

Optimistic Simulations of Physical Systems using Reverse Computation

Yarong Tang¹, Kalyan S. Perumalla^{2*}, Richard M. Fujimoto¹

Homa Karimabadi³, Jonathan Driscoll³, Yuri Omelchenko³

¹College of Computing, Georgia Institute of Technology, Atlanta, Georgia 30332-0280

²Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831-6085

³SciberQuest Inc., Solana Beach, California 92075

Abstract

Efficient computer simulation of complex physical phenomena has long been challenging due to their multi-physics and multi-scale nature. In contrast to traditional time-stepped execution methods, we describe an approach using optimistic parallel discrete event simulation (PDES) and reverse computation techniques to execute plasma physics codes. We show that reverse computation-based optimistic parallel execution can significantly reduce the execution time of an example plasma simulation without requiring a significant amount of additional memory compared to conservative execution techniques. We describe an application-level reverse computation technique that is efficient and suitable for complex scientific simulations.

1. Introduction

Parallel Discrete Event Simulation (PDES) has been an active research area in the high performance computing community for many years. Synchronization techniques for PDES systems are usually classified into two principal categories: conservative approaches that avoid violating the local causality constraint, and optimistic approaches that allow violations to occur, but provide a mechanism to recover. The operation of recovering a previous state in an optimistic parallel simulation is known as a *rollback*, and involves undoing incorrect computations.

A widely used technique for implementing rollback is *state-saving* that saves the values of state

* Corresponding author: [Email: perumallaks@ornl.gov](mailto:perumallaks@ornl.gov), [Phone: \(865\) 241-1315](tel:(865)241-1315), [Fax: \(865\) 576-0003](tel:(865)576-0003)

variables prior to an event computation and restores them by referring to these saved values upon rollback. *Copy state saving* creates an entire copy of a logical process's modifiable state. For simulations that have a small number of state changes for each event computation, *incremental state saving* can reduce the time and memory overheads of state saving by only keeping a log of changes to individual state variables. When a large portion of the state tends to change during each event, *infrequent state saving*, which periodically saves an entire copy of the modifiable state, can be an attractive alternative. A relatively new technique for rollback, *reverse computation* [1], realizes rollback by performing the inverse of the individual operations executed in the event computation. These techniques have been exploited in small- and large-scale parallel simulations.

However, advances in PDES research to date have yet to be explored in space physical science, where multi-physics and multi-scale physical systems are modeled by partial differential equations and particles. Traditionally, models of such physical systems have been simulated using time-driven or time-stepped approaches [2]. These simulations advance their states based on explicit time discretization imposed by the global numerical stability criterion, the Courant-Friedrichs-Levy (CFL) condition $\Delta t < (\Delta x/V)_{\min}$, where Δx and V are the spatial mesh size and the rate of local activities or local propagation speed, respectively. Such approaches have two inherent limitations which have prevented the simulation of more complex physical systems that are important in plasma physics and other areas of science. First, the time increment used in simulation is constrained by the global CFL condition to ensure the numerical stability, and results in different degrees of accuracy in multi-scale systems where rates of propagation differ in different regions. Secondly, synchronous updates of states across the entire simulation domain at a fixed time step can lead to unnecessary computations in less active regions where there are few or no state changes during a time increment. Various adaptive schemes [3, 4, 5, 6] have been developed in time-stepped research to allow variable temporal or spatial

discretization in an effort to overcome the limitations imposed by fixed time steps; among those is the most developed and widely used technique Adaptive Mesh Refinement (AMR) [3]. Allowing recursive creation of refined grids within existing ones, the AMR technique is able to resolve the computational regions of interest at a higher resolution, both in time and space. However, the time step for each refinement grid is still constant and constrained by the local CFL condition in that grid, and consideration for interpolation, synchronization among grids and flux conservation at grid boundaries adds much complexity to the algorithm. Recently research in nonlinear electrodynamics applications has seen an emerging time-stepped technique called Asynchronous Variational Integrator (AVI) [7] which bears similarity to the asynchronous event processing paradigm in discrete event simulation approaches. It allows the selection of independent time steps in each spatial mesh and implements asynchronous updates of simulation states via a priority queue similar to the pending event list/set in DES. Since the AVI approach is based on the Hamilton's variational principle, it has the desirable properties of conservation of local energy and momenta, subject to solvability of the local time steps. But its applicability is limited to dynamical systems in which the Lagrangian is expressible as a sum of component sub-Lagrangians, and like any other time-stepped approach, the time steps are subject to the CFL condition.

Discrete event simulation was recently used to model spatially discretized physical systems [8] where it is shown that this approach not only alleviates the constraint of the CFL condition but also provides a significant performance advantage over the time-stepped approach. The performance advantage in discrete event simulation approach to multi-scale physical systems such as the plasma simulation presented in [8] comes from the natural decoupling of spatial scales in time. This is capable of capturing both the fine resolutions at particle scales in fast-evolving regions and the coarse resolutions in less active regions. Interestingly, the "irregular time steps" inherent in DES seem to be

an elegant solution to the ultimate goal of getting around the global CFL condition the time-stepped research has long been seeking to achieve. In our work, we further the research effort in [8] and apply optimistic parallel discrete event simulation techniques to this problem. Due to the unique characteristics of plasma simulations that pose great modeling and simulation challenges, we take a simulation approach that is efficient and yet general to a broad category of physical systems. Because memory constraints are often a severe limitation in the size of the computations that can be performed, reverse computation offers greater promise than traditional state saving techniques. We explore the use of reverse execution for plasma simulations to gain new insights for such challenging, complex physical systems. The combination of DES methodology and reverse computing techniques offers the potential to dramatically reduce the amount of time required to perform plasma simulations without incurring a large penalty in additional memory requirements.

The main contributions in this work can be summarized as follows. To our knowledge, this is the first work to apply reverse computation techniques to parallel physical system simulations and to show performance advantages using this approach. In addition, we provide a simple model and guidelines for creating reverse simulation codes at the application level that may help physicists in many application domains such as space physics and astrophysics to develop simulation prototypes without comprehensive knowledge of PDES mechanisms.

