Parallel Scientific Computing on the IBM SP2 at Rensselaer's Scientific Computation Research Center

B. K. Szymanski, E. Deelman, J. E. Flaherty, C. D. Norton, J. D. Teresco and L. H. Ziantz

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY 12180
email: {szymansk,deelmane,flaherje,nortonc,terescoj,ziantzl}@cs.rpi.edu

Abstract

Various research applications using the adaptive solution of partial differential equations and parallel discrete event simulation as well as the investigation of object oriented programming paradigms are described. This work utilizes the IBM SP2 at Rensselaer's Scientific Computation Research Center (SCOREC). The software is designed using C, C++ and Fortran 90, together with MPI. Tools described include a parallel mesh database, load balancers and run-time code optimizers. Additionally, using plasma codes as an example, we compare compilers and architecture performance of the SP2 to competing distributed memory parallel machines. Applications studying finite element analysis of problems in biomechanics and fluid dynamics, the simulation of fusion plasma processes and the spread of Lyme disease are presented.

1 Adaptive Finite Element Methods

The finite element method (FEM) has become a standard engineering analysis tool for solving partial differential equations (PDEs). Adaptive FEMs have gained importance because they provide greater accuracy, reliability, robustness, and time and space efficiency. In such an analysis, the computational domain is first discretized to create a mesh. During the solution process, portions of the discrete domain are spatially refined or coarsened (h-refinement), the method order is varied (p-refinement), and/or the mesh is moved to follow evolving phenomena (r-refinement), each of which concentrates the computational effort in regions where the solution resolution is inadequate [1].

In order to solve large problems within a reasonable period of time, these methods have been implemented on parallel computers. While parallelism can yield faster solution times or greater accuracy in the same computing time, there are complications such as coordinating interprocessor communication and managing distributed data structures. The standard methodology for optimizing parallel FEM programs relies on the initial partitioning of the meshes involved in the computation. However, in parallel adaptive codes, a good initial partitioning is not sufficient to assure high performance throughout the computation. Load imbalance caused by adaptive enrichment necessitates a dynamic redistribution of data. Our focus is to build an efficient load balancer for parallel adaptive methods that can be used to solve complex problems. The tools that have been developed at Rensselaer Polytechnic Institute to facilitate this focus are described in the following sections.

Problem Areas under Investigation Current areas of research involving adaptive solutions to PDE's include rotorcraft aerodynamics, structural acoustics, biomechanics, composite media, and crystal growth processes. One of the biomechanics projects is striving to develop the high-performance computational tools needed to make significant advances towards understanding the complex biomechanical behavior of normal and pathological human joints, which will eventually lead to superior clinical treatments. Another biomechanics project involves the development of a high-performance problem-solving environment for optimization and control of chemical and biological processes. Such optimization requires the repetitive solution of time-dependent PDEs in two and three dimensions. Prototype models for tissue-engineering applications such as the control of wound contraction and for the engineering design of a bioartificial artery are under investigation.

SCOREC Mesh Tools The system we are using is built upon the *SCOREC Mesh Database* (MDB) [2] which provides a hierarchical representation of a finite element mesh. It also includes a set of operators to query and update the mesh data structure. The basic mesh entity hierarchy consists of three-dimensional *regions*, and their bounding *faces*, *edges*, and *vertices*. In our case, each region is a tetrahedral finite element. Mesh entities are explicitly classified relative to a geometric model of the problem domain. This classification assures that the mesh remains valid after h-refinement.

Meshes are created using the *SCOREC Finite Octree Automatic Mesh Generator* [3]. Beginning with a geometric model of the domain obtained from CAD software, the mesh generator creates a variable level octree that is consistent with the triangulation on the boundary of the domain. Octants are classified as interior, exterior, or boundary. Exterior octants receive no further consideration while interior octants are discretized using templates. Face removal procedures are used to connect the boundary triangulation to the interior octants.

Mesh enrichment [3] performs spatial (h-) refinement and coarsening in parallel using error indicator information and enrichment threshold values. From this information, edges are marked to be coarsened, refined, or to remain untouched. Mesh enrichment is done in several stages in the following order: 1) coarsening, 2) mesh optimization (optional), 3) refinement, 4) mesh optimization (optional), 5) refinement vertex snapping, and 6) mesh optimization (optional). Coarsening is edge based and collapses a marked edge to one of its end vertices. Regions that are connected to the collapsed vertex that cease to exist are deleted to form a polyhedral cavity, and the faces of the cavity are connected to the target vertex to form new mesh regions. Mesh optimization improves the quality of triangulations with respect to a given criterion (e.g., element shape). Refinement is performed using subdivision patterns. First, faces on partition boundaries with marked edges are triangulated using two-dimensional refinement patterns. Each processor then independently applies three-dimensional templates that have been determined for every configuration of marked edges to eliminate the need for over refinement. In stage 5 of the enrichment process, vertices created by the refinement process on a curved model boundary must be "snapped" to the appropriate model entity to ensure mesh validity.

The Parallel Mesh Database A *Parallel Mesh Database* (PMDB) [4], which provides operators to create and manipulate distributed meshes, is built on top of the MDB. Once an adaptive FEM calculation begins, it consists of alternating phases of computation, mesh enrichment, and load balancing. PMDB data structures and operators are used in all stages of this process.

When a mesh is distributed using PMDB, each processor holds a MDB mesh which is a subset of the complete one. In a parallel environment, entities along partition boundaries are shared by more than one processor (Figure 1, left), so a partition boundary data structure is maintained to store links that point to the location of each shared entity. Duplicated partition boundary entities have unique owner processors. PMDB provides fast query and update operations on this structure which are used by other SCOREC software. For example, a finite element solver can obtain scatter/gather maps of vertices to be used in its communication phase. Other information available include processor adjacencies and lists of off-processor duplicates of an entity. Fast traversal of entities on interprocessor boundaries is also provided. All interprocessor communication is done using the IBM implementation of the Message Passing Interface (MPI) [5].

