Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

PARALLEL ASYNCHRONOUS HYBRID SIMULATIONS OF STRONGLY INHOMOGENEOUS PLASMAS

Yuri A Omelchenko
Homa Karimabadi

SciberQuest, Inc.
12837 Caminito Del Canto
Del Mar, CA 92014, USA

ABSTRACT

Self-adaptive discrete-event simulation is a general paradigm for time integration of discretized partial differential equations and particle models. This novel approach enables local time steps for equations describing time evolution of grid-based elements (fluids, fields) and macro-particles on arbitrary grids while preserving underlying conservation laws. The solution-adaptive integration ensures robustness (stability) and efficiency (speed) of complex nonlinear simulations. Using this technique we achieved a breakthrough in simulations of multiscale plasma systems. A new particle-in-cell simulation tool, HYPERS (Hybrid Particle Event-Resolved Simulator), which solves a set of strongly coupled Maxwell's equations, electron fluid equations and ion particle equations of motion, is presented as the first multi-dimensional application of this technology. We discuss its parallel implementation and demonstrate first results from three-dimensional simulations of compact plasma objects that have been out of reach of conventional codes. Potential applications of the new methodology to other scientific and engineering domains are also discussed.

1 INTRODUCTION

1.1 Overview of Asynchronous Techniques

Multiple temporal and spatial scales, occurring in large-scale physical systems, present enormous computational challenges. Numerical techniques for scientific and engineering applications commonly assume that the governing equations and simulation variables are properly discretized in space on a suitable spatial mesh (grid) composed of discrete control elements (also referred to as "cells"). A mesh can be either uniform or adaptive to spatial scales. The discretized equations are normally "time stepped", i.e. solution values in all mesh elements are synchronously updated in discrete time intervals determined by the choice of constant or variable *global* time increments.

In practice, however, physical systems can develop multiple time scales either due to multi-physics processes or spatial inhomogeneities. As a result, to avoid explosive numerical instabilities, explicit time-stepping integration techniques have to employ the minimum time-step size determined by the most restrictive condition in the system.

To address the above numerical difficulties, a number of asynchronous time-stepping approaches have been developed for flux-conservative and particle systems. For instance, Berger and Oliger (1984) proposed the adaptive mesh refinement (AMR) technique for block-structured meshes. In this technique, refined (daughter) meshes overlap regions covered by the coarser (parent) mesh so that the global (composite) mesh is made of a hierarchy of nested levels of logically rectangular patches.

Each patch is still updated with a constant time increment restricted by the patch cell size and the maximum velocity magnitude.

On the other hand, other researchers (e.g., Dawson and Kirby 2001) implemented local time stepping by allowing solution values in different mesh elements (cells) to be updated with different time increments, which are usually selected to be fractions of the global time-step size in order to satisfy local stability conditions. In most cases special care is taken to preserve flux conservation across cell interfaces in all updates. More recently a new time integration approach has been applied to equations of nonlinear elastodynamics (Lew et al. 2003). It is based on a discrete spacetime form of Hamilton's variational principle. This algorithm permits the selection of independent time steps in mesh cells so that local time steps do not bear an integral relation to each other. Time advance is done by organizing computational elements into a priority queue based on their precomputed update times. This approach permits updating each subsystem with a frequency dictated by its natural timescale, subject to solvability of the local time steps. However, it is only applicable to Hamiltonian systems in which the Lagrangian is expressible as a sum of component sub-Lagrangians. More general techniques (e.g., Abedi et al. 2004 and references therein) introduce an extra finite dimension to the computational model – a separate *temporal* mesh. A discontinuous Galerkin solution is constructed in time-space by adapting the duration of each element to the local degree of spatial mesh refinement.

To summarize, both single-step and multi-step techniques mentioned above fall into the category of *time-driven* integration, i.e. these techniques advance simulation variables by choosing time increments typically based on linear stability requirements. However, since the time-step size is not by itself indicative of magnitude of *change* to physical variables (Omelchenko and Karimabadi 2006a), these assumptions may easily violate *causality relationships* and as a result cause numerical instabilities in strongly nonlinear and multi-physics systems. The alternative concept to time stepping is discrete-event simulation (DES) (or equivalently event-driven simulation), well known in computer science (e.g., Banks et al. 2000, Fujimoto 2000). In physical systems an event associated with a particular variable results in a physically small but *finite* change to this variable (Nutaro et al. 2003, Karimabadi et al. 2005, Omelchenko and Karimabadi 2006a).

Event-driven techniques have recently been applied to molecular dynamics systems (Valentini and Schwartzentruber 2009) to enable asynchronous collisional models. However, their application to self-consistent electromagnetic PIC simulations is not as straightforward because of strong nonlinearities, physical constraints and tight synchronization requirements for coupled electromagnetic field and particle components.

1.2 Self-adaptive Discrete-Event Simulation

Explicit time-stepping techniques have two major drawbacks. First, they are not "intelligent" enough to predict *when* and *where* the underlying model will generate changes to its variables. As a result, they may do "idle number crunching" without generating any information or, to the contrary, miss important information by choosing inappropriately large time steps. In addition, using unnecessarily very small time steps often results in numerical diffusion. Second, asynchronously time-driven elements of a global system may be difficult to synchronize, as any dynamic modification of originally predicted multiple steps may lead to numerical instabilities and loss of parallel efficiency.

