Parallel Discrete Event Simulations of Grid-based Models: Asynchronous Electromagnetic Hybrid Code*

Homa Karimabadi[1], Jonathan Driscoll[1], Jagrut Dave[1,2], Yuri Omelchenko[1], Kalyan Perumalla[2], Richard Fujimoto[2], and Nick Omidi[1]

[1] SciberNet, Inc., Solana Beach, CA, 92075, USA
{homak,driscoll,yurio,omidi}@scibernet.com
[2] Georgia Institute of Technology, Atlanta, GA, 30332, USA
{jagrut,kalyan,fujimoto}@cc.gatech.edu

Abstract. The traditional technique to simulate physical systems modeled by partial differential equations is by means of a time-stepped methodology where the state of the system is updated at regular discrete time intervals. This method has inherent inefficiencies. Recently, we proposed [1] a new asynchronous formulation based on a discrete-event-driven (as opposed to time-driven) approach, where the state of the simulation is updated on a "need-to-be-done-only" basis. Using a serial electrostatic implementation, we obtained more than two orders of magnitude speedup compared with traditional techniques. Here we examine issues related to the parallel extension of this technique and discuss several different parallel strategies. In particular, we present in some detail a newly developed discrete-event based parallel electromagnetic hybrid code and its performance using conservative synchronization on a cluster computer. These initial performance results are encouraging in that they demonstrate very good parallel speedup for large-scale simulation computations containing tens of thousands of cells, though overheads for inter-processor communication remain a challenge for smaller computations.

1 Introduction

Computer simulations of many important complex physical systems have reached a barrier as existing techniques are ill-equipped to deal with the multi-physics, multi-scale nature of such systems. An example is the solar wind interaction with the Earth's magnetosphere. This interaction leads to a highly inhomogeneous system consisting of discontinuities and boundaries and involves coupled processes operating over spatial and temporal scales spanning several orders of magnitude. Inclusion of such disparate scales is beyond the scope of existing codes [2].

* Research was supported by NSF ITR grant 0325046 at SciberNet Inc. and 0326431 at Georgia Institute of Technology. Some of the computations were performed at the San Diego Supercomputing Center.

We have taken a new approach [1] to the simulation of such complex systems. The conventional time-stepped grid-based Particle-In-Cell (PIC) models provide the sequential execution of synchronous (time-driven) field and particle updates. In a synchronous simulation, the distributed field cells and particles undergo simultaneous state transitions at regular discrete time intervals. In contrast, we propose a new, asynchronous type of PIC simulation based on a discrete-event-driven (as opposed to time-driven) approach, where particle and field time updates are carried out in space on a "need-to-be-done-only" basis. In these simulations, particle and field information "events" are queued and continuously executed in time. The technique has some similarity to Cellular Automata (CA) in that complex behaviors result from interaction of adjacent cells [3]. However, unlike CA, the interactions between cells are governed by a full set of partial differential equations rather than the simple rules as are typically used in CA. The power of this technique is in its asynchronous nature as well as elimination of unnecessary computations in regions where there is no significant change in time. This is in contrast to CA, which are largely based on synchronous execution (e.g. [4]); to date, asynchronous parallel discrete event simulation of CA have only been applied to relatively simple phenomena such as Ising spin [5]).

Using a serial electrostatic model, we have shown [1] that the discrete event technique can lead to more than two orders of magnitude speedup compared to conventional techniques. In the following, we discuss issues associated with the extension of this technique to parallel architectures. We then demonstrate, through a newly developed parallel hybrid code, that parallel processing can provide an additional order of magnitude improvement in performance.

2 Parallel Computation Issues

Discrete Event Simulation (DES) offers substantial benefits compared to conventional explicit time-driven simulation by reducing the amount of computation that must be performed. However, by itself, DES is not sufficient to achieve the desired performance and scalability. Parallel DES (PDES) can help address this issue. However, the irregular nature of PDES computations leads to difficulties. Synchronization overhead, the number of concurrent computations, load distribution and event processing rate impact PDES performance significantly [6].

As in conventional (time-driven) simulations, the parallelization of asynchronous (event-driven) continuous PIC models is realized by decomposing the global computation domain into subdomains. In each subdomain, individual cells and particles are aggregated into containers that may be mapped to different processors. The parallel execution of time-driven simulations is commonly achieved by copying field information from the inner lattice cells to ghost cells of neighboring subdomains and exchanging out-of-bounds particles between the processors at the end of each update cycle. By contrast, in parallel asynchronous PIC simulations both particle and field events are not synchronized by the global clock (i.e. they do not take place at the same time intervals throughout the simula-

tion domain), but occur at arbitrary time intervals, introducing synchronization problems. Unless precautions are taken, a process may receive an event message from a neighbor with a simulation time stamp that is in its past.