The remainder of this paper is organized as follows. The next section discusses related research. Section 3 provides an overview of the physical system that we simulate, and outlines the reverse computation approach. Section 4 gives an in-depth discussion of the reverse computation implementation and discusses challenges associated with this approach. Section 5 presents experimental results from a preliminary performance evaluation study. We conclude by reporting ongoing and future work in this area and provide guidelines for reversing parallel physical simulation

codes.

2. Related Work

A limited amount of research has examined physical system simulation using parallel discrete event simulation techniques. Perhaps the earliest was the “colliding pucks” application developed for the Time Warp Operating System (TWOS) [9]. This work, modeling a set of pucks traveling over a frictionless plane, was used to benchmark an early implementation of the Time Warp protocol. Lubachevsky discusses the use of conservative simulation protocols to create cellular automata models of Ising spin [10]. Other work describes challenges in using discrete event simulation techniques for a few 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].

Seminal work in optimistic parallel discrete event simulation was completed by Jefferson [14]. State saving has historically been the dominant approach to enabling the rollback of computations. Much work has been done to reduce the cost of state saving both using hardware [15] and software support. The less costly software approaches reported in the literature include copy state saving, incremental state saving [16, 17] and infrequent state saving [18, 19, 20, 21]. A different approach avoiding the cost of state saving was first described in [1], where reverse execution is used to roll back computations. Their reverse procedures were automatically generated by a compiler. More recent work using reverse execution for parallel network simulations, using manually generated code, was reported in [22]. Our work is different in that it applies reverse execution techniques to the simulation of physical systems that involve complex floating point operations and generates reverse code based on application semantics.

Traditionally, complex physical systems described by partial differential equations and particles are

modeled by time-stepped simulations. The authors in [8] recently demonstrated both the feasibility and efficiency of applying discrete event simulation (DES) methodology to model such complex systems. Their study shows that a performance improvement of up to two orders of magnitudes is achievable by switching from a time-stepped approach to an event-driven approach. Our study is based on their work and focuses on *parallel* execution techniques for their DES models. In particular, we examine the feasibility of optimizing the parallel synchronization mechanism by applying reverse computation techniques.

3. Overview

In this section we describe the computational plasma simulation model that is used as a case study. The underlying physics in this model may be a bit involved from a computer science point of view and is not required to be fully digested by a reader to understand the reverse computation techniques elaborated in section 4. Since our focus is on the use of reverse execution techniques in optimistic simulation, a high level understanding of the model suffices for our purpose. Interested readers can refer to [8] for a more complete discussion on the DES modeling approach.

3.1. Computational Model: PIC Simulation

One of the great challenges in space physics is to understand how the solar wind interacts with the earth's magnetosphere. While traditional work focused primarily on time-stepped methods, new methodology presented in [8] was among the first to apply a DES approach to simulating such complex physical systems. The basic idea behind the DES approach is, instead of advancing time in small (safe) time increments, state updates are scheduled as far into the future as locally predictable. This approach can lead to bigger leaps in simulation time and local stepping rather than global stepping in contrast to time-stepped method, and hence can be more efficient.

The particle-in-cell (PIC) [23] modeling approach serves as a good starting point in theoretical

research in plasma dynamics. This model is conceptually simple, yet captures the characteristics of physical phenomena involved and can be extended to more complex models. However, the computational ramifications of PIC simulation are nontrivial in that it simulates movements of millions or even billions of particles and their interactions with the fields, placing great demands on computational capacity. There has been considerable interest in improving its performance in the high performance computing community. Although we apply the reverse computation method only to this particular model in our study, the programming model and methodology used here are representative of grid-based physical systems. Therefore, the approach presented here may be applicable to a wider range of physical systems.

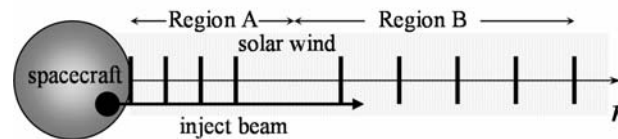


Figure 1. Schematic of the PIC model

3.2. *Spacecraft Charging Model*

We limit our simulations to a one-dimensional electrostatic model using spherical coordinates. Figure 1 illustrates a plasma PIC simulation of charging a spacecraft immersed in neutral plasma by injecting a charged beam from its surface [23]. The spacecraft is initially charge-neutral and immersed in the charge-neutral plasma of the solar wind. A charged beam is periodically injected from the spacecraft surface. The effect of beam injection is two-fold: the surface charge of the spacecraft incurs a change equal in magnitude but with an opposite sign to the injected beam particles; the charge in the cell immediately to the right of the spacecraft incurs a change equal to the beam both in magnitude and sign. The latter leads to a corresponding change in the electrostatic field which in turn affects the plasma particles' movement, triggering a self-consistent change in the field as this cycle of cause and

effect repeats. Here the interactions between particles and the field present a challenge in time-stepped simulations where special techniques are typically applied to break the “coupling,” whereas the “decoupling” is realized naturally in DES, as will become clearer shortly. Despite its simple concept, the spacecraft charging problem does have practical significances in space science. For example, the results of the simulation can help spacecraft designers understand plasma dynamics as well as its charging effects on the spacecraft, so that appropriate beam injection devices could be equipped on the spacecraft to offset the accumulated charge from plasma particles to prevent spacecraft internal electronic devices from malfunctioning.

With the physical phenomena and significance of this PIC model in mind, we are now in a position to take a closer look at the details in our simulation. In our spacecraft model, the simulation domain is divided into “cells” with each cell modeled as a Cell class. Upon simulation initialization, the spacecraft and all cells are charge-neutral, with each cell uniformly loaded with equal number of electrons and protons. Then a beam is injected from the spacecraft surface into the first cell, exciting solar wind particles and propagating to other cells. The main numerical computations involved are summarized as follows.