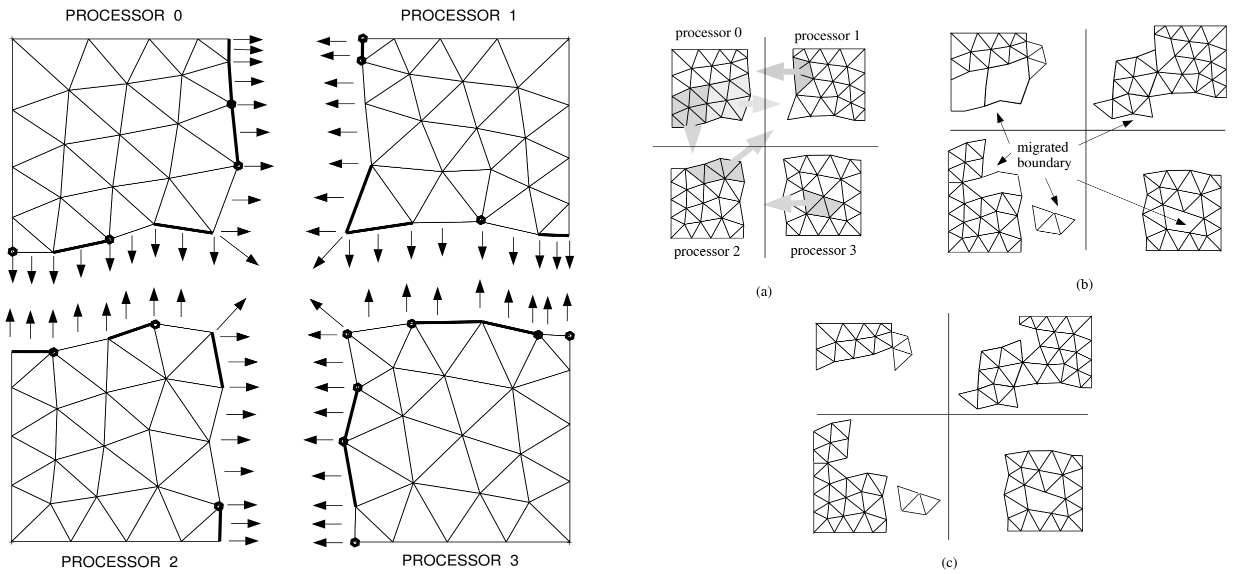

Figure 1: Two-dimensional example of a distributed mesh. Arrows represent interprocessor pointers between boundary entities. Heavy lines and points indicate the unique owner of each shared boundary entity (left). Example of arbitrary multiple migration (right).

Since meshes change during an analysis, portions will need to be moved from one processor to another to maintain load balance. This capability is provided by PMDB, which allows arbitrary multiple migration of regions between processors (Figure 1, right). Any top-level entity can be marked for migration, although it is frequently the boundary elements that are migrated. As shown in Figure 1, mesh migration is done in three main phases: (a) the senders migrate mesh entities to the receivers, (b) the senders and receivers send the migrated boundary to the owners, and (c) the owners update the boundary data structures and notify affected processors of the new location of each entity.

Routines for partition quality analysis are also in PMDB to produce qualitative (visual) and quantitative measurements [6]. Quality metrics include a ratio of the number of mesh faces on interprocessor boundaries to the total number of faces in the mesh, a measure of the average number of processors that need to exchange data during the solution stage, and the number of connected components within a partition. Capabilities to visualize mesh partitionings, components of the solution, error indicators, and mesh quality statistics using IBM Data Explorer are available.

Partitioning and Rebalancing There are three static partitioners available within the system. *Orthogonal Recursive Bisection* (ORB), also known as Recursive Coordinate Bisection, uses the coordinates of the mesh vertices to partition the mesh. At each recursive step, the cartesian coordinate of longest dimension of the domain is bisected, vertices are sorted according to the bisecting coordinate, and half of the vertices are assigned each subdomain. *Inertial Recursive Bisection* (IRB) repeatedly bisects the mesh orthogonal to a principal axis of inertia of the domain. *Recursive Spectral Bisection* (RSB) [7] is generally considered as one of the best static mesh partitioners. However, RSB can be computationally costly, and is usually prohibitively expensive when used with adaptive computation. Static versions of ORB and IRB are available in PMDB while RSB is available in the Chaco [8] package.

Partitioning can become imbalanced at each adaptive step of the computation. Since the mesh is distributed, the rebalancing algorithm must be parallel. Several dynamic load balancers are available and make use of PMDB data structures and mesh migration operators. *Iterative Tree Balancing* (ITB) [4] migrates entities to neighboring partitions to achieve balance. *Parallel Sort Inertial Recursive Bisection (PSIRB)* [3] is a parallel repartitioner based on the IRB static partitioner. The *Octree partitioner* [9] makes use of the octree structure underlying the mesh to achieve load balance.

The first step in a dynamic rebalancing is to determine the measure of imbalance. This requires a "cost function" that reflects the computational load on each processor. With h-refinement, it suffices to choose the number of elements on a processor as a cost function; however, p-refinement would necessitate inclusion of an element's order. If the solution algorithm employs spatially-dependent time steps then, typically, a greater number of smaller time steps must be taken on smaller elements and this must also be reflected in the cost. Predictive load balancing methods, currently an area of research, require elements to be weighted by the expected cost at some future time in the computation. PMDB provides an element weighting scheme which can be used to address each of these needs.

