

Scalable Simulation of Electromagnetic Hybrid Codes

Kalyan Perumalla¹, Richard Fujimoto², Homa Karimabadi³

¹Oak Ridge National Laboratory, Oak Ridge, TN, USA

²Georgia Institute of Technology, Atlanta, GA, USA

³SciberQuest Inc, Solana Beach, CA, USA

Abstract. New discrete-event formulations of physics simulation models are emerging that can outperform models based on traditional time-stepped techniques. Detailed simulation of the Earth’s magnetosphere, for example, requires execution of sub-models that are at widely differing timescales. In contrast to time-stepped simulation which requires tightly coupled updates to entire system state at regular time intervals, the new discrete event simulation (DES) approaches help evolve the states of sub-models on relatively independent timescales. However, parallel execution of DES-based models raises challenges with respect to their scalability and performance. One of the key challenges is to improve the computation granularity to offset synchronization and communication overheads within and across processors. Our previous work was limited in scalability and runtime performance due to the parallelization challenges. Here we report on optimizations we performed on DES-based plasma simulation models to improve parallel performance. The mapping of model to simulation processes is optimized via aggregation techniques, and the parallel runtime engine is optimized for communication and memory efficiency. The net result is the capability to simulate hybrid particle-in-cell (PIC) models with over 2 billion ion particles using 512 processors on supercomputing platforms.

1. Introduction

New discrete-event approaches are being developed to speed up simulations of inhomogeneous physical systems in order to efficiently accommodate the variety of spatial and temporal scales in such systems. The new discrete-event approaches are being proposed in place of traditional time-stepped approaches in order to overcome the worst-case limitations imposed by the fastest processes in the system. Parallel execution of these discrete-event models is challenging due to a combination of their characteristics, including fine-grained event computation and dynamic inter-entity event communication patterns. In this paper, we document our optimizations to a discrete-event model of a one-dimensional hybrid shock simulation that uses a particle-in-cell method to simulate electromagnetic fields in a plasma environment. In our earlier work, we reported results from a preliminary parallel implementation on a cluster of workstations. The previous implementation uncovered several avenues for improvement, including computation granularity issues, memory usage requirements and inter-processor communication overheads. Our new implementation incorporates optimizations to the discrete event model implementation to enable efficient parallel/distributed execution, and enabled scaling it to supercomputing platforms.

The rest of the document is organized as follows. Section 2 provides the motivation and background to this work. The one-dimensional hybrid shock application is outlined in Section 3. The optimizations to the parallel implementation are described

in Section 4, followed by a parallel execution performance study in Section 5. Finally, Section 6 outlines the status and future work.

2. Background and Related Work

The conventional approach to realizing PIC models with spatial grid elements is to use time-stepped execution where the state of the model, e.g., particle position, velocity, charge, etc., is updated at fixed time increments. Discrete event simulation offers an alternative approach where particle and field updates are instead only carried out on an “as needed” basis, e.g., when field values cross certain thresholds, resulting in state updates at irregular (and less frequent) time points. The time interval between updates is therefore dictated by the predicted rate of change. Particle and field update “events” are used to denote when state updates occur. These events are queued and continuously processed over time to complete the simulation. Event-driven PIC simulations automatically guarantee that the progression of the system captures important state changes while reducing computation of less interesting, “idle” information. Further details of this approach are presented in [1, 2], where performance measurements were presented showing as much as two orders of magnitude speedup for certain PIC simulations.

Further increases in speed and scalability can be accomplished by applying parallel discrete event simulation (PDES) techniques. Here, the computation is divided into a collection of simulation processes that communicate by exchanging time stamped messages (events). A central question that must be addressed in PDES concerns ensuring proper synchronization of the computation. Unlike time-stepped simulations, PDES techniques allow some simulation processes to progress ahead of others in simulation time. This introduces the possibility of synchronization errors where a simulation process receives a message (event) with time stamp smaller than its current simulation time. Several approaches have been proposed to address this problem [3]. One class, termed conservative synchronization, blocks simulation processes to ensure no such synchronization errors occur [4, 5]. By contrast, optimistic synchronization techniques allow such errors to occur, but recover using a rollback mechanism [6].