- Field update.** Because Gauss’s law allows the field value to be solely determined by the charge enclosed at a boundary, the field value can be expressed as $E \propto \frac{Q}{r^2}$ in spherical coordinates, where Q is the enclosed charge within the distance of r . Using the differential form of Gauss’s law, $dE \propto \frac{dQ}{r^2}$, one can solve the field locally and update any field changes in a cell by keeping a running sum of all the charges which have crossed its left and right boundaries [8].
- Particle movement.** Initially, all particles in cell i take on a charge given by $Q_i = 4\pi r_i^2 \Delta r_i \rho_i$, where r_i , Δr_i and ρ_i are the center coordinate, the width and the particle density of cell i ,

respectively. As a particle moves within a cell, its motion properties, such as acceleration Acc , velocity Vel and position Pos , can be easily calculated via equations of motion. In particular, the time at which a particle reaches the cell boundary and moves into the neighboring cell has significance in the DES approach: it indicates when the cell field values need updates and when the particle motion trajectory may be recomputed in the new electric field; it reflects the fine timescale of particle movement that the event-driven approach is designed to cope with. The future exit time of a particle is solved by finding the roots dt of the quadratic equations [8]:

$$\frac{1}{2} * Acc * dt^2 + Vel * dt + Pos - CellWidth = 0 \quad (1)$$

$$\frac{1}{2} * Acc * dt^2 + Vel * dt + Pos = 0 \quad (2)$$

which give

$$dt = \frac{-Vel \pm \sqrt{Vel^2 - 2 * Acc * (Pos - CellWidth)}}{Acc}, \frac{-Vel \pm \sqrt{Vel^2 - 2 * Acc * Pos}}{Acc} \quad (3)$$

Equations (1) and (2) represent the right and left exit conditions, respectively. dt is the time difference between the current simulation time and the particle's last movement time. The smallest real root for dt in (3) is used for predicting the particle's future exit time.

- **Field and particle interactions.** In the plasma simulation, the field and the particles are tightly coupled physical entities. When a particle moves out of a cell and then enters another cell, the field values in both cells will change. However, only when the change in a cell's electric field exceeds a predefined threshold value can the change actually impact all particles within that cell. Such an event is called a “wakeup” where the exit times of all particles in that cell have to be recomputed, using the new field value. The consequence of a wakeup is usually the acceleration of particles moving across cell boundaries, which in turn results in field updates. Traditional time-stepped plasma simulations advance the field and move different types of particles based on the same time

step, regardless of the difference in their evolution rates. The event-driven paradigm allows both the field and particles to evolve at their own timescales, leading to an elegant solution that is efficient and a close match to the physical phenomena.

3.3. Model Execution and Parallelization

In PDES of a PIC model, each cell in the simulation domain is mapped onto one logical process (LP) and one or multiple LPs are mapped onto one physical process (PP) according to a carefully tuned load-balancing scheme. Two distinct regions are shown in figure 1, based on different grid spacing, which in our PIC model differentiate the active regions from inactive regions. Note that field updates are limited to active cells in which events are allowed to be scheduled. The state of each LP includes the cell electric field variables and the states of all particles within the cell boundaries.

```
Cell::arrival( ParticleArrivalEvent *e ) {
    if ( this cell is active ) {
        update cell state;
        insert particle in cell;
    } else if ( e is a beam particle ) {
        activate cell;
    }
}
Cell::departure( ParticleDepartureEvent *e ) {
    if ( particle bounced from right neighbor ) {
        bounce particle back; // no cell state change
    } else {
        update cell state;
    }
}
Cell::inject( ParticleInjectEvent *e ) {
    update cell state;
    insert beam particles;
}
```

Figure 2. A simplified spacecraft charging model

The dynamic behavior of the system is driven by particle movements that are modeled by events.

There are three types of events associated with particle movement: *ParticleArrivalEvent*, *ParticleDepartureEvent* and *ParticleInjectEvent*. The corresponding event handlers are outlined in figure 2. When the simulation starts, all cells are initialized, including electric fields and uniformly distributed particles along with their physical states (mass, charges, velocities, positions, etc.). Each particle's movement is determined by its *MoveTime* or cell exit time. Whenever a particle is created in the case of a beam injection, or inserted in a cell when it has just entered a new cell, its exit time must be calculated, and a pair of departure and arrival events scheduled at the exit time in the corresponding cells. The departure event is scheduled on the source cell, and the arrival event is scheduled on the destination cell. In addition, a particle's exit time needs to be recalculated whenever the hosting cell "wakes up". The physical activities associated with each component shown in figure 2 will be explained in detail in section 4.

3.4. Reverse Computation Approach

The characteristics of this plasma simulation present three major challenges concerning the synchronization of parallel computations. The rationale for the parallelization approach used here is based on the following considerations.

- **Lookahead.** The simulation is highly dynamic. The amount of parallelism can vary dramatically as the simulation progresses. Dependencies among events are governed by each particle's exit time, but this time can be arbitrarily close in the near future. In the spacecraft charging model, particle arrival events can be scheduled across neighboring cells with almost no simulation time delay. This low level of "predictability" results in a low, dynamically-changing value of lookahead that makes efficient execution using conservative synchronization techniques difficult. In general, low lookahead is inherent in DES models of continuous systems due to their formulation which incur "immediate" updates via events to neighboring logical processes. This suggests that

optimistic synchronization [14] may be a more natural choice for this simulation. It is worth noting that it is not possible to use constrained variants of optimistic execution, such as local Time Warp (such as separating aggressiveness from risk).

