

# A New Simulation Technique for Study of Collisionless Shocks: Self-Adaptive Simulations

H. Karimabadi<sup>1</sup>, Y. Omelchenko<sup>1</sup>, J. Driscoll<sup>1</sup>, R. Fujimoto<sup>2</sup>, K. Perumalla<sup>2</sup>, D. Krauss-Varban<sup>1</sup>

<sup>1</sup>SciberQuest, Inc., Solana Beach, CA, 92075, USA

<sup>2</sup>Georgia Institute of Technology, Atlanta, GA, 30332, USA

**Abstract.** The traditional technique for simulating physical systems modeled by partial differential equations is by means of time-stepping methodology where the state of the system is updated at regular discrete time intervals. This method has inherent inefficiencies. In contrast to this methodology, we have developed a new asynchronous type of simulation based on a discrete-event-driven (as opposed to time-driven) approach, where the simulation state is updated on a "need-to-be-done-only" basis. Here we report on this new technique, show an example of particle acceleration in a fast magnetosonic shockwave, and briefly discuss additional issues that we are addressing concerning algorithm development and parallel execution.

**Keywords:** Self-adaptive, simulations, particle-in-cell, hybrid, shock waves, particle acceleration

**PACS:** 52.65.Rr, 52.65.Ww, 96.50.-e, 96.50.Fm, 96.50.Pw

## INTRODUCTION

The strongly disparate temporal and spatial scales commonly occurring in many complex physical systems pose a significant computational challenge and necessitate a leap in simulation technology. Although the Adaptive Mesh Refinement (AMR) methodology is a powerful technique for addressing large variations in spatial scales, the time update of variables is still done through traditional time-stepping methodology (referred to here as standard time-stepping or STS). This has inherent inefficiencies and suffers from the usual Courant-Friedrichs-Levy (CFL) limitations. We have developed a novel simulation technique by abandoning time-stepping and replacing it with an event-driven method<sup>1-5</sup>. Event-driven simulations have their origins in operations research and management science, and more recently have found application in war games and telecommunications but have not been applied to complex physical systems. We have combined traditional mesh discretization techniques with a novel discrete-event methodology and developed several plasma and fluid simulation codes<sup>1-5</sup>. In our discrete-event simulation (DES) approach<sup>3-4</sup>, the traditional measure of time advance,  $\Delta t$  is replaced by the meaningful physical information unit,  $\Delta f$ . In effect, this technique introduces an individual adaptive time line for every computational entity, enabling truly *asynchronous* time integration of the system state variables. As a result, at any given time a DES model has to process

only changes to its global state that exceed the minimum information unit,  $\Delta f$ . This eliminates unnecessary computation in the inactive regions. Several parallel applications of this technology have been built, ranging from an electrostatic particle code<sup>1,5</sup> to an electromagnetic hybrid (electron fluid, particle ions) code<sup>2,4</sup> to a diffusion-convection equation solver<sup>3</sup>, and have demonstrated superior metrics:

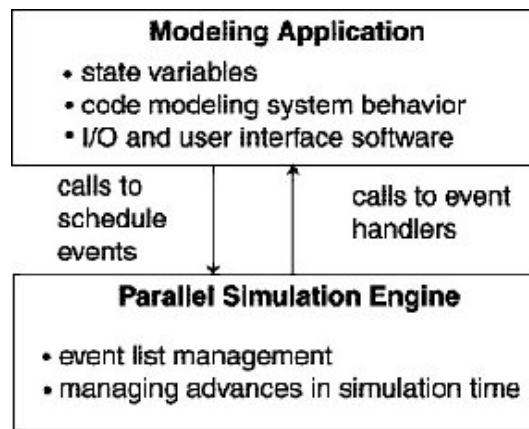
Faster: By eliminating unnecessary computations, speed-ups as large as a factor of 300 were achieved in one-dimension, with further enhancements being expected in 2D and 3D.