To resolve the above issues, a new *self-adaptive* approach to time integration of physical systems described by flux-conservative PDEs (Omelchenko and Karimabadi 2006a) and particle models (Omelchenko and Karimabadi 2006b) was proposed based on Discrete-Event Simulation (DES). Essentially, this new technology introduces the following principles to event-driven integration: (1) time increments for all physical variables, f ("states") are selected by imposing limiting bounds to their changes, Δf as per each update (the threshold Δf may be a function of f), (2) update events, scheduled to execute at their predicted time stamps, are allowed to be *preemptively* synchronized using application-

specific logics that enforces preservation of a given degree of physical (not temporal!) accuracy of numerical solution.

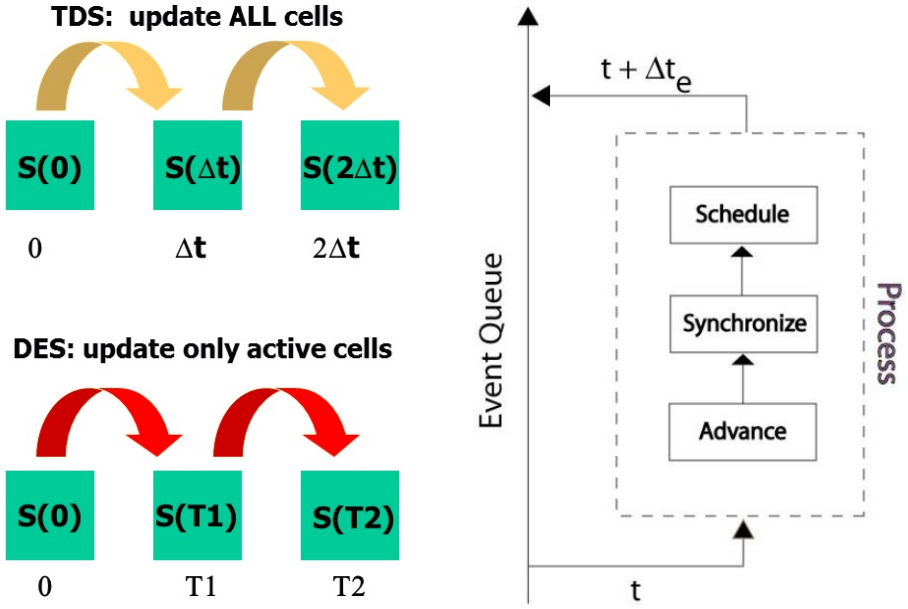

Figure 1: (LEFT) Comparison of time-driven (TDS) and self-adaptive event-driven (DES) simulations. DES automatically determines time levels where it processes only a required number of states. TDS synchronously updates all states with predetermined time steps. (RIGHT) Schematic of self-adaptive DES as introduced by Omelchenko and Karimabadi (2007). The event processing (1) advances a computational state by updating its discrete variables, (2) synchronizes this state with neighboring (dependent) states, and (3) reschedules a new event to take place in a predicted time interval, Δt_e.

There are three key aspects of this methodology (Figure 1): (i) event scheduling, (ii) preservation of discrete conservation laws, (iii) self-adaptivity. During *event scheduling* the DES algorithm predicts future updates of computational variables by assuming that their changes (e.g., magnetic field variations, particle displacements) should remain physically small in each update. *Preservation of conservation laws* (e.g., mass, divergence of magnetic field) is enforced by requiring that adjacent grid elements are updated at any time using identical common fluxes (usually defined at cell faces or cell edges). *Self-adaptivity* assumes that each cell synchronizes upon execution with its neighbors, which update themselves and in turn may be forced to synchronize with their own neighbors.

Let $df/dt = R(f)$ be a generalized governing equation for solution f at a given cell of a global mesh (Figure 2). The DES engine schedules and updates states asynchronously in an incremental fashion so that f undergoes a physically meaningful increment, Δf in each update. In other words, the DES code solves the event-driven analogue, $dt/df = 1/R(f)$. As a result, all parts of a multiscale physical system are advanced in time on their own time scales. Note that in our initial models (Omelchenko and Karimabadi 2006a, Omelchenko and Karimabadi 2006b) three processing stages shown in Figure 1 were applied to each event in a single call to the event processing function. However, in DES codes that use Preemptive Event Processing (PEP) (Omelchenko and Karimabadi 2007) these steps are performed on groups of selected events (see Section 2.2).

It should be emphasized that the self-adaptive DES is remarkably different from both time-stepped and other event-driven algorithms. First, the event scheduling procedure essentially serves only as the local "predictor" step in our algorithm. It sorts computational states by the estimated execution time and also guarantees that all events are predicted to be desynchronized in *f-space* by a reasonably small

physical quantity, Δf . Accordingly, the "corrector" step, applied to every *synchronized* state (executed as scheduled or preempted by PEP or the synchronization procedure), makes sure that during the entire simulation events do not become desynchronized by physical amounts larger than O(Δf). Essentially, the self-adaptive procedure detects significant (greater than Δf) deviations of solution trajectories, *f(t)* from their predicted values and *preempts* such events *ahead of* their scheduled execution times, forcing their states to trigger further synchronization calls to their neighbors.