- **Memory.** Realistic plasma simulations involve a large number of events, desirably on the order of billions. Complex data structures are often needed. This makes traditional approaches to optimistic execution using state saving problematic: the amount of memory required can be prohibitively large. Further, the amount of model-level computation performed for each event tends to be relatively small, on the order of a few microseconds on a contemporary CPU. This suggests that the time overhead for state saving may be significant, and hence could significantly degrade performance, even utilizing techniques such as incremental state saving. While various techniques to reduce the cost of state-saving-based rollbacks have been developed [24, 25], our experiences indicate much memory is still needed to achieve efficient parallel execution [26]. On the other hand, efficient reverse execution can take slightly more or even less computational time than in the forward execution while it has the potential to reduce or eliminate the memory needed for rollback. For these reasons, reverse computation was selected for the parallelization of this plasma simulation code. However, comparison of the reverse execution approach with other advanced check-pointing approaches remains an area of future investigation.
- **Floating point.** The reverse computation approach proposed in [1] uses an automated approach to creating the reverse execution code for each executable line of forward execution code. For example, a decrement statement is generated to undo an increment statement in the forward execution code. This approach becomes problematic when floating point arithmetic is used because the computation may not be easily reversed due to effects such as round-off error. Here, we explore a different approach where the program is viewed at a higher level of abstraction, and

suitable reverse computation code is developed manually. With our ongoing effort of applying reverse computation techniques to other plasma codes, we envision that a general framework for instrumenting the automation of reverse codes for certain plasma simulation could be developed in the near future.

These factors motivate the approach that was adopted for optimistic synchronization using manually derived reverse computation code. We believe this can be used to build a foundation for future work in developing scalable parallel simulators for complex physical systems.

4. Parallel Simulation Code

Here we use a one-dimensional model of the spacecraft electrostatic particle code as the illustrative example of our choice to discuss some of the challenges in generating the reverse code for this physical system simulation. There are two types of distinct physical entities in this simulation: particles and cells¹, with particles consisting of three species of particles - solar wind electrons, solar wind protons and injected beam particles. Particles move across cells and each cell keeps track of the particles residing within its own domain. The communication between adjacent cells occurs via particle movement events that contain information of particle physical states. The complex data structures housing the particles and the physical processes being captured require a careful modeling of the system. The object-oriented design used here allows one to encapsulate physical properties via classes.

The code in Figure 2 includes three event handlers, one for each type of event. Much of the complexity of the event computation is encapsulated within the **insert** and **update** operations. An **insert** operation includes an “insert” queue operation (data structures representing particles in the cell

¹ The spacecraft situated at one end of the spatial coordinate can be treated as a special cell that does not keep the physical states of particles. We use “cells” thereafter to refer to the regular cells unless otherwise specified.

can be organized as a priority queue sorted by their exit times or a list) and computation of the exit time of that particle from the cell based on equations of motion. An **update** operation recomputes the cell's field value based on the arrival particle's charge. The change in the field value may trigger an expensive wakeup computation that again scans the particle list and updates each particle's exit time.

It may be noted that the reverse computation techniques introduced in [1] would generate the reverse code for each instruction without taking into account the semantics of the higher level operations that are being performed. This clearly leads to inefficiencies for the queue management operations used frequently in this code, and as mentioned earlier, leads to difficulties concerning the reversibility of floating point operations. The model-specific approach taken here involves generating the reverse code for the application by exploiting knowledge of the higher level semantics of the operations being performed. We call this approach *application-level* reverse computation.

As a first approach, we investigated the use of a compiler for automatically generating reverse code from forward model code. The compiler would use automated, instruction-by-instruction reversal approach to achieve reverse code. However, the spacecraft charging code, representative of other complex physics models, was sufficiently complex for the compiler to resort to state-saving on most assignment instructions that were apparently destructive in nature (e.g., $x = f(y)$). Pervasive use of mathematical library functions such as *sqrt()* and *log()* also greatly hindered automation.

Careful readers may notice that the particle departure event handler depicted in figure 2 does not specify any deletion operation that matches the insertion operations in the particle arrival event handler. In fact, deletions are performed aggregately on an as-needed basis. This is done based on performance considerations. Instead of deleting each particle at its exiting time, a near-periodic deletion operation is used to amortize the cost of deletions in queues. It is observed that wakeup events within one cell happen almost periodically with a frequency determined by physical conditions in that

cell; furthermore, each wakeup requires a full scan of the cell's particle queue. We find that restricting deletion operations only at cell wakeup times reduces the total overhead of particle deletions without introducing excessive memory usage. Since the aggregate deletions are used to clean up the obsolete states for particles that have already exited, no rollbacks are needed to recover these obsolete states in the event of undoing a wakeup.

Figure 3 shows the reverse code of the simulation. Notice that after decomposing the forward code into components based on simulating physical processes, the reverse code is relatively easy to construct based on the operations shown in figure 3. The next step for generating the complete reverse code is just a matter of reversing each physical process. Examples of reversing some of the more difficult processes are shown in figure 3.

```
Cell::undo_arrival( ParticleArrivalEvent *e ) {
    if ( cell was activated ) {
        undo_activate cell;
    } else if ( cell already active ) {
        delete particle in cell;
        undo_update cell state;
    }
}
Cell::undo_departure( ParticleDepartureEvent *e ) {
    if ( particle was bounced from right neighbor ) {
        undo_bounce particle;
    } else {
        undo_update cell state;
    }
}
Cell::undo_inject( ParticleInjectEvent *e ) {
    delete beam particles;
    undo_update cell state;
}
```

Figure 3. A simplified reverse code of the spacecraft model

The **insert** operation appears in both particle arrival and injection event handlers. Its effect includes assigning memory for the new particle states in the queue and scheduling arrival/departure event pairs at each particle's future exiting time. Conversely, the **delete** operation in the reverse code should

perform the corresponding inverses of these processes. Particles in each cell are organized in a FILO (first-in last-out) queue, so the `delete` operation always removes the particle at the head of the queue that is exactly the same particle that was inserted in the forward computation. As for “undoing” event scheduling, it is assumed the underlying simulation engine provides the application with a primitive for explicitly retracting scheduled events; this is very useful in implementing the `delete` operation.

The effect of the `activate` process is to load an inactive cell (in which particle movements are effectively absent) with particles laid out in the cell following a uniform distribution. Nevertheless, its basic operation is in fact multiple particle insertions, and consequently, its reverse code can utilize the `insert-delete` pair operations described above.