In the following, we assume that the parallel simulation is composed of a collection of Simulation Processes (SPs) that communicate by exchanging time stamped event messages. Broadly, synchronization approaches may be classified as *conservative* or *optimistic*. Conservative synchronization ensures that each simulation process never receives an event in its past [8, 9]. Runtime performance is critically dependent on *apriori* determination of an application property called *lookahead (a time interval)*, which is roughly dependent on the degree to which the computation can predict future interactions with other processes without global information. On the other hand, the optimistic approach allows a process to receive a message in its past, but uses a rollback mechanism to recover [7]. Further discussion can be found in [10,11,12].

Another important issue concerns load balancing. As with any parallel or distributed application, the computation must be evenly balanced across processors and interprocessor communication should be minimized to achieve the best performance. Often these are conflicting goals. This is particularly challenging in PDES because of its irregular, unpredictable nature. Load balancing can greatly affect the efficiency of synchronization mechanisms (e.g. poor load distribution can lead to excessive rollbacks in optimistic systems). Automated schemes that balance workload at runtime using process migration present new challenges in this area [13,15].

Finally, it is desirable to decouple the parallel simulation engine that handles synchronization and communication from the application/models. This reduces the burden of the application developer, by not requiring an understanding of underlying PDES synchronization mechanisms. We have used an extensible simulation engine that provides multiple synchronization and event delivery mechanisms through a single interface, named μsik [16].

3 DES Model

We have developed a general architecture for parallel discrete event modeling of grid-based models. Details will be presented elsewhere. Here we present a simplified version of our technique that illustrates the salient features of our model without getting bogged down in all the details. In the following, we show results from a 1D parallel hybrid code (*light* version) that we have developed and tested using μsik as the simulation engine. This code is used to highlight unique parallel issues that are encountered in the DES modeling of plasmas. The light version does not strictly conserve flux. However, the lack of strict local flux conservation does not change the result significantly in the problem of interest here.

3.1 Hybrid Algorithm

Electromagnetic hybrid algorithms with fluid electrons and kinetic ions are ideally suited for physical phenomena that occur on ion time and spatial scales. Maxwell's equations are solved by neglecting the displacement current in Ampere's law (Darwin approximation), and by explicitly assuming charge neutrality. There are several variations of electromagnetic hybrid algorithms with fluid electrons and kinetic ions [18]. Here we use the one-dimensional resistive formulation [19] which casts field equations in terms of vector potential. The model problem uses the piston method where incoming plasma moving with flow speed larger than its thermal speed is reflected off the piston located on the rightmost boundary. This leads to the generation of a shockwave that propagates to the left. In this example, we use a flow speed large enough to form a fast magnetosonic shock. In all the runs shown here, the plasma is injected with a velocity of 1.0 (normalized to upstream Alfven speed), the background magnetic field is tilted at an angle of 30°, and the ion and electron betas are set to 0.1.

The simulation domain is divided into cells [1], and the ions are uniformly loaded into each cell. We conducted experiments ranging from 4,096 to 65,536 cells, and initialized each simulation to have 100 ions per cell. Each cell is modeled as an SP in μsik and the state of each SP includes the cell's field variables. The main tasks in the simulation are to a) initialize fields, b) initialize particles, c) calculate the exit time of each particle, d) sort IonQ (see below), e) push particle, f) update fields, g) recalculate exit time, and h) reschedule. This is accomplished through a combination of priority queues and three main classes of events. The ions are stored in either one of two priority queues as illustrated in Fig. 1. Ions are initialized within cells in an IonQ. As ions move out of the left most cell, new ions are injected into that cell in order to keep the flux of incoming ions fixed at the left boundary. MoveTime is the time at which an ion is to be moved next. The placement and removal of ions in IonQ and PendQ is controlled by comparing their MoveTimes to the current time and lookahead. Ions with MoveTimes more than current time + 2*lookahead have not yet been scheduled and are kept in the IonQ. A wakeup occurs when the fields in a given cell change by more than a certain threshold and MoveTimes of particles in the cell need to be updated. On a wakeup, only the ions in this queue recalculate their MoveTimes. Because ions in the IonQ have not yet been scheduled, a wakeup requires no event retractions. If an ion's MoveTime becomes less than current time + 2*lookahead in the future, the ion is scheduled to move, and is removed from the IonQ and placed in the PendQ. Thus, the front of the IonQ is at least one lookahead period ahead of the current time. This guarantees that each ion move will be scheduled at least one lookahead period in advance. The PendQ is used to keep track of ions that have already been scheduled to exit, but have not yet left the cell. These particles have MoveTimes that are less than the current time. Ions in the PendQ with MoveTimes earlier than the current time have already left the cell and must be removed before cell values such as density and temperature are calculated.