Like Leiss and Reddy [10], lightly loaded processors request load from their most heavily loaded neighbors in ITB. However, instead of considering an immediate neighborhood of processors, the algorithm views the requests as forming a forest of trees. Each tree is then linearized, and a logarithmic-time scan operation is used to compute load flows on each tree. The amount of data to be migrated is determined by these flows. Single layers of regions on interprocessor boundaries are moved from heavily loaded to lightly loaded processors based on the load to be migrated. This process is repeated until the load becomes balanced to within a tolerance, or until the maximum number of iterations has been performed. The iterative migration can also be run periodically with only a few iterations to help maintain balance between substages of an operation like mesh enrichment (cf. [4]) without a large run-time penalty.

PSIRB will produce the same partitions as the static IRB partitioner, but the coordinates in the inertial frame are sorted in parallel, allowing the algorithm to be used as a dynamic repartitioner. Thus, the mesh can be distributed before PSIRB is performed.

The Octree partitioner creates a one-dimensional ordering of the octree and divides it into nearly equal-sized segments based on tree topology. The first step of the algorithm is the determination of cost metrics for all subtrees. The second phase of the partitioning algorithm uses the cost information to construct the actual partitions. Since the number of partitions is prescribed and the total cost is known from the first phase, the optimal size of each partition is also known. Partitions consist of a set of octants that are each the root of a subtree and are determined by a truncated depth-first search. Thus, octree nodes are visited in depth-first order, and subtrees are accumulated into successive partitions. The subtree rooted at the visited node is added to the current partition if it fits. If it would exceed the optimal size of the current partition, a decision must be made as to whether it should be added, or whether the traversal should examine it further. In the latter case, the traversal continues with the offspring of the node and the subtree may be divided among two or more partitions. When the excess at a node is too large to justify inclusion in the current partition, and the node is either terminal or sufficiently deep in the tree, the partition is closed and subsequent nodes are added to the next partition.

Figure 2 shows the partitionings of the mesh about half of a cone (see the **Example** section) onto 8 processors using IRB and Octree partitionings.

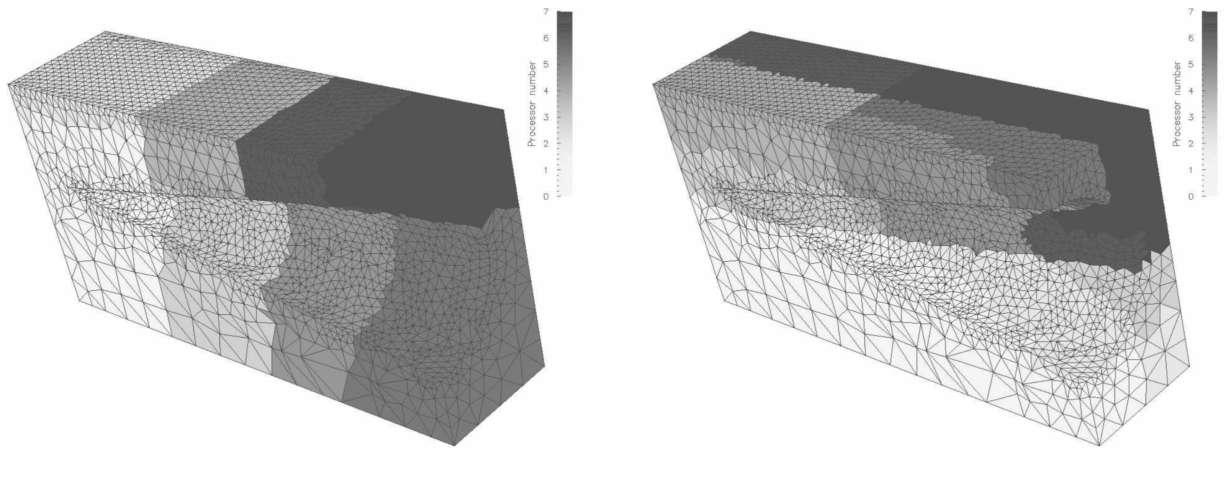

Figure 2: Initial "Half Cone" meshes distributed onto 8 processors by PSIRB (left) and Octree (right) partitionings. Shading indicates processor assignments.

Example Consider a Mach 2 shock hitting a cone with a 10-degree half angle from its side. The domain is a box surrounding the cone with one end at the base of the cone and the opposite beyond the tip of the cone. The aim is to examine the transient behavior of the shock as it traverses the cone. The flow is governed by the Euler equations [9]. Using symmetry, we solve for the flow about one half of the cone with an initial mesh containing 28,437 elements. The domain is quiescent initially, and a Mach 1.25 flow behind the Mach 2 shock enters the top of the flow domain (see Figure 3). Reflected conditions are applied on solid surfaces, planes of symmetry, and on the sides of the problem domain. Outflow conditions apply at the bottom face.

The transient analysis begins with a time step being attempted by the solver using the initial mesh with the initial conditions and boundary conditions. Each element has a maximum allowable time step size based on the Courant condition [9]. The minimal acceptable time step is selected as a global increment. The time step is either accepted or rejected based on an error indicator, which is currently the density gradient across element faces scaled by the element volume. Scaling forces smaller elements to have a lower error indicator. If the maximum elemental error indicator exceeds a tolerance, the time step is rejected and adaptive mesh refinement and coarsening take place. In a transient analysis, coarsening is essential to keep mesh sizes manageable since regions of interest move as the analysis proceeds. The solution is interpolated to the new mesh and the integration proceeds with a new time step. If the maximum error indicator is below the tolerance, the time step is accepted, the global time is incremented appropriately and another time step is attempted. The entire process is repeated until reaching a terminal time.

The transient flow problem was solved in parallel on 16 processors of an IBM SP2 computer. Figure 3 shows the flow density at global times $t = 0.1$ and $t = 0.3$. Away from the cone, the shock travels without appreciable disturbance, but the density increases significantly as the shock diffracts in regions near the cone.