More Accurate: By updating the system state based on local  $\Delta f$ 's rather than the global timestep, the user has more effective control over the desired numerical accuracy.

Stable: The DES codes run successfully in regimes where standard codes are subject to explosive numerical instabilities.

## DES ALGORITHMS

As shown in Figure 1, a discrete-event simulation (DES) system can be broken into two components: (1) the models and (2) the parallel simulation executive that manages events and the progression of simulated time. Development of next generation plasma codes requires innovations in both components.



**FIGURE 1.** The two main components of a discrete event simulation.

Field equations are discretized in space in the conservation form. Each computational mesh cell is assigned discrete states associated with the temporal evolution of local field and particle quantities. Transitions from one temporal state to another are called "events". Time integration of each field component is delayed by a time interval that is computed based on the magnitude of its predicted rate of change. Particles are scheduled for advance in each cell based on their current velocities, local field magnitude and cell size. Each computation cell keeps a registry of increments to its

original state (the state used for the prediction) caused by the neighboring cells and reschedules events (time advances) to earlier times if the cell state is significantly altered during the predicted time delay. The DES code programming architecture is drastically different from conventional (time-driven) codes. In particular, each mesh cell has a means of polling its neighbors and fetching global simulation information using its local data handlers. It is also “aware” of its role in establishing communication with remote (distributed) parts of the system or applying proper boundary conditions. A nontrivial problem is to preserve fluxes across mesh cell interfaces. In explicit time-driven codes adjacent cells are always advanced with fluxes taken at the same time level. DES cells schedule themselves asynchronously and therefore special care is taken to ensure that field quantities in cells with common interfaces are always integrated in time with identical fluxes across the common boundaries<sup>3,4</sup>. The key distinguishing elements of our DES algorithm are:

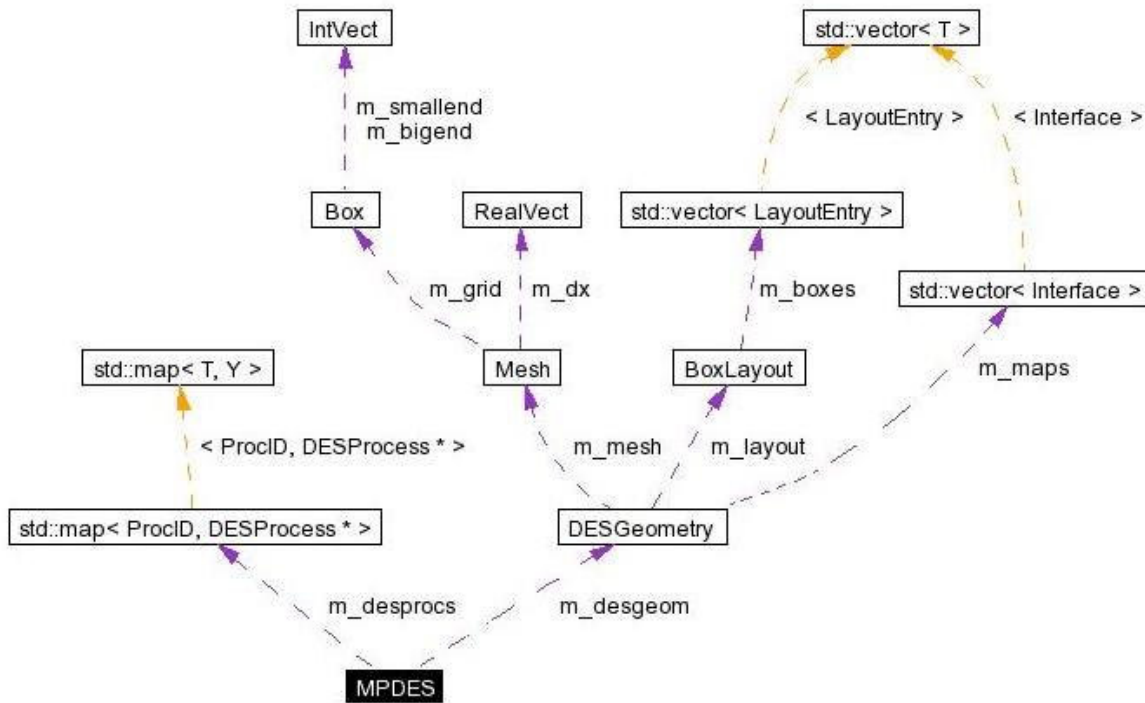
1. Each particle is advanced based on its own (time varying) time step.
2. Fields are updated *locally* and *only* when there is a change in that cell.
3. The solution is monitored and evolved based on the required accuracy rather than based on a pre-selected time-step size.