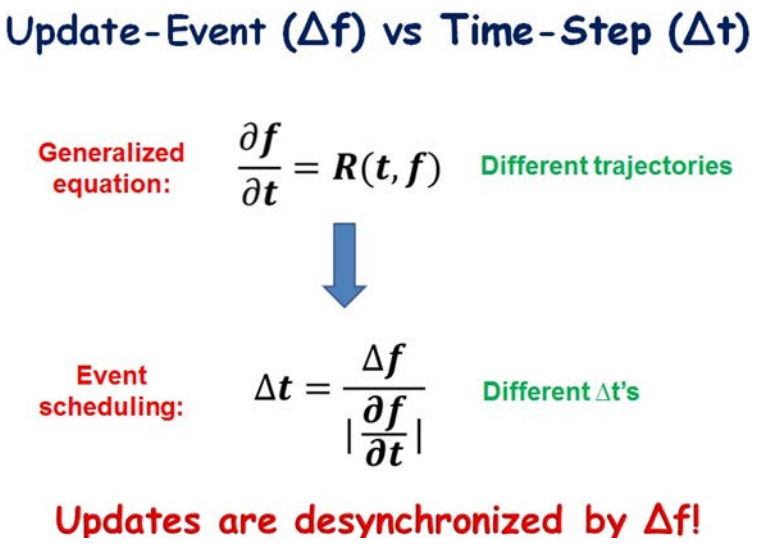

Figure 2: A generalized time evolution equation. The self-adaptive DES guarantees that events at any moment in the priority queue remain desynchronized by physical quantities not exceeding O(Δf).

The self-adaptive DES offers the following advantages over time-driven and conventional event-driven simulation techniques:

1. *Robustness (stability and accuracy)*. The integration algorithm exerts adaptive local control over the update rate of solution. This reduces integration errors associated with large time derivatives and prevents explosive numerical instabilities.

2. *Performance efficiency*. Spatially distributed physical quantities are updated *asynchronously*, in accordance with their natural temporal scales. This removes the global time step restriction and eliminates CPU overhead associated with idle computation.

We have applied the new technology to a number of diffusion-reaction-advection, gas dynamics models (Omelchenko and Karimabadi 2006a, Omelchenko and Karimabadi 2007) and plasma systems (Karimabadi et al. 2005, Omelchenko and Karimabadi 2006b), using discretization schemes with up to the second order accuracy both in space and time.

The rest of the paper discusses a self-adaptive event-driven integration algorithm implemented in a unidimensional hybrid code, HYPERS (Hybrid Particle Event-Resolved Simulator) (Omelchenko and Karimabadi 2012). Hybrid codes extend magnetohydrodynamics (MHD) by treating ion species as macro-particles, electrons as a neutralizing fluid, and by including the Hall term in the generalized Ohm's law (Winske et al. 2003). The HYPERS code differs entirely from other hybrid codes in the way simulation is performed. In lieu of time-stepping electromagnetic field and particle operations, it advances a closely coupled nonlinear plasma simulation via asynchronous events: numerical updates of discrete particles and fields. The paper is organized as follows. In Section 2 we briefly review the key principles of asynchronous plasma modeling and discuss PEP, which essentially generalizes the concept of time stepping. Section 3 describes the distributed data structures and parallelization strategy adopted in

HYPERS. In Section 4 we demonstrate results from first-ever asynchronous three-dimensional (3D) simulations of compact magnetized plasma objects (spheromaks). We summarize our conclusions and discuss other possible applications of our simulation approach in Section 5.

2 ASYNCHRONOUS HYBRID MODEL

2.1 Physical Events

HYPERS self-consistently solves for coupled radiation-free electromagnetic fields, fluid electrons and ion macro-particles on a block-structured computational grid, which covers a global Cartesian domain. In our asynchronous model we define two types of numerical updates: local (cell-level) and global (domain-level) events. Local events represent *asynchronous* updates of cell properties (e.g, local electromagnetic field components, particles, electron pressure). Global events represent domain-level operations that need to be executed synchronously throughout the whole computational domain (e.g., input/output procedures, single-timescale physical operations such as particle injection, etc).

The time evolution of the magnetic field, $\mathbf{B}$ is described by Faraday's law which is finite-differenced on a staggered (Yee's) grid in order to preserve the divergence-free property of $\mathbf{B}$:

$$\frac{\partial \mathbf{B}}{\partial t} = -c\nabla \times \mathbf{E}, \tag{1}$$

where c is the light speed and the electric field, $\mathbf{E}$ is found from the generalized Ohm's law $\mathbf{E} = \mathbf{E}\left(\mathbf{B}, n_i, p_e, \mathbf{u}_i, \nabla \times \mathbf{B}\right)$, written in the form of an algebraic function of the ion charge density, n_i and fluid velocity, $\mathbf{u}_i$ (obtained on the grid by scattering macro-particle properties to the grid nodes), magnetic field, $\mathbf{B}$, electron pressure, p_e and total current density, $\nabla \times \mathbf{B}$ (for more details see e.g., Winske et al. 2003).

At a given cell a field event forces a synchronous update of all face-centered components of magnetic field. Field events that are (1) executed as scheduled, or (2) preempted either by PEP or the self-adaptive event synchronization procedure, are called *synchronized* events. If, during the synchronization update, a field state's "trajectory", $\mathbf{B}(t)$ is found to have deviated significantly (by more than ΔB) from the predicted one, the corresponding event "wakes up" this state, i.e. forces it to further synchronize with its neighboring states. Note that states which fail the synchronization test are updated but not rescheduled.