Events can happen at any simulation time and are managed separately by individual cells of the simulation. The flow of the program including functions of

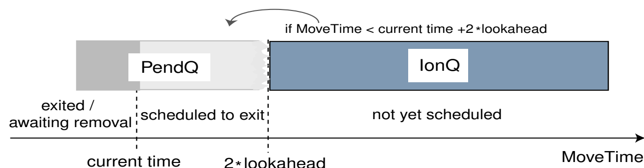

Fig. 1. At any moment, the ions in a cell are stored in one of two queues, the IonQ and the PendQ. Both are priority queues, sorted so that the ion with the earliest exit time is at the top.

events and their interaction with μsik is illustrated in Fig. 2. In this simulation, each cell handles three different types of events.

SendIon Event: This event is first run on each cell when the simulation is initialized, and is responsible for sending ions from one cell to the next. This is accomplished by scheduling the complementary "AddIon" events for neighboring cells. The SendIon event schedules an AddIon event corresponding to every Ion which exits within two lookahead periods, and always schedules at least one SendIon event. In addition, the SendIon Event checks to see if the fields have changed by some tolerance, waking up particles in that cell if necessary. SendIon events occur frequently and as a whole are computationally significant.

AddIon Event: This event is used to add a single ion to a cell. The ion's new exit time is calculated, and then it is added to the IonQ. The fields in the cell are then updated and Notify Events are scheduled for the left and right neighbor cells to inform them of the field change. The AddIon Event causes state changes and occurs sporadically in large batches.

NotifyEvent: This event updates the vector potential and temperature for the two neighbors.

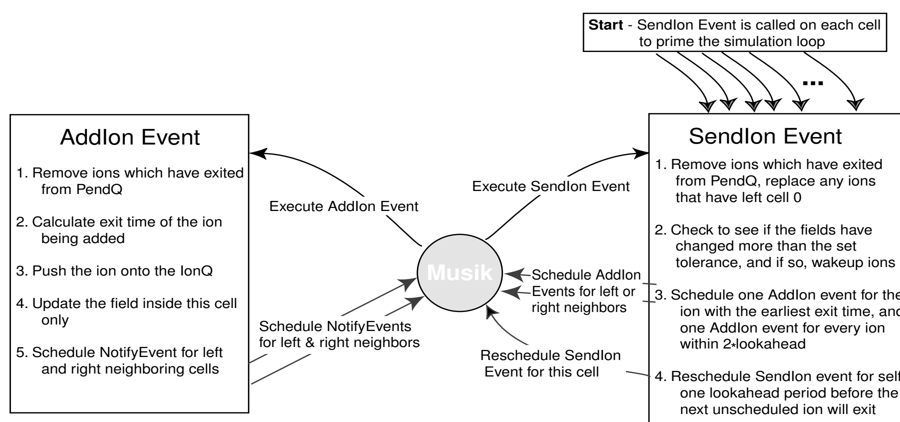

Fig. 2. Flow diagram of the parallel hybrid code.

Exit Time. We take the electric and magnetic fields to be constant within a cell, with arbitrary orientation and magnitude. In this case, a charged particle will have an equation of motion that can be calculated analytically and has the general form $\mathbf{R}(t) = \mathbf{A}t^2 + \mathbf{B}t + \mathbf{r}_c \sin(\omega_c t + \boldsymbol{\phi}) + \mathbf{C}$, where $R(t)$is the position of the particle. Newton's method is used to solve for the exact exit time.

Lookahead. If the typical velocity of a particle is v, and a typical cell width is x, then the time it takes for a particle to cross a cell is x/v. Lookahead must be a factor smaller than this time so that a particle covers a small fraction of the cell width in one lookahead period. On the other hand, if the lookahead is too small, the parallel performance will be poor. This happens when there are few event computations during a lookahead period. Synchronization overhead becomes larger than the computational load. We use the time it takes for the first particle to exit a cell to set the lookahead.