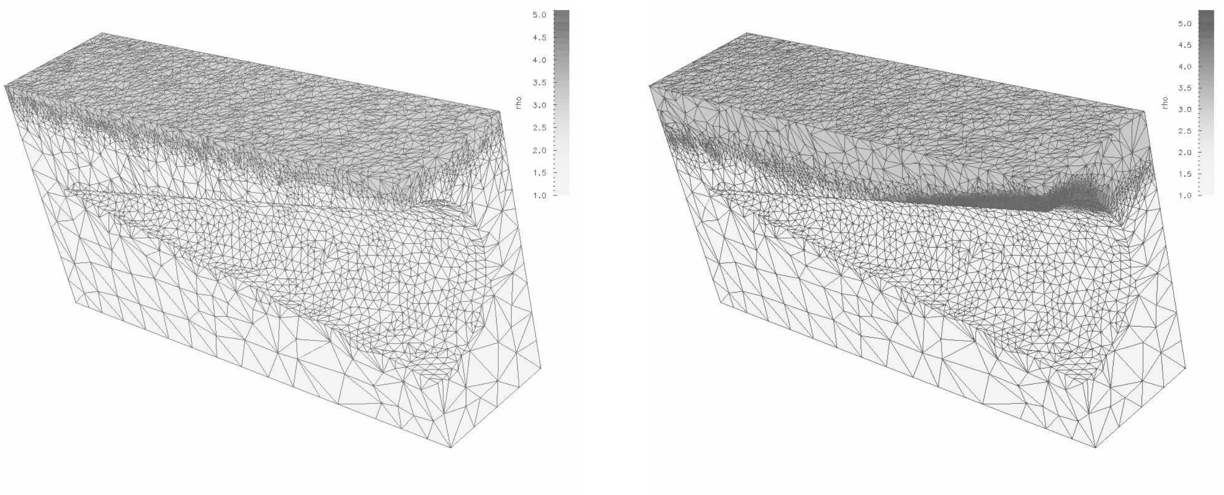

Figure 3: "Half Cone" meshes at times $t = 0.1$ (left) and $t = 0.3$ (right). Shading indicates values of the density on each element.

2 Object Oriented Paradigms in Scientific Computing

The prominent success of high performance computing used in modeling scientific, engineering and physical phenomena, continually motivates the development of very ambitious applications. In most instances, programming languages such as Fortran 77 and C have been used, but many of the most challenging applications are stretching the limits of these languages. Many problems in science are full of physical and mathematical abstractions, yet these are often difficult to integrate into the programming process. Furthermore, as modern languages and machines are introduced, there is a need to explore their effectiveness in scientific programming, their performance and usability, to enable scientists to select the best environment for developing their applications.

We have been investigating traditional and modern object oriented paradigms in modeling plasma particle-in-cell (PIC) simulations [11]. This work is in loose collaboration with the Numerical Tokamak Project (NTP) [12], which is an HPCC research effort to model and understand the transport of particles and energy in a tokamak fusion energy device. The PIC model [13] follows the trajectories of millions of particles in their self-consistent electromagnetic fields (both external and self-generated). Since the computation required is extremely large, both in terms of the field sizes and number of particles, the only alternative is to use massively parallel computers. While a variety of institutions are involved in many aspects of this project, our efforts concentrate on the problems of evaluating new languages, machines and programming techniques applicable to the next generation of advanced simulation programs.

The Research Direction Much of our research on the IBM SP2, and other machines, involves studying new programming paradigms based on object oriented technology. This work advances our understanding of emerging language standards and new paradigms for scientific computations. The most significant concerns in building up scientific applications include organization of the software (which impacts extension to new problems), clarity of design (which influences shared development among multiple contributors) and the development of appropriate computational abstractions.

Issues complicating the programming of plasma codes include adding more realism and physical effects, irregular data/communication characteristics requiring dynamic load balancing, modular design for extensibility and appropriate abstractions in programming. In the following, we comment on our scalable plasma object oriented codes designed from the Fortran 77 program skeletons, development efforts in C++ and Fortran 90 object oriented programming and implementation and performance of these codes on the IBM SP2, Intel Paragon and Cray T3D distributed memory supercomputers.

Object Oriented Plasma Modeling Computational modeling of plasma using the General Concurrent Particle-in-Cell model [14] involves operations on particles and fields that are partitioned across a distributed memory architecture. Finding appropriate abstractions to represent both the physical and computational/numerical aspects of this problem is challenging due to complex interactions among simulation components. In Figure 4 the class design for a three-dimensional parallel code is presented using the Object Modeling Technique (OMT) notation [15]. The Plasma class performs operations on the collection of particles to update their positions and velocities during a simulation time step, monitor energy diagnostics and to load balance particles across processors based on the domain decomposition of the particle partitioning. Aggregation diamonds lead to classes from which the plasma is composed, namely particle species and various fields. The Species template class represents a collection of particle objects with attributes that describe the species characteristics, such as electrons or ions. A Particle Distribution class initializes specific species distributions to account for initial conditions in various experiments.

Since a variety of fields exist (scalar charge density and vector electric fields), their common properties are inherited from the GeneralField3D class into the ScalarField3D and VectorField3D classes respectively. These fields are used in the solution of field equations (Maxwell's equations or their subset), from which the forces on particles are determined by interpolation. While the particles are described by continuous variables, the fields are represented by a discrete grid. Nevertheless, they are coupled and can describe each other in a physical sense, which explains the diamond ternary relation linking the collective plasma to the fields. The actual solution of Poisson's Equation to find the electric field from the charge density is solved in complex Fourier space, so a ComplexFields class is used to model this computational abstraction. Both fields and particles can be partitioned, hence a SlabPartition class handles the partitioning of these objects automatically. For portability, a VirtualParallelMachine class is used, however MPI is our primary message passing system. A number of auxiliary classes support object based message passing in MPI with multidimensional dynamic storage management useful in dynamically modifying field partitionings.