We have developed a library of C++ classes (SciDES) designed to provide a set of discrete-event software tools for implementing finite difference and particle-in-cell methods for the solution of coupled partial differential equations and equations of particle motion. SciDES standardizes fundamental data structures and algorithms for programming distributed time-dependent scientific models on block-structured computational domains and formalizes the most essential aspects of the distributed physics-based DES models in the form of a pseudo-distributed architecture. This pursues several goals. First, the SciDES API separates the computational physics algorithms from the communication issues by abstracting them into well defined concepts (C++ classes) and providing all the necessary "go-between" implementation details. Second, it fosters more efficient cooperation of computational physicists with computer scientists working on the distributed discrete-event engine algorithms since it allows substitution of pseudo-distributed plug-in modules by their Message Passing Interface (MPI) counterparts in a plug-and-play fashion without breaking the physics core of the code. In addition, the ability to run virtual distributed simulations on a single CPU enables testing various physical mechanisms that provide important insight into predictive properties of physics-based parallel discrete-event simulations.

An example of our SciDES architecture is shown in Figure 2. The class MPDES abstracts the virtual multi-processor DES environment. In this diagram dark dashed arrows represent ownership (the “has a” relationship) and light dashed arrows mark class instantiation from template classes.

## PARALLELIZATION

The parallelization of asynchronous (event-driven) continuous PIC models presents a number of challenges. As in conventional (time-driven) simulations, it is



**FIGURE 2.** The *MPDES* class collaboration diagram. The *MPDES* object encapsulates the global simulation geometry properties and defines the table of virtual DES processes.

realized by decomposing the global computation domain into subdomains. In each subdomain, the individual cells and particles are aggregated into containers, which are mapped to distributed parallel processors in a way that achieves maximum load balancing efficiency. The parallel execution of conventional (time-driven) simulations is achieved by copying field information from the inner lattice cells to the ghost cells of the neighboring subdomains and exchanging out-of-bounds particles between the processors at the end of each update cycle. In 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 levels throughout the simulation domain), but occur at arbitrary time intervals. This may introduce synchronization problems if some processors get ahead (in simulated time) of other processors (the "optimistic" approach, described later). As a result, a processor may receive an event message from a neighbor with a simulation time stamp that is in the receiver's past, thus causing a causality error. On the other hand, parts of a distributed discrete-event simulation can be forced to execute synchronously with remote tasks corresponding to the neighboring subdomains (the "conservative" approach). If so, the parallel speed-up critically depends on the underlying domain decomposition technique and additional predictive ("look-ahead") properties of the simulation in question. Regardless of the approach taken, it is important to note that DES computations offer substantial efficiencies compared to conventional explicit time-driven simulations due to the reduction in the amount of computation to be performed.

The following are some of the important issues that must be addressed in parallel discrete event simulations of continuous systems:

**Synchronization:** This is by far the paramount issue to be carefully resolved for achieving the best parallel execution performance. Broadly there are two approaches commonly used - conservative and optimistic.