4 Results

Figure 3(a) compares results of traditional time-stepped hybrid simulation and our event-stepped simulation for a single processor. We have plotted the y and z (transverse) components of the magnetic field, the total magnetic field, and the plasma density versus x, after the shock wave has separated from the piston on the right hand side. The match between the two simulations is remarkable as DES captures the (i) correct shock wave speed, and (ii) details of the wavetrain associated with the shock wave. This match is impressive considering the fact that the differences seen in Fig. 3(a) are within statistical fluctuations associated with changes in the noise level in hybrid codes.

4.1 Effect of Lookahead

Next, we consider the effects of changing the lookahead on both the accuracy of the results as well as the execution time. The hardware for the runs shown here was a high-performance cluster at the Georgia Tech High Performance Computing laboratory. The cluster has 8 nodes, each with 4 2.8 GHz Xeon processors and 4 GB of RAM. The simulation uses 4 μsik Federates (one per processor), each with 2 Regions and 512 cells per Region. A Region is a grouping of cells for efficient load-distribution, described in Sect. 4.3.

Figure 3(b) shows variations in the spatial profile of B_{tot}, with lookahead. The zero lookahead run yields the most accurate result and is treated as a baseline. In the hybrid algorithm, the maximum lookahead must be less than the exit time of the earliest scheduled particle, which is approximately 0.15 for our choice of parameters. Deviations of the profile from the baseline are less than 10%, even when the maximum lookahead value is used.

Figure 4(a) shows the speedup in execution time relative to the zero lookahead run. The important point from this figure is that even small departures from zero lookahead lead to substantial improvements in speed. In fact, the most

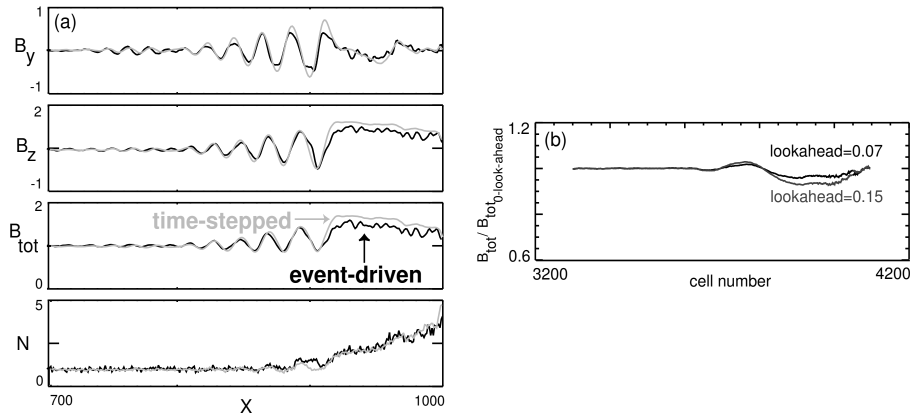

Fig. 3. (a) Comparison of time-stepped and event-driven simulations of a fast magnetosonic shock. (b) The ratio of the total magnetic field from lookahead runs of 0.07 and 0.15 relative to the field from zero lookahead run.

dramatic speedup (a factor of 3) is achieved when lookahead is changed from 0 to 0.005. Further changes in lookahead do improve performance, but at a much slower rate. For example, increasing the lookahead by an order of magnitude from 0.005 to 0.05 leads to only an additional 15% speedup.

4.2 Scaling with the Number of Processors

The 1D modeling of the shock problem in Fig. 3(a) can be easily performed with a serial version of our code. However, our ultimate goal is to develop a 3D version of the code. As a simple means to evaluating the parallel execution in a 3D model, we have considered cell numbers as large as 65,536. This is sufficient to identify the key issues of parallel execution. Figure 4(b) shows the speedup as a function of the number of processors up to 128. The speedup is measured with respect to a sequential run. These runs were made on a 17 node cluster, with each node having 8 550 MHz CPUs and 4 GB of RAM. The simulation domain consisted of 8,192 cells in one case and 65,536 cells in the other.