Software Design in C++ and Fortran 90 We have designed and implemented many scalar and parallel multidimensional codes based on object oriented models. As shown in Table 1, the benefits of encapsulation and abstraction modeling include well-defined interfaces to various routines in C++ and Fortran 90, compared to the Fortran 77 code. The Fortran 77

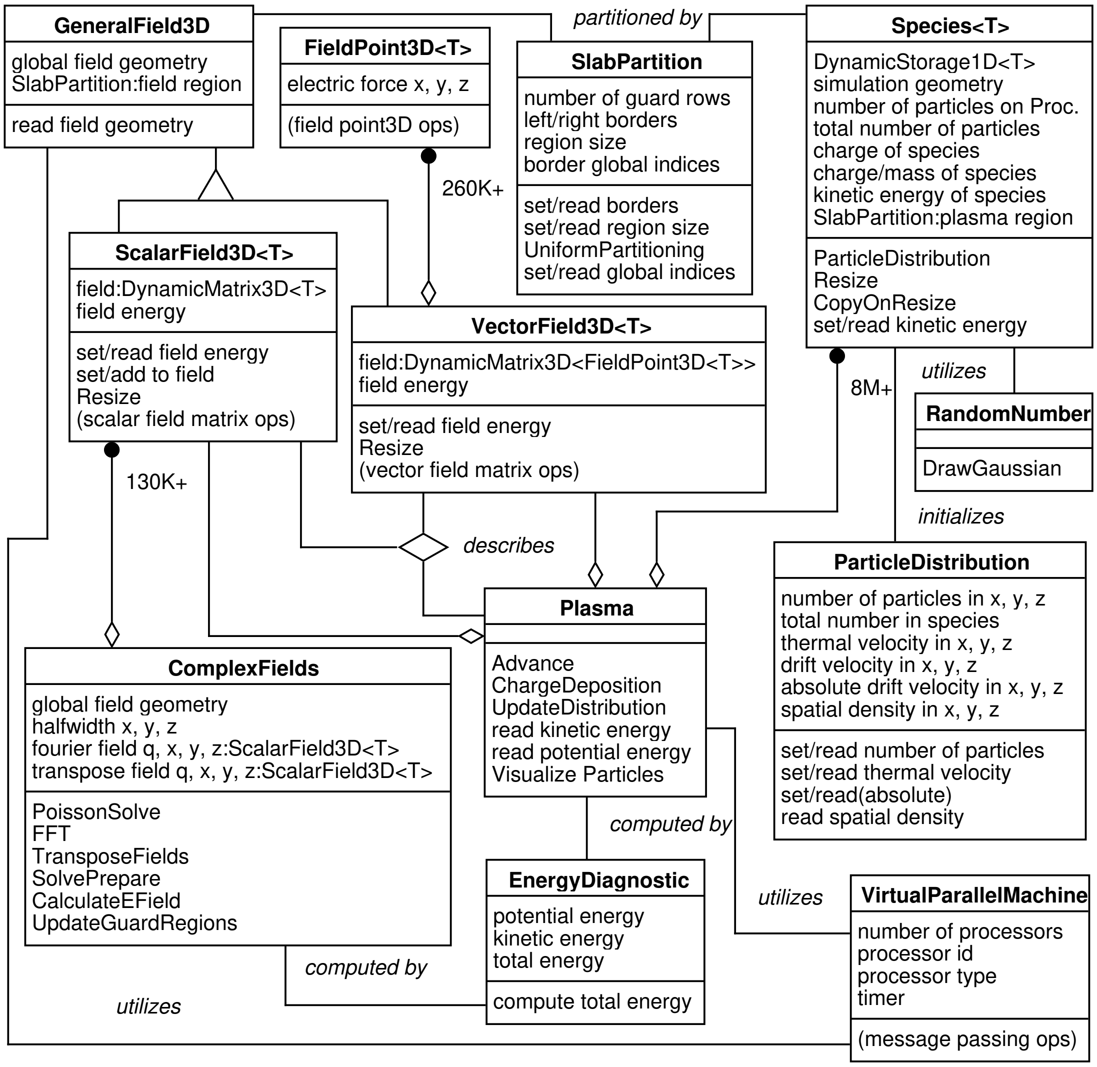

Figure 4: Class hierarchy for 3D parallel PIC code using OMT notation

call requires various arrays, dimensional parameters and constants to be passed as parameters, creating long argument lists. The Fortran 90 and C++ codes encapsulate the relevant features into object definitions, simplifying interfaces and collaborative code development. As a result, the C++ and Fortran 90 operations are dimension independent and need not be changed when the dimensionality of the problem changes.

Fortran 77	`call pdost2(part,q,npp,noff,qme,nx,idimp,npmax,nblok,nxv,nypmx)`
C++	`plasma.ChargeDeposition(electrons, cdensity);`
Fortran 90	`call plasma_dpost2(electrons,cdensity,edges)`

Table 1: Object/function calls in two-dimensional charge deposition operation.

Fortran 90 provides a number of important features beyond Fortran 77 that allow for object oriented programming [17]. Encapsulation, overloading, generics, inheritance, dynamic memory management, and a rich set of array operations, have opened new design alternatives for scientists looking to modernize their existing applications. Since Fortran 90 is backward compatible with Fortran 77, the new features can be incrementally introduced. Indeed, many of these powerful ideas are useful even if the object paradigm is not introduced.

Performance Comparisons on Supercomputers Our development environments include the IBM SP2, Intel Paragon and Cray T3D distributed memory supercomputers. The original Fortran 77 codes compile and execute across these machines, as expected given the wide support for this language. These codes served as the benchmark to which the C++ and Fortran 90 object oriented programs were compared.