Macro-particles (representing multiple species of ions) are advanced in space through the governing Newton-Lorentz equations:

$$\frac{d\mathbf{v}_p}{dt} = \frac{q_i}{m_i}\left(\mathbf{E}_p^* + \frac{\mathbf{v}_p \times \mathbf{B}}{c}\right), \quad \frac{d\mathbf{r}_p}{dt} = \mathbf{v}_p, \tag{2}$$

where $\mathbf{r}_p, \mathbf{v}_p$ are the particle positions and velocities, respectively, q_i, m_i are the ion charge and mass, and $\mathbf{E}_p^*$ is the effective electric field that includes a collisional operator consistent with the Ohm's law.

A particle event at a given cell involves either (1) advancing a number of internal particles with close time stamps ("push" times), or (2) processing incoming particles. A particle state is not rescheduled if the minimum local (to the cell) particle time stamp does not change during its update.

It should be emphasized that the enhanced robustness of asynchronous field integration, as compared to traditional (time-stepping) solvers, results from *self-adaptivity of local field events*: processed states automatically trigger additional updates if their "trajectories" are found to have deviated from original estimates made at the scheduling time.

2.2 Preemptive Event Processing (PEP)

Parallelization of event-driven codes is not as straightforward compared to time-stepped applications. Traditional discrete-event applications rely on a number of deterministic and optimistic synchronization

schemes when running on parallel systems (Fujimoto 2000). Deterministic algorithms execute events always in timestamp-increasing order. This typically results in too few CPU operations per global synchronization. Optimistic strategies allow processors to execute their local events ahead of the global simulation time but they suffer major performance penalties in situations where rollbacks of parts or the whole of the global system need to be performed (Tang et al. 2006).

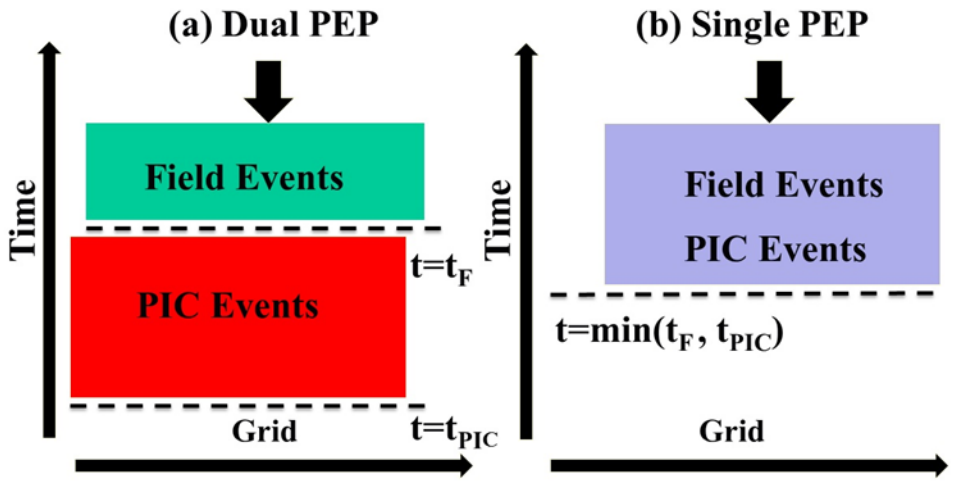

Figure 3: Two modes of preemptive event processing used in HYPERS: (a) the dual-queue PEP procedure processes field and particle events at separate time levels, (b) the single-queue PEP procedure performs field and particle operations at common synchronization times.

To enable efficient execution of physics-driven DES models on massively parallel, distributed memory computer architectures we have developed a new approach (Omelchenko and Karimabadi 2007, Omelchenko and Karimabadi 2012) – Preemptive Event Processing (PEP) (Figure 3). The fundamental difference of this algorithm from other synchronization strategies used in conventional discrete-event simulations is the forced synchronization of events *prior to* their scheduled times. The synchronization of time-distributed physical updates is achieved by projecting sufficiently close (in *f*-space) *future events* to a given moment of time where they can be effectively executed in parallel (see Figure 3). The width of the projection "window" is a free parameter usually taken to be of order the difference between two successive PEP levels (Omelchenko and Karimabadi 2007).

PEP can be in effect thought as a *generalization* of time stepping since it results in updating the global system state at discrete, irregularly spaced (self-adaptive) time intervals. The significant difference of PEP in regard to regular time stepping, however, is that at any given time PEP adaptively selects and processes only *a fraction of* all variables available in the system. Naturally, when all variables in the system evolve on *a single time scale*, PEP synchronizes updates in a similar manner to time stepping.

The parallel efficiency of PEP depends on (1) a choice of domain decomposition, (2) a trade-off between parallel speedup and overhead incurred by communication and event handling. The original PEP algorithm proposed by Omelchenko and Karimabadi (2007) employs on each processor a single event queue used for scheduling events of all types. In the single-queue PEP (Figure 3b) global synchronization time levels always match the earliest current event timestamp available in the system, with all events being executed within a common time window determined by the fastest time scale. In HYPERS, however, the PEP procedure defines separate queues for particle and field events, effectively implementing an asynchronous version of "physical subcycling", which makes possible efficient parallel processing of disparate field and particle events on their characteristic time scales. This dual-queue PEP chooses generally different global synchronization times for executing particle and field operations, as shown in Figure 3a.

3 PARALLEL ALGORITHM

3.1 Data Structures and Communication Strategies

In HYPERS the global computational domain can be decomposed into an arbitrary number of disjoint rectangular boxes ("subdomains") which can be assigned to participating processors in an arbitrary manner (Figure 4). Complex geometries are currently approximated with staircase boundaries. Each subdomain is surrounded by a layer of ghost cells. Subdomains with common interfaces exchange data via message passing *regardless* of whether they reside on the same or different processors. Field data at ghost cells are either updated by applying local boundary conditions or received from corresponding border states, which are interior to adjacent subdomains. Particles are transferred between subdomains from ghost to border cells, respectively.