The cell state `update` process is in fact the most complex process and its reverse code is not trivial. Each `update` performs two major computations: it computes the cell’s new field values and then updates the cell’s particle queue. Each cell computes its field locally and keeps track of field values at its left and right boundaries by summing the charges passing through that boundary. During a cell’s field update, an addition of charge at its boundary can be simply reversed as a subtraction of the charge at the boundary upon rollback. The update on a particle queue, however, is not as easy. It requires a check of the wakeup condition (i.e., the field change exceeding a threshold) and triggers a wakeup event if necessary. Otherwise it makes no changes. As previously described, a cell wakeup is the single most computationally expensive operation in the plasma simulation. In the event of a wakeup, all particles’ states are recomputed based on the new cell field values and obsolete particle states are erased. The destructive nature of the re-computation is one of the principal challenges in generating the reverse code for this simulation.

```

Particle::update_position( double dt ) {
    Pos += Vel * dt + 0.5 * Acc * dt * dt;
}
Particle::update_velocity( double dt ) {
    Vel += Acc * dt;
}
Particle::update_acceleration( double cell_field ) {
    Acc = cell_field * particle_charge / particle_mass;
}
Particle::reverse_position( double dt ) {
    Pos = Pos - Vel * dt - 0.5 * Acc * dt * dt;
}
Particle::update_state( double cell_field, double dt ) {
    update_position(dt);
    update_velocity(dt);
    update_acceleration(cell_field);
    dt = update_dt(); // solve eq. (1) and (2)
    MoveTime = now + dt;
}
Particle::reverse_state( double cell_field, double dt ) {
    update_acceleration(cell_field);
    update_velocity(-dt);
    reverse_position(dt);
    dt = update_dt();
    MoveTime = now + dt;
}

```

Figure 4. The reverse code example of the particle states

One example of an irreversible operation is the calculation of a particle's exit time *MoveTime*, solved through equations (1) – (3). The smallest real value of *dt* in equation (3) is used to determine *MoveTime* and the exit direction. An initial inspection of the quadratic equations seems to suggest the impossibility of applying reverse computation to this process. However, if we apply the reverse computation approach at the application level, we find that the movement of the particle is highly reversible. Based on the physical laws of particle motion, the recovery of *dt* does not require the direct inverse of the quadratic equations. Indeed, as illustrated in figure 4, the particle's acceleration, velocity and position states can be simply rolled back by reverse computation and then the critical state *dt* can be reconstructed by carrying the forward computation using the recovered particle states. Note that the

parameters *cell_field* and *dt* in the reverse code refer to the rolled-back *cell_field* value and the time difference between now and the time when the particle moved right before the wakeup. Finally, we note that random number generation is essential to this parallel simulation. Therefore, an efficient random number generator (RNG) that is reversible and has a long period is required. For this purpose, we use the reversible RNG that is described in [1].

5. Performance Evaluation

The application-level reverse computation approach is best implemented in a system that decouples implementation details of the simulation engine from the application in order to allow one to focus efforts on application semantics. As a result, a simulation engine that can support reverse computation at an application level and provide efficient management of large numbers of events with minimal storage requirement is needed. In addition, the simulation engine should have the flexibility and extensibility to support future refinements of the parallel simulation.

The parallel simulation code using reverse execution described in the previous section was implemented using *μsik*, a general-purpose parallel/distributed simulation engine based on a micro-kernel architecture [27]. *μsik* provides primitives supporting multiple synchronization approaches, including optimistic and conservative synchronization, as well as means to relax event ordering rules and mixing different approaches to synchronization within a single parallel execution. It therefore provides the capabilities needed for the parallel physical system simulations described here.

In a *μsik* simulation, logical (simulation) processes (LP) are fully autonomous entities that communicate via events. In our simulation model, each cell is implemented as an LP and can choose to run conservatively or optimistically. A conservative implementation of the simulation described here performed very poorly, due to poor lookahead, and is not discussed further. We focus on optimistic execution using our reverse handlers to support rollback.

5.1. Experiment Configuration

To demonstrate the feasibility and efficiency of reverse computation in the electrostatic plasma simulation, we carried out all experiments on a Symmetric Multi-Processor (SMP) machine running Red Hat Linux 7.3 with a customized 2.4.18-10smp kernel. The SMP machine is equipped with eight Pentium III 550MHz Xeon processors that share 4GB of memory.

We use normalized units throughout our simulation, where length, time and velocity are normalized to electron Debye length, electron plasma frequency and electron thermal velocity, respectively. The spacecraft is assumed to have a 500 unit radius and each cell has a width of 0.24 units. The solar wind plasma is initially loaded with uniformly distributed electrons and protons. We choose the initial values of 30 electrons and 30 protons per cell. The injected positron beam has energy of 10 keV with an injection period of 0.004. Upon initialization, there are up to 7000 cells of which the first 70 close to the spacecraft are “active”; as the simulation progresses up to time 60, the beam travels further away from spacecraft surface and thus more cells are activated.

5.2. Parallel Performance

Figure 5 shows a snapshot visualization of phase space structures for the solar wind electrons, protons and beam particles from a time-stepped simulation and the optimistic PDES simulation. Both simulations are run up to 60 time units, and with the same simulation parameters except using different random number generators (RNG). The PDES used a specialized reversible RNG in contrast to the generic single-stream RNG used in the time-stepped simulation. Despite this difference, the two phase space structures at the end of the simulations are rather close in form. The result from the PDES execution with reverse computation is verified to accurately capture the main features of movement for all three species of particles at the end of the simulation. In particular, it can be seen that the beam front in both simulations has propagated to the same distance and beam particles display a similar shape in phase space. It is also evident that the electron phase space has a finer resolution in the PDES

case for up to 4000 cells. This is the result of its fine time-scale based on individual particles. A minor difference to note is that, in the PDES case, the phase space does not extend all the way to the right wall, whereas in the time-stepped model, it does. This is because we model an expanding box in PDES but not the time-stepped model. Overall, the results helped serve as validation of our optimistic simulation model against the original sequential simulation model.

Validation could perhaps be improved by employing more rigorous comparison testing, such as using deviation measures for phase-space plots. For our purposes, however, the closeness of match by visual comparison is sufficiently clear.