PDES systems are typically composed of a simulation engine that handles issues such as synchronization, and invoking the simulation model entities at appropriate times. The μ sik system [7] used here is one example of a PDES simulation engine that can be configured to handle either conservative or optimistic synchronization methods. μ sik is based on a micro-kernel approach to parallel simulation engine design where fundamental mechanisms necessary for synchronization are implemented within the micro-kernel, and the rest of the kernel is built over the micro-kernel. Both conservative and optimistic parallel PIC simulations have been realized utilizing the μ sik system [1, 2, 8].

A limited amount of work has examined the application of PDES techniques to physical system simulation. Perhaps the earliest was the “colliding pucks” application developed for the Time Warp Operating System (TWOS) [9]. Lubachevsky discusses the use of conservative simulation protocols to create cellular automata models of Ising spin [10] and other physical system problems [11]. A formal approach to both discrete event and continuous simulation modeling based on

DEVS (Discrete Event System Specification), was proposed by Zeigler et al. [12] and some numerical solutions have been examined based on the DEVS formalism [13].

3. One-Dimensional Hybrid Shock Discrete Event Model

Here we provide a brief description of our DES model. Additional information can be found in [2]. 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 [14]. Here we use the one-dimensional (1-D) resistive formulation which casts field equations in terms of vector potential.

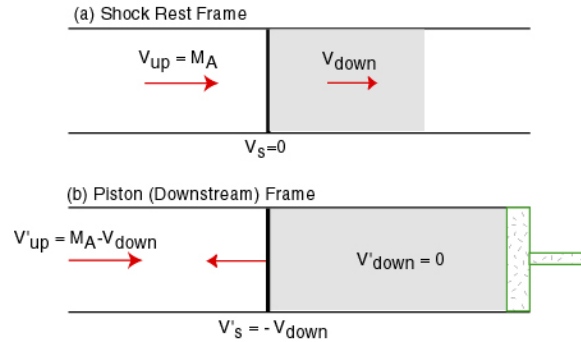


Figure 1: Simulation of a shock using the piston method

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, as illustrated Figure 1. M_A is the shock Mach number and V_{down} is the downstream flow velocity based on the Rankine-Hugoniot condition. This leads to the generation of a shockwave that propagates to the left in the piston frame of reference. 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 Alfvén 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. Each cell is modeled as a Logical Process (LP) in μsik and the state of each LP 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, (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.

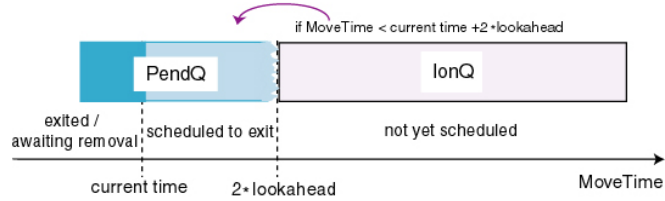


Figure 2: Organization of particle events in PendQ and IonQ priority queues

The ions are stored in either one of two priority queues[2], as shown in Figure 2. Ions are initialized within cells in an IonQ. As ions move out of the leftmost 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 moves to an adjacent cell. The placement and removal of ions in IonQ and PendQ is controlled by comparing their MoveTimes to the current time and lookahead (lookahead is the shortest delay between the current simulation time of the cell and the time of any event scheduled into the future by the cell). 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, ions in IonQ 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. 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 are 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.

4. Optimizations

As mentioned earlier, our preliminary implementation of a prototype for parallel execution of the 1-D hybrid shock model was limited in different ways. First, the per-event overhead incurred due to discrete event processing was found to be large due to the low granularity of event computation. Secondly, our parallel execution was constrained by sockets-based communication, which suffered from inefficiencies. Finally, the discrete event simulation engine itself was in an evolutionary state, and was consequently not optimized for memory usage. Our optimizations were aimed along these lines: The mapping from cells to simulation processes is changed to an aggregate scheme in order to minimize overheads. With communication subsystem optimizations, runtime performance has been significantly improved. Additionally, porting to a supercomputer enabled the simulation to scale up to 512 processors. By specializing the data structure to conservative synchronization (at runtime), the memory requirements to represent the cells and particles have been reduced. The largest configurations that can be simulated have been pushed significantly, to include over 2 billion ion particles overall. Some of these optimizations are described in

detail next.