We have had disappointing experiences on the Cray T3D with the Cray C++ and Cray Fortran 90 compilers. We use class templates in parameterizing features of the plasma model, such as the species and field components. Unfortunately, the Cray C++ compiler cannot instantiate the template classes correctly. The identical program does compile cleanly generating results under other compilers, on both parallel architectures and on workstations for the small scale programs. Additionally, the Cray Fortran 90 compiler understands the array syntax, but it does not yet support modules (as of June 1996). We use modules extensively in modeling classes in Fortran 90.

The Intel Paragon experiences have been much more positive, although the performance has been less favorable compared to the IBM SP2. The Paragon uses two or more i860 processors at clocked at 50MHz with a 175 MB/sec mesh connected network. The Intel C++ compiler generally performed well, template instantiation issues were not a problem.

Machine	Dim	Size	PEs	Fortran 77	Fortran 90	C++
IBM SP2	2D	350K	16	35.81	55.85	111.00
Intel Paragon	2D	350K	16	111.35	N/A	393.01
IBM SP2	2D	3.5M	32	193.22	372.28	1228.00
IBM SP2	3D	350K	16	87.05	—	192.00
Intel Paragon	3D	350K	16	393.42	N/A	1483.62
IBM SP2	3D	8M	32	1564.27	—	2797.00
Cray T3D	3D	8M	32	2582.50	—	—

Table 2: Sample performance characteristics for 2D/3D two-stream instability experiments.

Our most successful efforts have been on the IBM SP2 both in terms of compiler support and performance seen in Table 2. Our Paragon does not have a Fortran 90 compiler available. On the IBM SP2, we used the xlC and xlf90 compilers with MPI, where the thin-nodes have a 66.7 MHz clock with a 40 MB/sec network rate using us (user space switch) protocol. The performance of the object oriented Fortran 90 codes vary based on the object model used, so we are currently investigating ways to maximize performance.

The Impact of Languages and Architectures in the Future Language issues are relevant as developers in academia, government and industry make decisions regarding programming approaches for future scientific computation projects. There is a need for a comprehensive and serious study of the options available when considering languages for important projects, particularly when issues such as legacy codes and the need to design for extension are considered. In our experiences, the SP2 environment has been the most successful in supporting object oriented scientific programming. We plan to investigate the design effects of abstraction, how abstraction effects efficiency and the integration of multidisciplinary codes in scientific computing at Rensselaer.

3 Simulating Lyme Disease

Lyme Disease Lyme disease is heavily studied in the United States, particularly in the Northeast, where the disease is prevalent [18, 19]. It is transmitted to humans by infected deer ticks (*Ixodes scapularis*). The disease is sustained by the cycle of infection between ticks and white-footed mice (*Peromyscus leucopus*) (Figure 5) [20]. The tick life cycle spans two years. Larvae hatch in the middle of the summer, then start looking for a blood meal. When a larva finds a suitable host, such as a mouse, it bites it, takes a blood meal, and drops off. The blood meal provides the necessary energy for the larva to molt into a nymph, which survives the winter and starts looking for another blood meal in the spring. Again the mouse is a good host for the tick. When the tick drops off, it molts into an adult. Adult ticks feed again, and deposit eggs. The preferential hosts for the adult tick are deer.

In our simulations, we assume no transoverial transmission of the disease—the larvae hatch uninfected. If a tick is unable to find a suitable host, it dies. Since the disease cycle consists of mouse-tick interactions, we model the period during which the two species come in contact with each other, namely the 180-day period in the late spring and summer when the nymphal and larval ticks are active and seeking a blood meal. Mice, during that time, are looking for nesting sites and may carry ticks a considerable distance [21].

On the first day of the simulation, mice and nymphs are present. Infected nymphs can infect uninfected mice, and infected mice can infect uninfected nymphs. At about the 90th day, uninfected larvae hatch. If they feed on infected mice, they acquire the disease. At the end of the simulation, at day 180, we have adult and nymphal ticks, some of which are infected. The disease requires that both mice and ticks be present in the environment. If there were no mice, even in the presence of infected ticks, the disease would die out with the end of the tick life cycle.

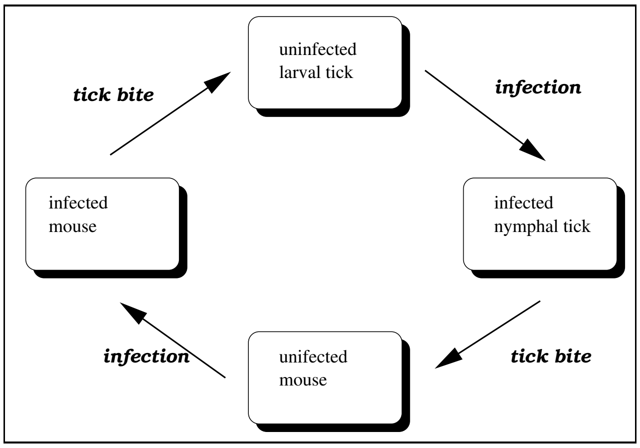

Figure 5: The cycle of Lyme Disease

.

Simulation Methodology The goal of the simulation is to understand the mouse-tick interaction at its most basic level. We want determine how many mice are needed to sustain the disease, or, conversely, under what conditions the disease will die off. The level of desired detail led us to discrete event simulation [22] and individual-based modeling [23]. Mice are treated as individuals and are the objects of the simulation. The following events are associated with mice: dispersal (locating a new nesting site), movement (from one location to another), being bitten by a tick (larval or nymphal), dropping a tick, and death. Ticks are too numerous to model as objects. They are present in space in the form of *blobs*, which are composites of all tick types (larval, nymphal, and adult), both infected and uninfected. We also want to be able to simulate large sets of mice and ticks; therefore, we have turned towards a parallel platform. Discrete event simulation is non-trivial to parallelize, since, in the sequential version, events are removed from a priority queue. The main challenge in parallelizing the problem is the necessity of preserving the causality between events.