**Conservative:** This approach always ensures *safe* timestamp-ordered processing. However, runtime performance is critically dependent on *a priori* determination of an application property called *lookahead*, which is roughly dependent on the degree to which the computation can predict future computations without global information. In the conservative approach, events that are beyond the next lookahead window are blocked until the window advances sufficiently far to cover those events. Typically the lookahead property is very hard to extract in complex applications, as it tends to be implicitly defined in the source code interdependencies. The appeal of this approach however is that it is one of the easiest schemes to implement if the lookahead can be specified by the application.

**Optimistic:** This approach avoids blocking by optimistically processing events beyond the lookahead window. When some events are later detected to have been processed in an incorrect order, the system invokes compensation code such as state restoration or reverse computation. Since blocking is not used, the lookahead value is not as important, and could even be specified to be zero. While this approach eliminates the problem of lookahead extraction, it has a different challenge – namely, support for compensation code.

**Combination:** Sometimes it might help to have some parts of the application execute optimistically ahead (e.g., parts for which lookahead is low are hard to extract), while other parts execute conservatively (e.g., parts for which lookahead is large, or for which compensation code is difficult to generate). In such cases, a combination of conservative and optimistic synchronization techniques can be appropriate.

**Load Balancing:** As with any parallel/distributed application, the best performance is obtained when the load is evenly balanced across all resources. In parallel simulation in particular, load imbalance can have a very adverse effect. This is because typically the slowest processor can hold back the progress of simulation (virtual) time, which in turn slows down even those processors that are relatively lightly loaded. In optimistic methods, a poor load distribution can lead to an excessive amount of rolled back computation.

**Automated/Adaptive:** Automated schemes are preferable for load-balancing at runtime. These schemes vary with the particular synchronization approach used.

**Support Primitives:** In order to permit automated/adaptive load balancing by the system, it is important to provide appropriate primitives to the application, so that application-level entities can be easily moved across processors by the system in a transparent manner as needed.

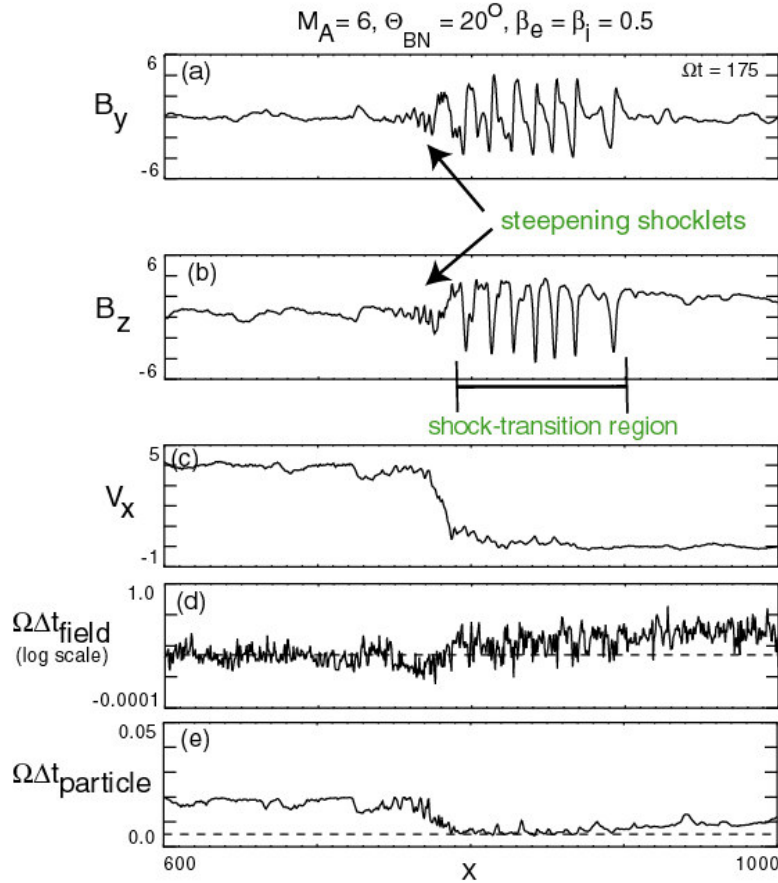
**Modeling and Runtime Interface:** To be able to decouple the implementation details of the parallel simulation executive from the application/models, it is best to define the model-simulation interface in an implementation-independent fashion. Not only this helps avoid reimplementing of models whenever the engine programming structure is modified, but it also permits experimentation with multiple synchronization and load-balancing approaches for the same application. Additionally, it enables engine-level optimizations to remain transparent to the application, so that the application-developer is not burdened or sidetracked with such issues during model development.