4.1. Mapping Cells to DES Logical Processes

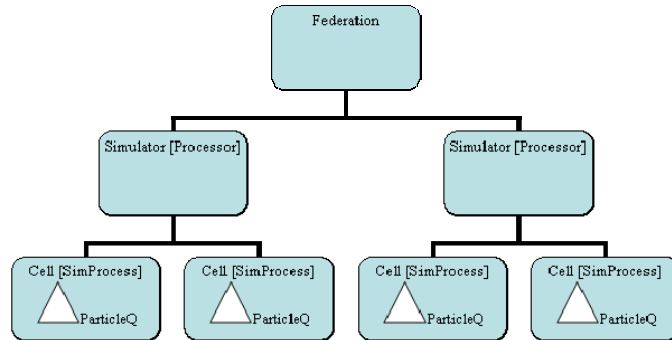


Figure 3: A suboptimal way of realizing a particle-in-cell DES model.

One way to realize PIC models, shown in Figure 3, is to map each cell to a logical process (LP). This provides maximum flexibility for load balancing, but makes every particle-transfer event to go through the (micro-kernel) PDES simulator, making it inefficient due to lack of optimization for locality of communication. Also, shared state is disallowed in this scheme, which makes it impossible for neighboring cells to exchange data via direct access to data structures.

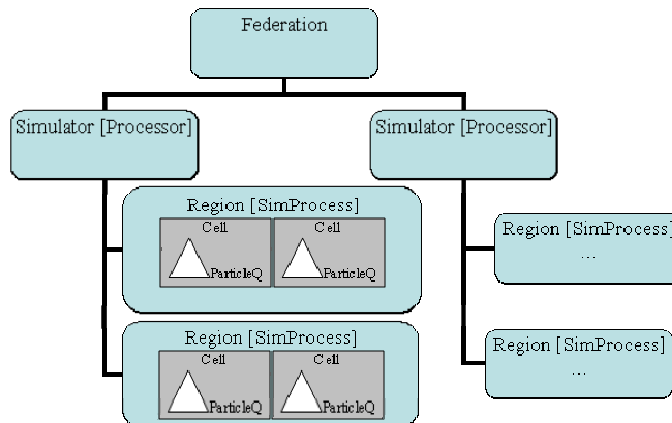


Figure 4: An efficient way of realizing a particle-in-cell DES model.

A more efficient alternative approach is shown in Figure 4. The concept of a “region” is introduced, which is an aggregate that contains multiple cells. Instead of mapping one *cell* per LP, each *region* is mapped to an LP. It results in memory savings, because the memory overheads of an LP are not incurred for every cell. Also, it is more natural to model: each region can be viewed as a sequential engine that simulates multiple cells. Particles crossing regions (i.e., across sequential engines) are sent as μ sik events across simulation processes (and, by natural implication, across processors). In our earlier work, we used the one-LP-per-Cell mapping scheme, which incurred overheads. We re-implemented the model with the new scheme based on multiple-cells-per-region, which significantly cut down event

scheduling and event processing overheads.

4.2. *Communication Subsystem*

Our earlier system used Berkeley sockets-based inter-processor communication. However, sockets have limited buffering capacities, which led to deadlocks on large-scale configurations due to the fact that large number of events (particle transfers, field updates) needed to be transferred across processors simultaneously. We have since then ported our engine to use high-performance communications based on native MPI implementations of the supercomputer platforms. Moving to MPI helped use large user-level buffers and avoid deadlocking while also improving the runtime performance considerably. The availability of control by the application on the size of the buffers helped us customize the communication based on the largest expected event message exchange rate in the application.