In Parallel Discrete Event Simulation (PDES), the physical problem is decomposed into several physical processes. The behavior of each such process is simulated by a logical process (LP). Each LP possesses its own event queue, state information, and clock. Events are removed from the event queue and processed. The processing of an event causes a state change. LPs communicate with each other via event messages. To address the causality constraints between events, two major protocols have been developed [24]. The first protocol, known as conservative, preserves the causality between events by frequently synchronizing the simulation processes [25]. In distributed memory machines, such as the IBM SP2, the overhead of frequent synchronization can cause significant performance degradation. This degradation and the fact that, in our application, many events can happen concurrently within an LP, led us to the implementation and use of the second protocol, known as optimistic [26]. In this approach, causality errors are allowed to occur. When an LP receives an event with a time lower than its logical time, it has to roll back the computation to the time of event, process the event, and then continue the regular processing. Upon rollback, the state of the LP is restored to that just prior to the time of the occurrence of the event.

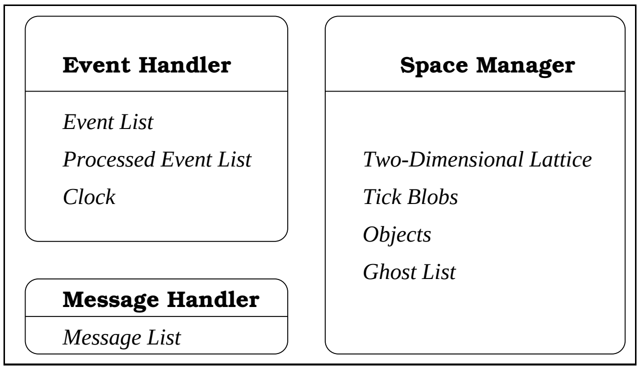

Figure 6: Components of a Logical Process

.

Computational Model We use a spatial decomposition to divide the problem into LPs. The territory in which the mice and ticks live is discretized and mapped onto a two-dimensional lattice. The lattice is then divided into LPs. The initial mapping can be one LP to one processor. We have implemented the system using C++ and MPI, and have used an object-oriented methodology to model the LPs. Our LPs are composed of several entities: the Space Manager, which oversees the movement of objects on the lattice, the Event Scheduler, which maintains the event queue and is responsible for the processing of events, and the Message Handler, which sends and receives messages between LPs. The LPs model the behavior of the objects that are present in the section of the lattice assigned to them. When an object moves out of the area assigned to an LP, an event containing that object and all its future events is sent to the appropriate Space Manager. In order to be able to quickly restore the state when a causality error occurs, the Space Manager keeps a list (Ghost List) of objects sent out and of their events. Figure 6 presents an overview of an LP. The Processed Event List and Message List are also needed to support rollback.

Results The aim of this project is not only to obtain biologically significant results [27], but also to achieve good performance and to contribute to the field of PDES. As mentioned earlier, rollback is a vital part of optimistic PDES. In order to support the LP's ability to go backward as well as forward in time, state information has to be saved. As the simulation progresses, the amount of information saved increases. It is possible that the amount of information saved will grow past the available memory. It is therefore necessary for the LPs to be able to determine which information is obsolete and can be discarded. In order to do this, the LPs have to find the global minimum time of all the unprocessed events in the system and of the events still in transit; this time is known as the Global Virtual Time (GVT). Since there are no events with a time lower than the GVT, the LPs will not roll back beyond the GVT. Thus, all events with timestamps lower than the GVT can be discarded. It is important to be able to calculate the GVT as efficiently as possible, especially since this calculation is performed several times during the simulation.

We have devised an algorithm (Continuously Monitored Global Virtual Time (CMGVT)) which continuously monitors the progress of the simulation in the system. The main difference between our algorithm and others is the fact that there is no explicit synchronization between the processes. In other algorithms, rounds of synchronization messages have to be sent [28], during which time no new simulation messages can enter the system. By contrast, our algorithm runs continuously on every process in the system. When messages are sent between processes, some information about the state of the sending LP is appended to the message. This information includes the local knowledge of the sender about other LPs in the system (such as the local time of the other LPs) as well as basic information about outstanding messages. Upon message receipt, this information is integrated with the local knowledge of the receiving LP. At any point in time each LP has its own view of the state of the entire system. From this information it is straightforward to calculate the GVT.

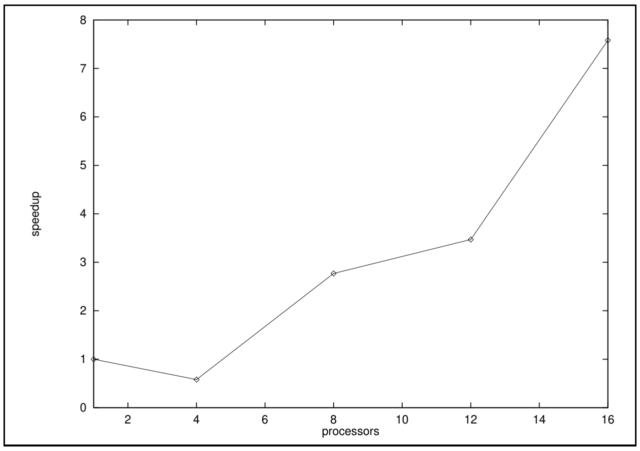

Figure 7: Speedup for CMGVT with Optimism Interval of 20 Days

.