With the preceding issues in mind, we are carefully developing appropriate interfaces and implementations of our parallel execution engine. A brief description of our approach follows:

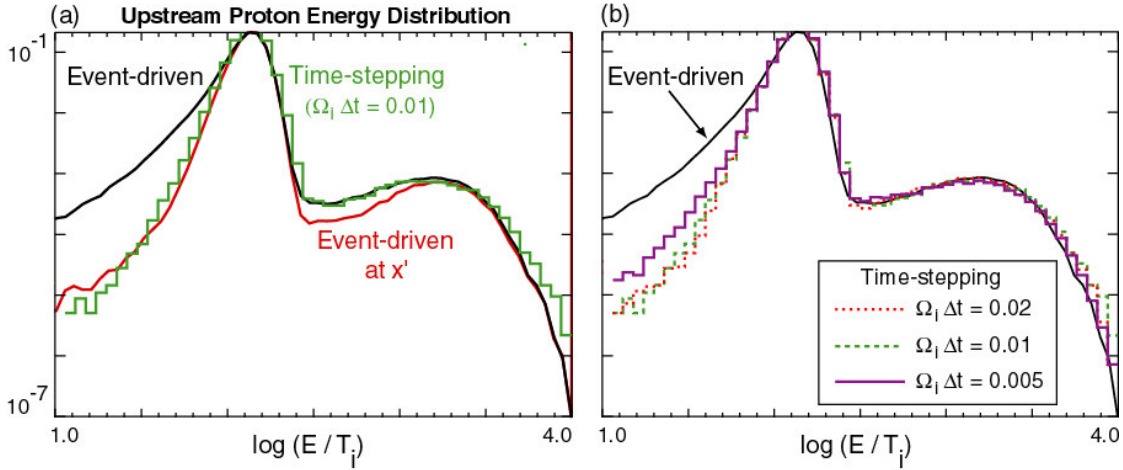
- The synchronization issue is being resolved by providing a transparent interface that does not mandate one synchronization approach over another. The underlying implementation is also being developed such that different model entities can choose different synchronization (conservative or optimistic execution style), as is most appropriate for them.
- The load balancing issue is being addressed by the use of an “indirect messaging” interface layer that decouples application entities from their processor mapping.
- The modeling and runtime interface is also kept abstract and flexible, so that radically alternative implementations can be implemented underneath the interface.

## EXAMPLE

We now demonstrate the power of this new methodology through an example. Figures 3a-c show the spatial profile of a fast magnetosonic shockwave as obtained from our new DES-based hybrid code<sup>4</sup>. Figures 3d and 3e show the field and average particle update rates, respectively. It is clear that the resulting waves span a wide range in spatial and temporal scales. The dashed lines in Fig. 3d-e indicate a timestep of  $\Omega t = 0.005$ . In the time-stepping simulation (STS), one chooses a fixed time increment whereas in DES the field update rate is automatically adjusted by the algorithm and each particle is advanced based on its own time-varying timestep. As figure 3d demonstrates, choosing a fixed uniform timestep would over-resolve the frequency in some regions and under-resolve it in other regions. Aside from computational inefficiency, this leads to less accurate solutions compared to DES as demonstrated in Figure 4. Figure 4a shows that in the DES model ions entering the near-front upstream region are trapped and pre-decelerated (enhanced counts at low energies) by the well-resolved turbulence generated in the vicinity of the shock front (Fig. 3b). On the contrary, the ion distribution function in STS is found insensitive to the upstream location of the reference upstream position because of the inability of STS to separate high-frequency wave activity from thermal noise.