4.3. *μsik Engine Enhancements*

Since the PDES engine was designed to support both conservative as well as optimistic methods of synchronization in parallel execution, it was organized to accommodate the general case. However, the generality in the initial versions of the engine resulted in overheads of optimistic synchronization encroaching into conservative execution as well (e.g., the causal list maintenance among events, required for rollbacks, in the form of several pointer variables per event). This overhead is unnecessary in purely conservative execution, such as our 1-D hybrid simulation. The improvement here was to dynamically allocate space for event causal list pointers only upon first reference for the same within each event. This automatically ensures resilience to arbitrary combinations of optimistic and conservative logical processes. A 40% memory savings was realized by this dynamic allocation approach. Since every particle (ion) arrival or departure is represented as an event, this translated directly into increase in the number of particles that can be simulated in a given amount of memory.

5. Performance Study

We now turn to a study of scalability and runtime performance. All performance data reported here are collected on the San Diego Supercomputing Center's IBM DataStar supercomputer (www.sdsc.edu/user_services/datastar). The DataStar is a cluster of IBM P655 nodes, each node with 8 Power4 1.5GHz processors and 16GB memory (shared by the 8 processors). The nodes are connected by an IBM Federation Switch providing low latency and high bandwidth communication. The performance on up to 512 processors is shown in Figure 5. The observed performance is significantly better than previously reported, as a cumulative result of all the optimizations. Since the amount of concurrency is dependent on the simulated number of cells, we experimented with three configurations: small (150 cells/CPU), medium (1,500 cells/CPU) and large (40,000 cells/CPU). The total number of cells is scaled with the number of processors.

It is observed that the speedup with small configuration is less than that with medium-sized configuration. This is due to lack of enough concurrency with the smaller number of cells, making parallel synchronization overheads dominate. On the other

hand, we observe lower speedup with large-sized configuration than that with the medium-sized. This turns out to be due to the large amount of inter-processor event communication inherent in the larger run, imposing greater messaging overhead in the parallel run. To confirm this, we instrumented the code to obtain measures of inter-processor event types and their counts. It was observed that the number of “notify” events increases with the number of cells, which contributes significantly to the messaging overheads.

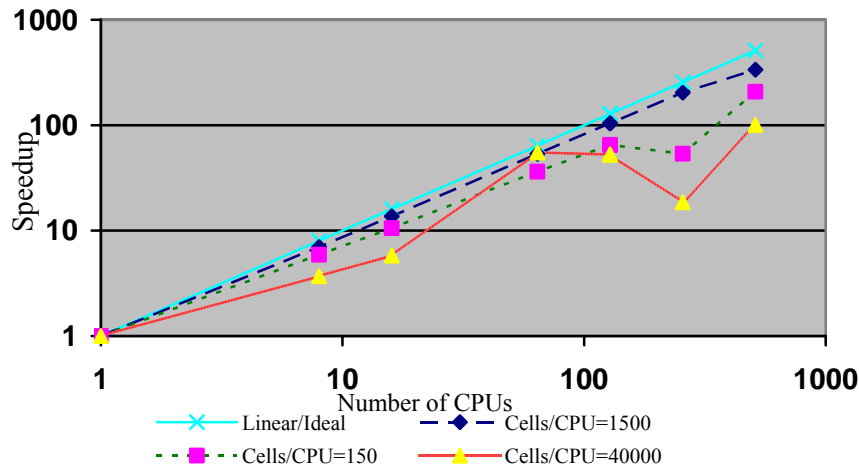


Figure 5: Runtime speedup of region-based Hybrid Shock code on varying no. of CPUs.

The next observation is on memory requirements. Each Ion takes approximately 150 bytes to be represented, and each cell has 100 ions. For 40,000 cells per CPU, the memory consumed to represent all the ions is $150\text{bytes/ion} \times 100\text{ions/cell} \times 40,000\text{cells/CPU} = 600\text{MB}$ per CPU. Also, MPI buffers at each CPU have to be allocated sufficiently large to prevent deadlocks. In a 2 billion ion simulation on 512 CPUs, with a conservative estimate of one million incoming ions into a CPU between synchronization steps, an MPI buffer of size $2\text{bloat}/\text{byte} \times 400\text{bytes}/\text{message} \times 10^6\text{messages} = 800\text{MB}$ is required at each CPU to avoid full buffers. The extra “bloat” factor of 2 on byte size is used to portably accommodate potential memory cost due to MPI pack/unpack data type conversions & representations. With these metrics, the number of particles has been increased linearly with the number of processors, reaching 20 million cells and over 2 billion ions in the largest execution using 40,000 cells per CPU on 512 CPUs.