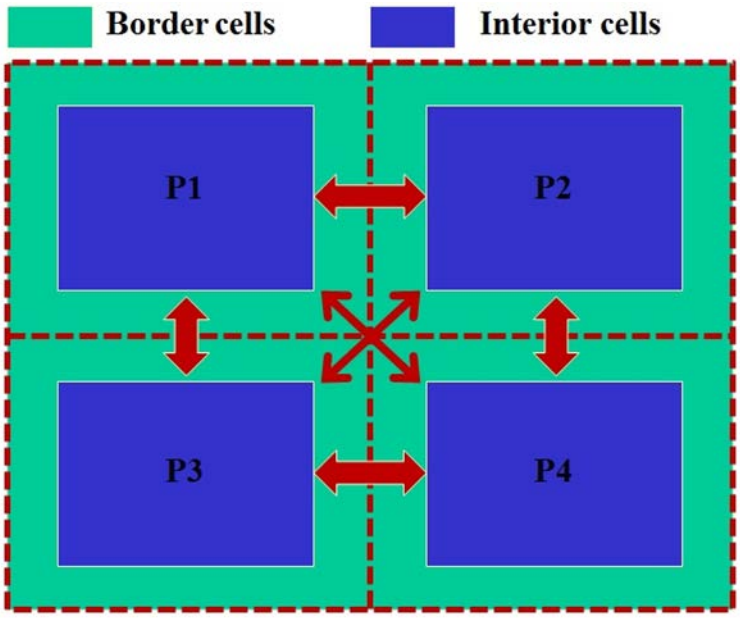

Figure 4: An example of a 4-processor parallel domain decomposition. The dashed lines indicate processor boundaries. The parts of the computational domain that require synchronization are colored green.

All inter-subdomain communications are implemented via the same abstract programming interface (API). Under the "hood", local (shared memory) operations are implemented via simple memory copying, while remote (distributed memory) data transfers are performed with MPI. Both cell-level and block-level communications are supported. In addition to facilitating the programming of data transfer operations, the parallel communication API allows effective debugging of asynchronous runs.

It should be noted that in a distributed-memory environment the self-adaptive (wakeup-based) field-event synchronization algorithm may result in triggering a sequence of synchronization updates across processor boundaries. This procedure, if not modified, may entail multiple rounds of message passing across the global system. To prevent this scenario, we apply a simple and effective solution: at each PEP level all processors force *all of its border states* to probe for their possible "overflow" (i.e. a model-specific synchronization condition) *before* messages passed by interior (border) states are delivered to their destination (ghost) states. As a result, when a synchronization message arrives at a destination processor upon the completion of the PEP step, it does not have to generate any new synchronization calls.

HYPERS effectively enables flexible restart and post-processing capabilities which go beyond two parallel output modes commonly used in scientific computing, i.e. (1) local data output (one file per each

processor), or (2) global output (a single file). The HYPERS I/O interface is designed to enable scalability of I/O operations and allow restarts with numbers of processors varying from one check point to another throughout a simulation. This is achieved by partitioning the global computational domain into an arbitrary number of "restart server" subdomains for I/O purposes (these subdomains in general differ from processor-assigned "primary" subdomains). Data belonging to these spatial blocks are saved in parallel to separate files. Currently the restart server configuration is kept constant during the simulation. However, the simulation can proceed from one restart point to another with different primary domain decompositions and processor assignments. This capability is expected to enable load-balanced discrete-event simulations and effective visualization of large data sets in the future.

HYPERS uses special data "registers" (linked lists) to keep track of particle and field states updated at each PEP level: *PPEPStack* and *FPEPStack*, respectively. Cell-level communications of ghost particle and border field states are facilitated by linking them into lists, *SentGhostList* and *SentBorderList*, respectively.

3.2 Parallel Control Flow

At startup/restart the magnetic field and particle data are initialized (or read in) at all interior cells of the computational domain. Each processor initializes *its own data set*. After applying non-periodic boundary conditions and depositing particle moments, the hybrid PIC system is finalized by performing the following steps:

1. Border cells communicate magnetic field components, $\mathbf{B}$ to corresponding ghost cells. Partial particle moments, stored at the subdomain border interfaces, are communicated to neighboring processors where they are summed up to obtain full moments.
2. Electric fields, $\mathbf{E}$ and magnetic field time derivatives, $\partial\mathbf{B}/\partial t$ are computed. All field timestamps are initialized to the initial time. The field and particles states are placed in the *FPEPStack* and *PPEPStack* registers, respectively.

Following the initialization stage, the parallel cycle proceeds as follows until the finish time is reached:

1. States from the *FPEPStack* and *PPEPStack* lists predict their next timestamps and schedule themselves into the corresponding event queues.
2. Partial particle moments are restored at the subdomain interfaces.
3. The PEP step is performed. It may result in processing either particle or field events, (dual-queue PEP) or both types of events at the same time (single-queue PEP). Updated field and particle states are stored in the corresponding state registers, with field states being marked as *synchronized* or simply *updated*.
4. Field states at border cells are preemptively probed for synchronization conditions.
5. Communication states found in the PEP registers send messages to their destination states: ghost particle states transfer particles, border field states transfer only their update status. On arrival to their destination states, particles deposit moments and their new states are placed into *PPEPStack* for rescheduling. On destination processors ghost field states trigger updates in neighboring cells. Those field states, in turn, are placed into *FPEPStack* .
6. At the subdomain interfaces partial particle moments are stored and communicated to neighboring subdomains. New net moments are obtained by their summation.