**FIGURE 3.** DES Simulation of a fast magnetosonic shockwave and the associated particle acceleration. DES self-adapts by reducing local field update timesteps in order to resolve high-frequency turbulence preceding the shock front (d). STS refers to the standard time-stepped simulation.



**FIGURE 4.** Comparison of upstream particle distributions in the DES and time-stepped simulations for the parameters defined in Figure 3. (a) In the case of DES, the distribution function is evaluated in two different intervals: the entire upstream, (black curve) and excluding the first 75 ion inertial lengths upstream,  $x' = x_s - 75$  (red curve). Here  $x_s$  is the position of the shock ( $\sim 500$ ). (b) Comparison of distributions and particle acceleration in the DES and STS simulations for three time steps, as indicated. The time-stepped code does not properly account for deceleration in front of the shock (lowest energies) even at the smallest time step, and overestimates acceleration at the highest energies.

Solutions for three different STS time steps shown in Figure 4b demonstrate that the advanced predictor-corrector code<sup>6,7</sup> is accurate for most of the spectrum, but overestimates energization to the highest energies, and does not describe the near-upstream waves accurately enough to produce the ensuing pre-deceleration (low energies), even at the smallest time step. In figures 3-4,  $\Omega$  is the ion gyrofrequency,  $T_i$  is the ion temperature, all distances are normalized to ion inertial length, and density and magnetic field are normalized to their upstream values.

## SUMMARY

By combining techniques from two distinct fields of time-stepping and event-driven simulations, we have developed new and more efficient algorithms for modeling of systems described by partial differential equations. The DES algorithms are intelligent in that they self-adapt by adjusting local update rates in accordance with required accuracy, which results in more accurate and faster simulations. The technique is general and ideally suited for multi-scale problems characterized by disparate ranges in spatial and temporal scales. Given the ubiquitous nature of multi-scale problems in many areas of science and industry, we expect this technique to find broad application. We are currently extending this technique to several areas ranging from plasma physics to computational biology.

## ACKNOWLEDGMENTS

This research was supported by NSF ITR grants 0325046, 0539106, and 0326431.

## REFERENCES

1. Karimabadi, H., J. Driscoll, Y. A. Omelchenko, and N. Omidi, "A new asynchronous methodology for modeling of physical systems: breaking the curse of Courant condition", *J. Comp. Phys.* **205** (2), 755 (2005).
2. Karimabadi, H., J. Driscoll, J. Dave, Y. A. Omelchenko, K. Perumalla, R. Fujimoto, and N. Omidi, "Parallel discrete event simulation of grid-based models: asynchronous electromagnetic hybrid code", *Lecture Notes in Computer Science*, Springer-Verlag, in press, (2005).
3. Omelchenko, Y. A., and H. Karimabadi, "Self-adaptive time integration of flux-conserving equations with sources", *J. Comp. Phys.*, submitted, (2005).
4. Omelchenko, Y. A., and H. Karimabadi, "Event-driven hybrid particle-in-cell simulation: a new paradigm for multi-scale plasma modeling", *J. Comp. Phys.*, submitted, (2005).
5. Y. Tang, R. M. Fujimoto, K. Perumalla, H. Karimabadi, J. Driscoll, Y. Omelchenko, "Optimistic Parallel Discrete Event Simulations of Physical Systems using Reverse Computation," Principles of Advanced and Distributed Simulation, June (2005).
6. Karimabadi, H., D. Krauss-Varban, J. D. 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, doi:10.1029/2004JA010478, (2004).
7. Krauss-Varban, D., From theoretical foundation to invaluable research tool: Modern hybrid simulations, Proceedings of ISSS-7, pp. 15-18, March, (2005).