Although this configuration is a bit large for the 1-D case, we are interested in observing the scaling properties of our system, with the goal of achieving efficient parallel execution for two-dimensional (2-D) and three-dimensional (3-D) versions as well. In a 2-D shock simulation, the number of cells and particles would be reasonable for some of the bigger runs. We in fact verified our expectation of similar performance on 2-D by observing a speedup of 194 on 256 processors and 248 on 512 processors on a configuration with 400 cells/CPU, and 10,000 ions/cell.

6. Status and Future Work

To our knowledge, the performance results reported here represent some of the largest executions of parallel discrete event-based physics simulation models. The techniques used here are fully extendable to multiple dimensions and non-uniform meshes. We are currently developing a uni-dimensional infrastructure with adaptive logical mapping capabilities. Our immediate application areas include global kinetic simulations of the Earth's magnetosphere and particle acceleration due to turbulence at fast magnetosonic shocks. Given the generality of the technique, however, we expect future applications to a wide variety of physics based simulations.

Acknowledgements

This work has been partly supported at Georgia Tech by NSF grant ATM-0326431 and by the NSF ITR Grant No. 0539106 at SciberQuest, Inc. The use of computing facilities at the San Diego Supercomputing Center is gratefully acknowledged.

References

- [1] H. Karimabadi, J. Driscoll, Y. Omelchenko and N. Omidi, "A New Asynchronous Methodology for Modeling of Physical Systems: Breaking the Curse of Courant Condition," *Journal of Computational Physics*, vol. 205(2), 2005.
- [2] H. Karimabadi, J. Driscoll, Y. Omelchenko, K. S. Perumalla, R. M. Fujimoto, and N. Omidi, "Parallel Discrete Event Simulation of Grid-based Models: Asynchronous Electromagnetic Hybrid Code," Springer LNCS Proceedings, pp. 580-588, 2005.
- [3] R. M. Fujimoto, *Parallel and Distributed Simulation Systems*: Wiley Interscience, 2000.
- [4] K. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," in *Communications of the ACM*, vol. 24, 1981.
- [5] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering*, vol. SE-5(5), pp. 440-452, 1978.
- [6] D. Jefferson, "Virtual Time," *ACM Transactions on Programming Languages and Systems*, vol. 7(3), pp. 404-425, 1985.
- [7] K. S. Perumalla, "μsik - A Micro-Kernel for Parallel/Distributed Simulation Systems," Workshop on Principles of Advanced and Distributed Simulation, 2005.
- [8] Y. Tang, K. S. Perumalla, R. M. Fujimoto, H. Karimabadi, J. Driscoll, & Y. Omelchenko, "Optimistic Parallel Discrete Event Simulations of Physical Systems using Reverse Computation," Workshop on Principles of Advanced and Distributed Simulation, 2005.
- [9] P. Hontalas, et al., "Performance of the Colliding Pucks Simulation on the Time Warp Operating System," Distributed Simulation, 1989.
- [10] B. D. Lubachevsky, "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks," *Communications of the ACM*, vol. 32(1), pp. 111-123, 1989.
- [11] B. Lubachevsky, "Several Unsolved Problems in Large-Scale Discrete Event Simulations," Workshop on Parallel and Distributed Simulation, 1993.
- [12] B. P. Zeigler, et al, *Theory of Modeling & Simulation*, 2nd ed: Academic Press, 2000.
- [13] J. Nutaro, "Parallel Discrete Event Simulation with Application to Continuous Systems," in *Department of Electrical and Computer Engineering*, vol. Ph.D. Tucson, AZ: University of Arizona, 2003, pp. 182.
- [14] H. 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," *Journal of Geophysical Research*, vol. 109, pp. 1-21, 2004.