As is evident from Fig. 4(b), the parallel speedup is good till eight processors, but declines as more processors are added. This is due to the architecture of the cluster, which uses a collection of 8 processor computers communicating through TCP/IP. With up to 8 processors, the entire simulation runs through shared memory, and the communication overheads are low. However, with more than 8 processors, the overheads associated with TCP/IP begin to offset the speed gained by using more processors. This reduces the slope of the curve. For 8,192 cells, the speedup does not increase significantly after 32 processors. This is because of increased synchronization costs, which negate the gains from parallel processing. Processors do not have a sufficient computational load between global synchronizations and spend a greater fraction of time waiting on

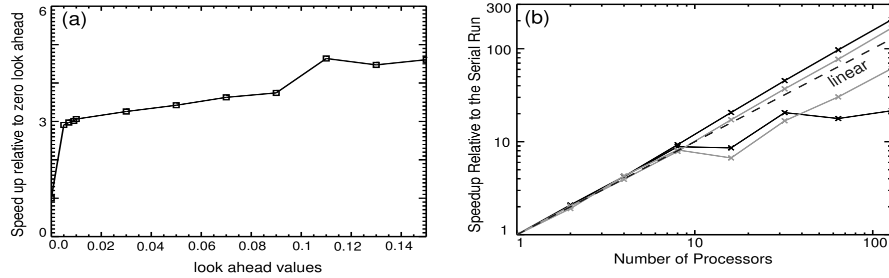

Fig. 4. (a) Speedup with increased lookahead. (b) Scaling with the number of processors. The dashed line is a linear scaling curve. Speedup for 8,192 cells is in black and 65,536 cells is in gray. For each domain size, there are two curves - speedup considering the overall execution time and speedup without considering communication time. The scaling is superlinear if communication time is removed.

other processors. For 65,536 cells, there is enough computation between global synchronizations to obtain good speedup up to 128 processors.

Since the overheads associated with inter-processor communication become relatively smaller as the simulation size increases, we do not anticipate this effect being as pronounced with larger, 3D simulations. In 3D simulations, each processor would have several orders of magnitude more cells, making the relative overheads associated with TCP/IP much less. To test the idea that poor scaling is the result of the communication overheads, we have also plotted speedup with the communication time subtracted out in Fig. 4(b). The speedup in this case is better than expected, and is in fact super-linear. In other words, doubling the number of processors more than doubles the execution speed. This is the result of a peculiar feature of DES. Execution time does not necessarily scale linearly with the simulation size, even on one processor. This is in part due to the fact that as the event queue becomes larger (contains more pending events), the time associated with scheduling and retrieving each event increases. So, as the simulation gets distributed over more and more processors, each processor is effectively dealing with a smaller piece of the simulation, making the scaling non-linear. Memory performance (specifically, cache performance) can also lead to super-linear speedup. By keeping the same size problem but distributing it over more processors, the memory footprint in each processor shrinks. The total amount of cache memory increases in proportion with the number of processors used - with enough processors, one can, for example, fit the entire computation into the processors' caches. An extreme case of this is when the problem is so large that it does not fit into the memory of a single machine, causing excessive paging. Although both effects could be causing the super-linear scaling seen in Fig. 4(b), the data structure performance appears to be dominant. In a no-load test of μsik, we changed the number of cells from ten to a million. The time to process a single event increased from 6.50 to 22.15 microseconds, indicating a scaling behavior of NlogN.

Figure 5(a) shows the percentage of time spent in communication and blocking in each case. There is a significant increase in the fraction of time spent in communication and blocking for more than 8 processors. For 65,536 cells, the percentage settles to around 60% for higher number of processors. However, for 8,192 cells, the percentage keeps on increasing until 90% for 128 processors.

4.3 Load Balancing

Figure 5(b) shows the variation in execution time as a function of the number of Regions per processor, as distributed by the Region Deal algorithm. In this scheme, the simulation is broken into small Regions which are then "dealt" out much like a card game among processors. These simulation runs were performed on the first cluster mentioned earlier.

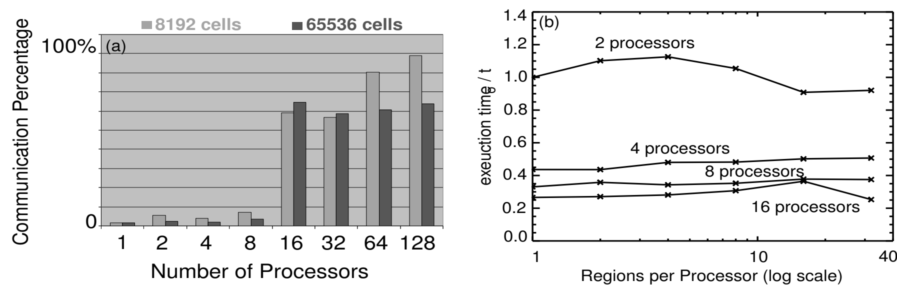

Fig. 5. (a) Percentage of time spent in communication. (b) Performance of load balancing algorithm.