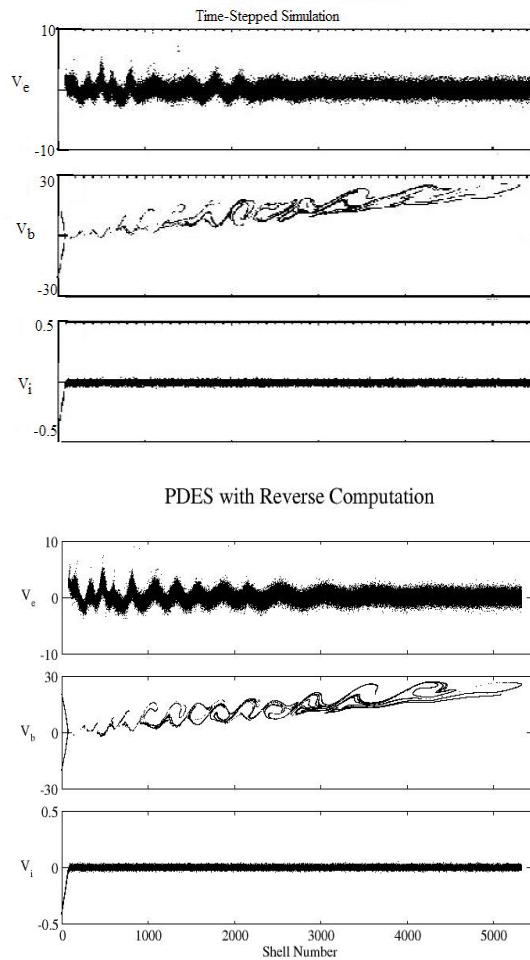


Figure 5. Validation by phase space comparison of time-stepped simulation and PDES with reverse computation.

The speedup of DES over TDS has been discussed in great detail in [8]. In our work, we focus on further improving the DES performance by realizing parallelization in the simulation and utilizing optimistic synchronization. All the parallel experiments discussed in the following section were run with the same physical parameters and resulted in the same number of committed events as the sequential runs. Figure 6 shows the parallel speedup in terms of execution time for up to 8 processors. The sequential data is measured by running the parallel code on a single processor. It should be noted that the single processor execution incurs neither rollbacks nor state saving overhead. Because μsik was designed for both efficient sequential and parallel execution, we believe these measurements reflect the performance one could expect to see using a reasonably efficient sequential simulation engine.

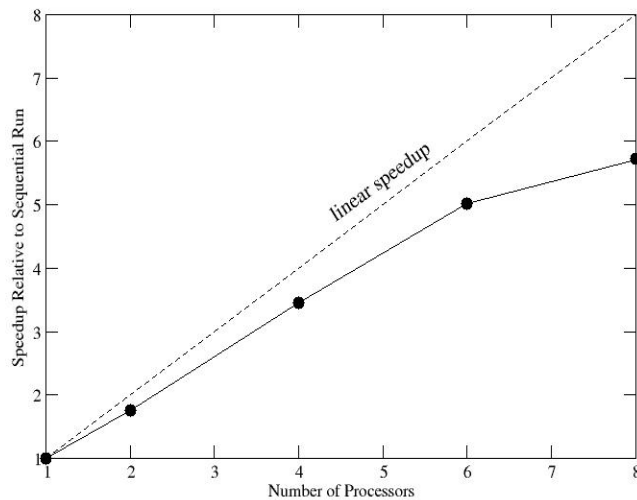


Figure 6. PDES vs. sequential DES

We observe that the optimistic parallel execution achieves a nearly linear speedup up to 4 processors, but the performance improvement is somewhat less in going from 4 to 8 processors. This phenomenon is largely due to the fact that there is relatively little computation per particle event. As

the computation is distributed over more and more processors, the amount of computation between event communications decreases, resulting in reduced speedup. We expect that this problem will not persist if a larger, more complex physical model such as a three dimensional plasma code were used. An initial test with an increased simulation time of 2 units did show better speedup performance due to the fact that the longer the simulation runs, the more cells are activated, resulting in more balanced computation for each processor.

A second factor that results in less than optimal performance concerns the distribution of the workload. Figure 7 shows the amount of computation assigned to each processor in each of the runs. Here, the load is distributed by first dividing the physical area encompassed by the simulation into two regions (as illustrated in figure 1) with the initially active cells closer to the spacecraft in the “heavy activity” region, and other cells forming the “less active” region. Cells in the active regions are evenly grouped and distributed among the available processors, while other cells are grouped into sub-regions or “blocks” and distributed among processors in a round-robin fashion. All simulations shown in figure 7 have a fixed “block” size. We observe that during the lifetime of each simulation, the processor load shows significant variation as more and more cells become active. Upon simulation termination, the simulation with the largest number of processors tends to be the least balanced. The imbalance is inherent of such simulations due to their highly dynamic nature and the static load-balancing scheme. Further investigation of characteristics of electrostatic plasma simulations is needed to aid in the development of a more efficient load-balancing algorithm for this application that can lead to better parallel performance for large numbers of processors.

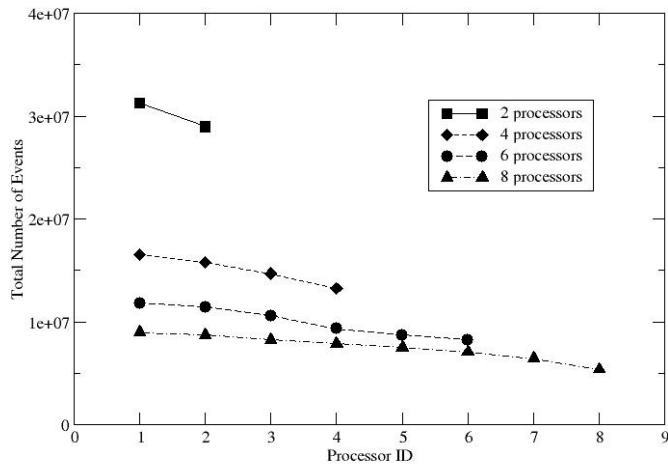


Figure 7. Event rate distribution

5.3. Efficiency

Intuitively, grid-based physical systems such as the electrostatic plasma simulation studied here have the desirable features of locally solved field values and queuing/dequeuing operations that are time-reversible, but the evolution of the system itself (beam injections, cell wakeups in our case) is not time-reversible. However, with the application-level reverse computation illustrated here, we have shown that numerical operations in the electrostatic plasma simulation chosen for this study are truly reversible, despite round-off errors and irreversible evolution processes. The most important discovery from our study is that application-level reverse computation may be quite efficient for these scientific simulations.

