time support for dynamic object-oriented parallel processing. *ACM Transactions on Computer Systems*, **14**(1996),139-170
23. M. L. Hines and N. T. Carnevale. The NEURON simulation environment. *Neural Computation* **9**(1997),1179-1209
24. R. Jagannathan. Coarse-grain dataflow programming of conventional parallel computers, in L. Bic, J.-L. Gaudiot, and G. Gao (Eds.), *Advanced Topics in Dataflow Computing and Multithreading*, IEEE Computer Society Press, 1995, 113-129.

25. C. Johnson, S. Parker, and D. Weinstein. Large-scale computational science applications using the scirun problem solving environment. *Proc. Supercomputing* 2000 (2000).
26. J.R. Gurd, et al. The Manchester prototype dataflow computer. *Communications ACM*, **28**(1985), 34-52
27. L. V. Kalé. The virtualization model of parallel programming, *Runtime optimizations and the state of art. LACSI* (2002).
28. H. Kasahara and A. Yoshida. A data-localization compilation scheme using partial-static task assignment for fortran coarse-grain parallel processing. *Parallel Computing*, **24**(1998), 579-596
29. P. Kelly, O. Beckmann, A. Field, and S. Baden. Themis, Component dependence metadata in adaptive parallel applications. *Parallel Processing Letters*, **11**(2001), 455-470
30. S. R. Kohn and S. B. Baden. A parallel software infrastructure for structured adaptive mesh methods. *Proc. Supercomputing 1995*, (1995).
31. Z. Lan, V. E. Taylor, and G. Bryan. Dynamic load balancing of samr applications on distributed systems, *Proc. SC '01* (2001).
32. C. Leiserson, K. Randall, and Y. Zhou. Cilk, An efficient multithreaded runtime system. *Proc. Fifth ACM SIGPLAN Symp.Principles and Practice of Parallel Programming* (1995). 207-216
33. B.-H. Lim, P. Heidelberger, P. Pattnaik, and M. Snir. Message proxies for efficient, protected communication on smp clusters. *Proc. Third International. Symp. on High-Performance Computer Architecture*, (1997), 116-27
34. B. D. Lubachevsky. Efficient parallel simulations of asynchronous cellular arrays. *Complex Systems* **1**(1987), 1099-1123
35. E. P. Markatos and T. LeBlanc. Load balancing versus locality management in shared-memory multiprocessors. *Proc. Int'l. Conf. on Parallel Processing* (1992)
36. J. McCorquodale, D. de St. Germain, S. Parker, and C. Johnson. The untah parallelism infrastructure, A performance evaluation. *High Performance Computing*, Seattle WA (2001)
37. C.-W. Ou and S. Ranka. Parallel incremental graph partitioning. *IEEE Trans. Parallel Distributed Systems*, **8**(1997), 884-896
38. M. Parashar and J. C. Browne. Systems engineering issues in the implementation of an infrastructure for parallel structured adaptive meshes. In S. B. Baden, N. Chrisochoides, M. Norman, and D. Gannon (Eds.) *Proc. Workshop on Structured Adaptive Mesh Refinement Grid Methods*, Springer-Verlag, Berlin, 1998.
39. M. Parashar and I. Yotov. An environment for parallel multi-block, multi-resolution reservoir simulations. *Proc. 11th Intl. Conf. Parallel and Distributed Computing Systems,*. (1998), 230-235
40. J. C. Phillips, G. Zheng, S. Kumar, and L. V. V. Kalé. NAMD, Biomolecular simulation on thousands of processors. *Proc SC* 2002(2002)
41. J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with spacefilling curves. *IEEE Transactions Parallel and Distributed Systems*, **7**(1996), 288-300
42. 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 (Ed.), *Computational Neuroscience*, (1998), 279-284
43. 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 (Eds) *Synapses*. Johns Hopkins University Press, 2001
44. 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** (1999). 1177-1187
45. 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 Academy Sciences USA*, **93**(1996), 5747-5752
46. J. D. Teresco, M. W. Beall, J. E. Flaherty, and M. S. Shephard. A hierarchical partition model for adaptive finite element computation. *Computational. Methods. Applied. Mechanical. Engeering.*, **184**(2000), 269-285
47. S. Vajracharya, S. Karmesin, P. Beckman, J. Crotinger, A. Malony, S. Shende, R. Oldehoeft, and S. Smith. Smarts, Exploting temporal locality and parallelism through vertical execution. *International Conference on Supercomputing*, (1999).