The curves show a different trend for higher number of processors (4,8,16) than for 2 processors. For higher number of processors, the variation in execution time because of the Region Deal load balancing scheme is less pronounced. The best execution times are close to the execution time for 1 Region per processor, with variations of less than 1,000 seconds. Also, increasing the number of Regions per processor increases the execution time in most cases. This is because of the increased synchronization overhead that negates the benefits of the load distribution scheme. For 2 processors, having more Regions per processor leads to better load distribution and hence reduced execution time. The execution time settles around 14,000 seconds for 16 Regions or more. In this case too, contiguous cells are assigned to different processors and incur greater synchronization overhead for higher number of Regions per processor.

References

1. Karimabadi, H, Driscoll, J, Omelchenko, Y.A. and N. Omidi, *A New Asynchronous Methodology for Modeling of Physical Systems: Breaking the Curse of Courant Condition, J. Computational Physics,* (2005), in press.

2. Karimabadi, H. and N. Omidi. *Latest Advances in Hybrid Codes and their Application to Global Magnetospheric Simulations.* in *GEM, http://www-ssc.igpp.ucla.edu/gem/tutorial/index.html)* (2002).
3. Ilachinski, A., Cellular Automata, A Discrete Universe, World Scientific, 2002.
4. Smith, L., R. Beckman, et al. (1995). TRANSIMS: Transportation Analysis and Simulation System. Proceedings of the Fifth National Conference on Transportation Planning Methods. Seattle, Washington, Transportation Research Board.
5. Lubachevsky, B. D. (1989). Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks. Communications of the ACM **32**(1): 111-123.
6. Fujimoto, R.M., *Parallel and Distributed Simulation Systems.* (2000): Wiley Interscience.
7. Jefferson, D., *Virtual Time,* ACM Transactions on Programming Languages and Systems, (1985), 7(3):pp. 404-425.
8. Chandy, K. and J. Misra (1979). Distributed Simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering.
9. Chandy, K. and J. Misra (1981). Asynchronous distributed simulation via a sequence of parallel computations. Communications of the ACM. **24.**
10. Fujimoto, R. M. (1999), Exploiting Temporal Uncertainty in Parallel and Distributed Simulations, Proceedings of the 13th Workshop on Parallel and Distributed Simulation: 46-53.
11. Rao, D. M., N. V. Thondugulam, et al. (1998). Unsynchronized Parallel Discrete Event Simulation. Proceedings of the Winter Simulation Conference**:** 1563-1570.
12. Rajaei, H., R. Ayani, et al. (1993). The Local Time Warp Approach to Parallel Simulation. Proceedings of the 7th Workshop on Parallel and Distributed Simulation**:** 119-126.
13. Boukerche, A., and S. K. Das, Dynamic Load Balancing Strategies for Conservative Parallel Simulations, Workshop on Parallel and Distributed Simulation, 1997.
14. Carothers, C. D., and R. M. Fujimoto, “Efficient Execution of Time Warp Programs on Heterogeneous, NOW Platforms,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, No. 3, pp. 299-317, March 2000.
15. Gan,B. P., *et al.*, Load balancing for conservative simulation on shared memory multiprocessor systems, Workshop on Parallel and Distributed Simulation, 2000
16. Perumalla, K.S., *μsik – A Micro-Kernel for Parallel and Distributed Simulation Systems*, to appear in the Workshop on Principles of Advanced and Distributed Simulation, May, 2005.
17. Bagrodia, R., R. Meyer, et al. (1998). Parsec: A Parallel Simulation Environment for Complex Systems. IEEE Computer **31**(10): 77-85.
18. Karimabadi, H., D. Krauss-Varban, J. Huba, and H. X. Vu, On magnetic reconnection regimes and associated three-dimensional asymmetries: Hybrid, Hall-less hybrid, and Hall-MHD simulations, J. Geophys. Res., Vol. 109, A09205,(2004).
19. Winske, D. and N. Omidi, *Hybrid codes: Methods and Applications*, in *Computer Space Plasma Physics: Simulation Techniques and Software*, H. Matsumoto and Y. Omura, Editors. (1993), Terra Scientific Publishing Company. p. 103-160.