The efficiency mainly comes from two contributing factors: the smaller amount of additional memory required for optimistic execution, particularly, queue operations where no additional state is required to perform rollbacks; note the fact that the simulation is not constrained by arbitrarily small look-ahead values. However, there are still important practical issues related to reverse computation.

Ideally, one would like to apply reverse computation to all reversible operations. But reverse

computation also comes at a cost: if the number of destructive operations is sufficiently large and no efficient application-level reverse computation can be found, employing reverse computation can result in worse performance than state-saving. One such case as pointed out in [1] is when a rollback spans several processed events. Merely switching pointers to restore a state based on the earliest rolled back event incurs a small cost in copy state-saving; while reverse computation must roll back one event at a time and thus excessive rollbacks can cause performance to degrade considerably. The effect of this is particularly severe in our simulation when a rollback spans multiple wakeup events.

Our solution to reducing the rollbacks of costly wakeup events is to limit the “optimism” of the parallel execution. The idea of controlling optimism is not new. One such approach is first described in the Moving Time Window (MTW) protocol [28], where LPs are not allowed to advance beyond a time window above the GVT. Although other approaches such as probabilistic rollbacks [29], local rollbacks [30], Breathing Time Buckets [31] and Wolf Call mechanism [32], have been proposed to counter the effect of over-optimism, we simply choose to use a time window-based approach because of its simplicity and low overhead. *μsik* supplies simulation applications with a convenient facility for our purpose. A “*run-ahead*” parameter, which is in fact the moving window size, can be set by the model upon simulation initialization. This limits how far in simulation time each LP can run ahead of other LPs during optimistic execution. By tuning the run-ahead parameter based on cell wakeup frequency, we are able to reduce or eliminate consecutive rollbacks of wakeup events. The significant performance gain in our experiment indicates the extra operation associated with the window maintenance is a small cost to pay. Further improvement in parallel performance is possible by fine-tuning the run-ahead value. This is left for future investigation.

In addition to the basic reverse computation techniques discussed here, advanced reverse techniques can be applied to the plasma simulation. For example, compiler-supported reverse computation can be

used to further optimize the parallel performance at run-time. This approach is beyond the scope of our discussion and will be studied in the future.

6. Conclusions

In this work, we have applied reverse execution to perform parallel discrete event simulations of a physical system. The spacecraft charging application was used to demonstrate feasibility of applying this approach in modeling and simulation of physical systems. The spacecraft charging application is a sufficiently complex application, and hence particularly challenging for reverse execution. It contains several nuances, such as: dynamic activation of neutral cells by beam particles; runtime retraction of particle events; en masse “wake up” of all particles within a cell; periodic, dynamic creation of particles; multiple particle types and their properties; and, complex boundary conditions.

Despite the complexity, we demonstrated that an application-level reverse computation approach can be used to (manually) generate efficient reverse code. These results suggest that reverse computation merits further investigation as an approach for parallel/distributed simulation of physical systems modeled using a discrete event simulation paradigm.

As previously mentioned, the PIC simulation considered in this paper is only a simplified example of reverse execution in simulating physical systems. The examples given in section 4 are representative and certainly do not encompass the diversity and complexity of all physical system simulations. However, the underlying reverse techniques can be used in other grid-based models without extensive modifications. Here we provide some guidelines for the development of parallel physical discrete event simulations using reverse computation. Since our exploration of reverse computation is an on-going research effort, the guidelines provided here should be used as references rather than strict rules for applying reverse computation in scientific simulations.

- Reverse computation is well-suited for fine-grained applications such as 1D electrostatic grid-

based plasma models. It is especially useful where efficient queue management is needed. But other optimization techniques should also be considered in order to fully optimize parallel performance.

- Good knowledge of the application semantics, especially the underlying physics, can be beneficial in producing reverse code for physical systems. Model-specific optimization can be quite efficient but requires knowledge of application-level operations. The simple example of reversing the quadratic equation would not have been efficient, if at all possible, without knowledge of the physics involved (particle's motion in this case).
- The modeling process largely determines how successfully reverse computation will improve parallel performance. Initial analysis in [1] shows that complex use of jump instructions such as goto, break and continue are difficult to optimize in terms of memory usage.
- In modeling physical systems, it seems desirable to avoid monolithic code for event handlers and instead use of functions calls is preferable that are associated with each physical process. If an event handler only consists of a long sequence of simple instructions, it is difficult to extract application semantics and therefore reverse computation can degenerate to instruction-by-instruction reverse execution. Using small function calls that reflect a more accurate mapping to physical processes helps to develop reverse codes based on physical properties of the system. Another obvious advantage is easier debugging and testing for the reverse code. As future work, we plan to investigate approaches by which application-level semantics can be exposed to an automated reverse code-generating compiler for generating reverse code comparable to manual, application-level reversal techniques described here. One possibility is to define this in an application domain-specific manner, such as a library of forward and reverse particle physics models.

The work presented here is only an initial step based on a simplified physical system. Yet, the results show promise. Our goal is to build a scalable parallel simulator for complex physical systems by exploitation of more advanced reverse computation techniques.

Acknowledgements

This work was supported in part by the National Science Foundation grants ATM-0326431, 0325046 and 0539106.