7. New electric fields are computed and magnetic field time derivatives are recomputed using these electric fields. Border field states found in *FPEPStack* transfer the updated field data (magnetic fields, their time derivatives, timestamps) to their ghost destinations.

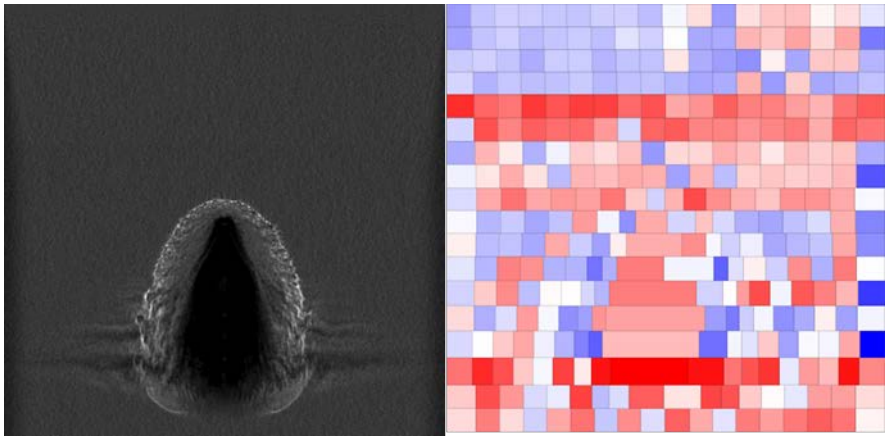

Figure 5: An example of load balanced dynamic domain partitioning. (LEFT) An instantaneous total CPU load distribution from a 2D magnetospheric HYPERS run. (RIGHT) a18x18 CPU rectangular domain decomposition obtained with a jaggered partitioning technique.

3.3 Load Balancing Issues

The parallel HYPERS performance has been tested using up to several hundred processors with 1D,2D,3D domain decompositions. The scalings observed were found to be consistent with predictions obtained through comparative analysis of processor load distributions. The PEP synchronization algorithm allows to merge different in physics and disparate in simulation time calculations into a single or dual stream of calculations. Effectively this allows to introduce a concept of total computational load per processor as a function of simulation time. Given this measure of total local load, we can dynamically construct load maps and compute optimum partitions using various partitioning algorithms (Figure 5).

As mentioned above, the HYPERS architecture already enables arbitrary block-structured domain decompositions and processor configurations on restarts. This capability is a critical prerequisite for implementing appropriate dynamic load balancing and runtime tuning techniques, in particular those that are based on m-way jagged partitioning and relaxed-hierarchical partitioning (Omelchenko et al. 2010). Scalability studies dealing with dynamic load balancing will be highlighted elsewhere.

4 SPHEROMAK SIMULATIONS

One of the most challenging goals in plasma physics is to understand the physical links between turbulence, coherent plasma structures, reconnection and dissipation in strongly nonlinear regimes. In laboratory this can be done through studying dynamics of fast evolving “plasma plumes” initially created as compact spherical magnetized plasma objects (“spheromaks”). A single spheromak or two double-sided spheromaks can be created at the ends of an experimental cylindrical device and pushed forward at varying speeds. Their dynamics and merging produce small-scale cells of turbulence as well as generate shock structures.

Large-scale laboratory plasmas are extremely difficult to model with standard hybrid codes due to high computational costs and severe numerical difficulties encountered in the form of explosive instabilities. These instabilities arise due to uncontrolled growth of high-frequency oscillations in low-density plasmas immersed in strong magnetic fields.

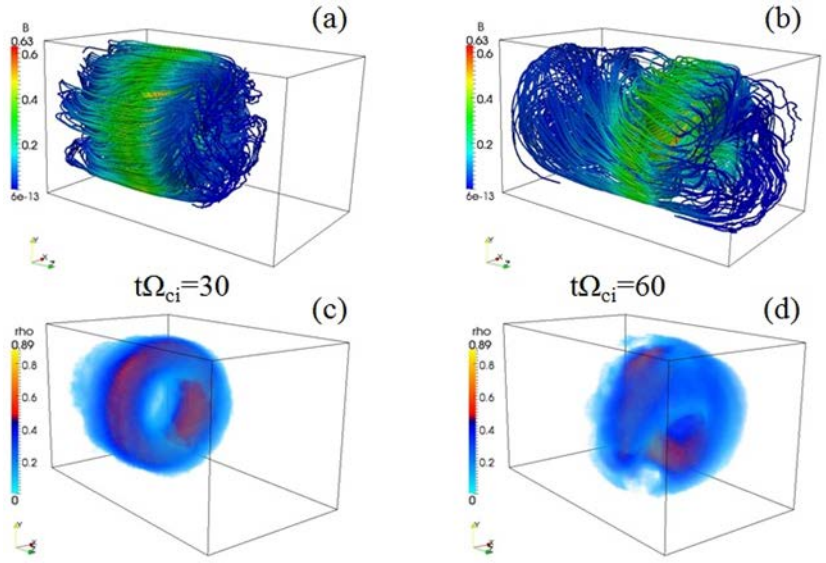

Figure 6: Relaxation of a single spheromak: (a,b) magnetic field lines, (c,d) plasma density. The initially toroidally symmetric plasma (a,c) relaxes to a helix (b,d).

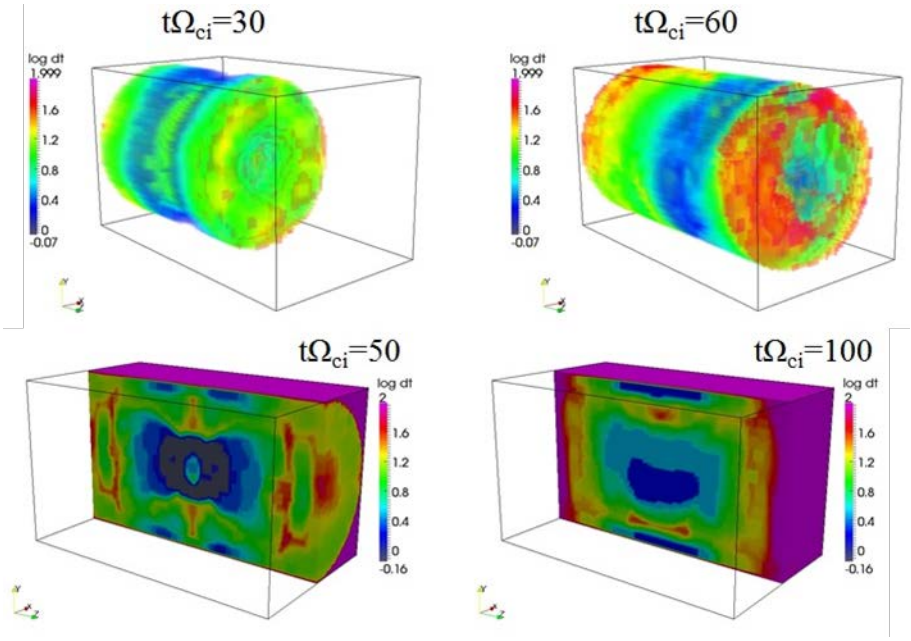

Figure 7: Multiscale distributions of local field time steps in event-driven hybrid spheromak simulations. Top panels (iso-surfaces) show the time evolution of a single spheromak. Bottom panels (volume rendering) show the merging of two spheromaks.

Below we describe a first-ever long-time hybrid simulation of such systems made possible by the self-adaptive nature of the HYPERS solver. Note that these results were obtained in a parallel simulation involving only 4 dual-core processors of a single workstation. Analogous time-stepped hybrid simulations, however, not only would require *hundreds of CPUs*, but would not be guaranteed to run *stably*. Here we are primarily concerned with computational aspects of our simulations. Detailed discussion of the underlying physics will be given elsewhere.

Figure 6 shows the time evolution of a single spheromak initialized with zero plasma flow velocity, uniform density and a small toroidal magnetic field perturbation. As theoretically predicted and experimentally confirmed, the spheromak becomes unstable to an ideal MHD tilt mode and quickly relaxes to a stable helical structure.

Figure 7 illustrates the capability of the HYPERS solver to select adaptive time steps for local electromagnetic fields. Note that field time steps in our simulations differ by almost two orders of magnitude, with the smallest ones dynamically arising in regions of low density and high magnetic fields. These values are constrained by both local Courant conditions and accuracy considerations. Similar maps can be constructed for grid-averaged particle time steps as well (Figure 8).

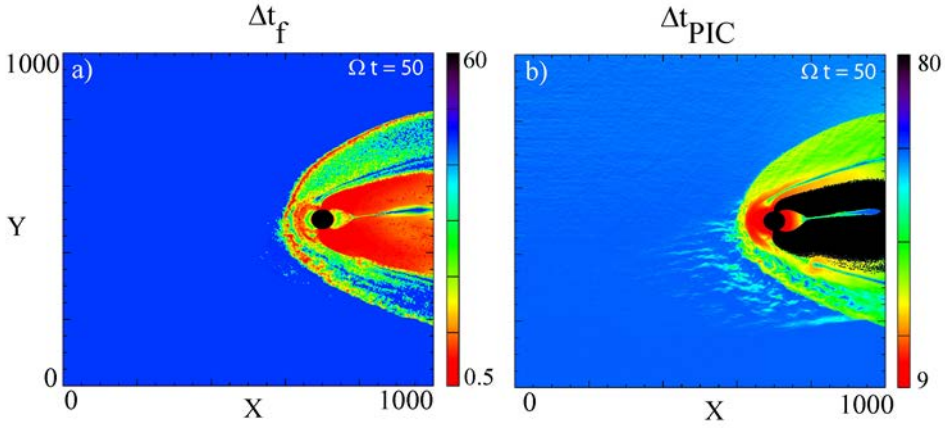

Figure 8: Local time steps used for updating (a) fields and (b) particles in 2D magnetospheric simulations by Omelchenko et al. (2014). Smaller particle time steps in panel (b) indicate stronger magnetic fields and wave-particle interactions resulting in higher particle energies. Black colored regions contain no particles.