References

- [1] Carothers, C. D., K. Perumalla and R. M. Fujimoto. Efficient Optimistic Parallel Simulation Using Reverse Computation. *ACM Transactions on Modeling and Computer Simulation*, **9**(3): 224-253, 1999.
- [2] Birdsall, C. K. and A. B. Langdon. Plasma Physics via Computer Simulation. *McGraw-Hill Book Company*, 1985.
- [3] Berger, M. J. and J. Olinger. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *J. Computational Physics*, **53**: 484-512, 1984.
- [4] Otani, N. F. Computer Modeling in Cardiac Electrophysiology. *J. Computational Physics*, **161**: 21-32, 2000.
- [5] Dawson, C. and R. Kirby. High Resolution Schemes for Conservation Laws with Locally Varying Time Steps. *SIAM Journal of Scientific Computing*, **22**(6): p. 2256, 2001.
- [6] Abedi, R., S. Chung, J. Erickson, Y. Fan, M. Garland, D. Guoy, R. Haber, J. M. Sullivan, S. Thite, and Y. Zhou. Spacetime Meshing with Adaptive Refinement and Coarsening, in *Proceedings of the 12th Annual Symposium on Computational Geometry*, 2004. p. 300-308.
- [7] Lew, A., J. E. Marsden, M. Ortiz, and M. West. Asynchronous Variational Integrators. *Archive for Rational Mechanics and Analysis*, **167**(2): 85-146, 2003.
- [8] 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. Computational Physics*, **205**(2): 755-775, 2005.
- [9] Hontalas, P., B. Beckman, M. DiLorenzo, L. Blume, P. Reiher, K. Sturdevant, L. V. Warren, J. Wedel, F. Wieland and D. Jefferson. Performance of the Colliding Pucks Simulation on the Time Warp Operating System. *Distributed Simulation, Society for Computer Simulation International*, 1989.

- [10] Lubachevsky, B. D. Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks. *Communications of the ACM*, **32**(1): 111-123, 1989.
- [11] Lubachevsky, B. D. Several Unsolved Problems in Large-Scale Discrete Event Simulations, in *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, 1993. p. 60-67.
- [12] Zeigler, B. P., H. Praehofer and T. G. Kim. Theory of Modeling and Simulation: Second Edition. *Academic Press*, 2000.
- [13] Nutaro, J. J. Parallel Discrete Event Simulation with Application to Continuous Systems. *Ph. D. Dissertation*. 2003. Electrical and Computer Engineering Dept, University of Arizona.
- [14] Jefferson, D. Virtual Time. *ACM Transactions on Programming Languages and Systems*, **7**(3): 404-425, 1985.
- [15] Fujimoto, R. M., J. J. Tsai and G. Gopalakrishnan. The Roll Back Chip: Hardware Support For Distributed Simulation Using Time Warp, in *Proceedings of SCS Distributed Simulation Conference*, **19**(3): 81-86, 1988.
- [16] Bauer, H. and C. Sporrer. Reducing Rollback Overhead in Time Warp Based Distributed Simulation with Optimized Incremental State Saving, in *Proceedings of the 26th Annual Simulation Symposium*, 1993.
- [17] Steinman, J. Incremental State Saving in SPEEDES Using C++, in *Proceedings of the 1993 Winter Simulation Conference*, 1993.
- [18] Lin, Y. B. and E. D. Lazowska. The Optimal Checkpoint Interval in Time Warp Parallel Simulation, *Technical Report 89-09-04*, Dept. of Computer Science and Engineering, University of Washington, 1989.
- [19] Bellenot, S. State Skipping Performance with the Time Warp Operating System, in *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, 1992. p. 53-61.
- [20] Franks, S., F. Gomes, B. Unger, and J. Cleary. State Saving for Interactive Optimistic Simulation, in *Proceedings of the 11th Workshop on Parallel and Distributed Simulation*, 1997. p. 72-79.
- [21] Fleischmann, J. and P. Wilsey. Comparative Analysis of Periodic State Saving Techniques in Time Warp Simulators, in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, 1995. p. 50-58.
- [22] Yuan, G., C. D. Carothers and S. Kalyanaraman. Large-Scale TCP Models Using Optimistic Parallel Simulation, in *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, 2003. p. 153.
- [23] Pritchett, P. L. and R. M. Winglee. The Plasma Environment during Particle Beam Injection into Space Plasmas: 1. Electron-beams, *J. of Geophysical Res. – Space Physics*, 1987. **92** (A7): 7673-7688.
- [24] Jefferson, D. R., Virtual Time II: Storage Management in Distributed Simulation, in *Proceedings of the Ninth Annual*

- ACM Symposium on Principles of Distributed Computing*, 1990. p. 75-89.
- [25] Preiss, B. R. and W. M. Loucks. Memory Management Techniques for Time Warp on a Distributed Memory Machine, in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, 1995. p. 30-39.
- [26] Das, S. R. and R. M. Fujimoto. An Empirical Evaluation of Performance-Memory Tradeoffs in Time Warp. *IEEE Transactions on Parallel and Distributed Systems*, 1997. **8**(2): p. 210-224.
- [27] Perumalla, K. μ sik -- A Micro-kernel for Parallel/Distributed Simulation Systems, in *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, 2005.
- [28] Sokol, L. M. and B. K. Stucky. MTW: Experimental Results for a Constrained Optimistic Scheduling Paradigm, in *Proceedings of the SCS Multiconference on Distributed Simulations*, 1990. **22**: 169-173.
- [29] Madiseti, V., D. Hardaker and R. Fujimoto. The MIMDIX Operating System for Parallel Simulation. *Journal on Parallel and Distributed Computing*, 1993. **18**(4): 473-483.
- [30] Dickens, P. M. and P. F. Reynolds. SRADS with Local Rollback, in *Proceedings of the SCS Multiconference on Distributed Simulation*, 1990. **22**: 161-164.
- [31] Steinman, J. S. Breathing Time Warp, in *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, 1993. p. 109-118.
- [32] Madiseti, V., J. Walrand and D. Messerschmitt, WOLF: a Rollback Algorithm for Optimistic Distributed Simulation Systems, in *Proceedings of the 20th Conference on Winter Simulation*, 1988. p. 296-305.