5 CONCLUSION

In this paper we have reported the most recent advances in global multiscale plasma simulations made possible by the new computational technology implemented in a parallel hybrid PIC code, HYPERS. This code differs from all existing simulation codes *in the way simulation is performed*. The numerical formalism developed here is quite general and expected to find adoption in a variety of scientific and engineering fields, where multiple time scales and nonlinear phenomena continue to present fundamental computational challenges. For instance, large-scale PDE-based models with multiple time scales require enormous resources in terms of CPU time if tackled using traditional time-stepping algorithms. Due to memory constraints the current linear solver (implicit) technology sets a limit on the size of the problems solved on irregular meshes. The application of the self-adaptive discrete-event methodology to matrix-free solvers on large irregular grids will enable accurate solution of general time-dependent hyperbolic and parabolic PDEs presently beyond reach. Examples of such applications include magnetosphere and climate models, propagation of seismic waves and electromagnetic fields in complex geological formations, oil and gas transport in porous media, next-generation particle accelerators with tight tolerances, light and signal propagation in biological tissues, to name a few.

ACKNOWLEDGMENTS

This work was supported by the DOE grant DE-SC0010528 and the National Science Foundation grants 0529919, 0539106, 0613410, 0903726 at SciberQuest, Inc.

REFERENCES

Abedi R., S.-H. Chung, J. Erickson, Y. Fan, M. Garland, D. Guoy, R. Haber, J.M. Sullivan, S. Thite, Y. Zhou. 2004. "Spacetime meshing with adaptive refinement and coarsening." In *Proceedings of 20th Annual Symposium on Computational Geometry*, Brooklyn, New York, USA, 300-308.

Banks, J., J. S. Carson, B. L. Nelson, and D. M. Nicol. 2000. *Discrete-Event System Simulation*. 3rd ed. Upper Saddle River, New Jersey: Prentice-Hall, Inc.

Berger M.J. and J. Oliger. 1984. "Adaptive mesh refinement for hyperbolic partial differential equations" *J. Comp. Phys*. 53: 484 .

Dawson C. and R. Kirby. 2001. "High resolution schemes for conservation laws with locally varying time steps." *SIAM J. Sci. Comput*. **22** (6): 2256.

Fujimoto, R.M.. 2000. *Parallel and distributed simulation systems*. Wiley Interscience.

Karimabadi H., J. Driscoll, Y.A. Omelchenko, and N. Omidi. 2005. "A new asynchronous methodology of modeling of physical systems: breaking the curse of courant condition." *J. Comp. Phys*. 205: 755-775.

Lew A., J.E. Mardsen, M. Ortiz, and M. West. 2003. "Asynchronous variational integrators." *Arch. Rational Mech. Anal.* 167: 85.

Nutaro J., B.P. Zeigler, R. Jammalamadaka, and S. Akerkar. 2003. "Discrete event simulation of gas dynamics within the DEVS framework." *Lecture Notes on Computer Science* 2660: 319.

Omelchenko Y.A., H. Karimabadi. 2006a. "Self-adaptive time integration of flux-conservative equations with sources." *J. Comp. Phys*. 216: 179-194.

Omelchenko Y.A., H. Karimabadi. 2006b. "Event-Driven, Hybrid Particle-in-Cell Simulation: A New Paradigm for Multi-Scale Plasma Modeling." *J. Comp. Phys.* 216: 153-178.

Omelchenko Y.A., H. Karimabadi. 2007. "A time-accurate explicit multi-scale technique for gas dynamics." *J. Comp. Phys.* 226: 282-300.

Omelchenko Y.A., H. Karimabadi, E. Saule, and U. V. Catalyurek. 2010. "Parallel Event-Driven Global Magnetospheric Hybrid Simulations." In *AGU Fall Meeting Abstracts*, page A1756.

Omelchenko Y.A., H. Karimabadi. 2012. HYPERS: "A unidimensional asynchronous framework for hybrid simulations." *J. Comp. Phys.* 231: 1766-1780.

Omelchenko, Y.A., H. Karimabadi, H.X. Vu 2014. "Advances in multiscale simulations of solar wind interactions with the Earth's magnetosphere." *ASP Conference Series*, Numerical Modeling of Space Plasma Flows, 488: 155-160.

Tang T.R., K.S. Perumalla, R.M. Fujimoto, H. Karimabadi, J. Driscoll, and Y.A. Omelchenko. 2006. "Optimistic simulations of physical systems using reverse computation." *SIMULATION, Tran. Soc. For Modeling and Simulation Intern* 82: 61-73.

Valentini P., Schwartzentruber, T.E. 2009. "A combined Event-Driven/Time-Driven molecular dynamics algorithm for the simulation of shock waves in rarefied gases." *J. Comp. Phys.* 228: 8766-8778.

Winske, D., L. Yin, N. Omidi, H. Karimabadi, and K. Quest. 2003. "Hybrid simulation codes: past, present and future – a tutorial." *Lecture Notes in Physics* 615: 136, Springer-Verlag.

AUTHOR BIOGRAPHIES

YURI A. OMELCHENKO is Chief Architect at SciberQuest, Inc. in Del Mar, CA. He received his M.S. and Ph.D. in Plasma Physics from the Moscow Institute of Physics and Technology in Moscow, Russia. His research interests lie in plasma physics, numerical algorithm development and high-performance computing. His email address is omelche@gmail.com.

HOMA KARIMABADI is Chief Scientist at SciberQuest, Inc. in Del Mar, CA. He holds a M.S. and Ph.D. in Plasma Astrophysics from the University of Maryland, USA. His research has spanned a wide variety of areas including high-performance computing, intelligent algorithms, scientific visualization, plasma physics, among others. His email address is homakar@gmail.com.