In PDES it is often the case that an LP gets far ahead of the others. An LP might, for example, not have many events to process. This situation is not desirable, because the work done by the LP will most likely be rolled back. In order to prevent processes from running far ahead of others, restrictions on the amount of virtual time that an LP can be ahead of others are made. Imposing these restrictions, which we call, curbing the optimism, is easy when using CMGVT [29]. Since each LP knows about the local time of all the other LPs, it can decide not to process events beyond a certain amount of time ahead of the other LPs. Performance results with an optimism interval of 20 days are presented in Figure 7. The dip in the curve for four processors is due to the fact that each process is responsible for a large part of the lattice. Currently, when a rollback occurs, events in the entire area assigned to the process have to be undone. Thus the cost of a rollback proportional to the size of the lattice assigned to a process. We are actively developing algorithms to minimize that cost.

Acknowledgements

This work was partially supported by NSF Grants ASC-9318184, BIR-9320264, CCR-9527151, AFOSR Grant F49620-95-1-0407, ARO Grant DAAH04-95-1-0091 and by the NASA Graduate Student Researchers Program under Grant NGT-70334, carried out at Rensselaer and in part at the Jet Propulsion Laboratory, California Institute of Technology. The content does not necessarily reflect the position or policy of the U.S. Government.

References

[1] K. Clark, J. E. Flaherty, and M. S. Shephard. *Applied Numerical Mathematics*, special ed. on Adaptive Methods for Partial Differential Equations, 14:255–283, 1994.

[2] M. W. Beall and M. S. Shephard. Mesh Data Structures for Advanced Finite Element Applications. SCOREC Report 23-1995. Scientific Computation Research Center, Rensselaer Polytechnic Institute, Troy, 1995.

[3] M. S. Shephard , J. E. Flaherty, H. L. de Cougny, C. Özturan, C. L. Bottasso, and M. W. Beall. Parallel automated adaptive procedures for unstructured Meshes. *Parallel Computing in CFD*, R-807, AGARD, Neuilly-Sur-Seine, 6.1-6.49, 1995.

[4] C. L. Bottasso, H. L. de Cougny, M. Dindar, J. E. Flaherty, C. Özturan, Z. Rusak, and M. S. Shephard. Compressible aerodynamics using a parallel adaptive time-discontinuous Galerkin least-squares finite element method. Paper 94-1888, *Twelfeth AIAA App. Aerodyn. Conf.*, Colorado Springs, 1994.

[5] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI.* The MIT Press, 1994.

[6] C. L. Bottasso, J. E. Flaherty, C. Özturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. The quality of partitions produced by an iterative load balancer. In B. K. Szymanski and B. Sinharoy, Eds., *Proceedings Third Workshop on Languages, Compilers, and Runtime Systems*, Kluwer, Boston, pp. 265–277, 1995.

[7] A. Pothen, H. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. on Matrix Anal. and Applics.*, 11(3):430–452, 1990.

[8] B. Hendrickson and R. Leland. The Chaco user's guide, version 1.0. Technical Report SAND93-2339, Sandia National Laboratories, Albuquerque, New Mexico, 1993.

[9] K. D. Devine, J. E. Flaherty, R. M. Loy, and S. R. Wheat. Parallel partitioning strategies for the adaptive solution of conservation laws. In I. Babuška, J.E. Flaherty, W.D. Henshaw, J.E. Hopcroft, J.E. Oliger, and T. Tezduyar, Eds., *Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations*, Springer-Verlag, Berlin-Heidelberg, pp. 215–242, 1995.

[10] E. Leiss and H. Reddy. Distributed load balancing: design and performance analysis. *W. M. Kuck Research Computation Laboratory*, 5:205–270, 1989.

[11] C. D. Norton, B. K. Szymanski, and V. K. Decyk. Object Oriented Parallel Computation for Plasma Simulation. *Communications of the ACM*, 38(10):88–100, October 1995.

[12] B. I. Cohen, D. C. Barnes, J. M. Dawson, G. W. Hammett, W. W. Lee, G. D. Kerbel, J.-N. Leboeuf, P. C. Liewer, T. Tajima, and R. E. Waltz. The numerical tokamak project: simulation of turbulent transport. *Computer Physics Communications*, 87(1&2):1–15, May II 1995.

[13] C. K. Birdsall and A. B. Langdon. *Plasma Physics via Computer Simulation*. The Adam Hilger Series on Plasma Physics. Adam Hilger, New York, 1991.

[14] P. C. Liewer and V. K. Decyk. A General Concurrent Algorithm for Plasma Particle-in-Cell Simulation Codes. *J. of Computational Physics*, 85:302–322, 1989.

[15] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[16] V. K. Decyk, C. D. Norton, and B. K. Szymanski. Experiences with Object Oriented Parallel Plasma PIC Simulation. In *Proc. Computing in High Energy Physics (CHEP '95)*, Singapore, Sept. 18–25 1995. Lafex/Fermilab, World Scientific.

[17] V. K. Decyk, C. D. Norton, and B. K. Szymanski. Introduction to Object-Oriented Concepts Using Fortran 90. Technical Report 10-1996, Scientific Computation Research Center (SCOREC), Rensselaer Polytechnic Institute, Troy, NY 12180-3590, June 1996. Submitted to Computers in Physics.

[18] A. Barbour and D. Fish. The biological and social phenomenon of Lyme disease. *Science*, 260:1610–1616, 1993.

[19] G.L. Miller, R.B. Craven, R.E. Bailey, and T.F. Tsai. The epidemiology of Lyme disease in the United States 1987-1998. *Laboratory Medicine*, 21:285–289, 1990.

[20] F.S. Kantor. Disarming Lyme Disease. *Scientific American*, pages 34–39, September 1994.

[21] R.S. Ostfeld, O.M. Cepeda, K.R. Hazler, and M.C. Miller. Ecology of Lyme disease: habitat associations of ticks (Ixodes scapularis) in a rural landscape. *Ecol. Applications*, 5:353–361, 1995.

[22] C.G. Cassandras. *Discrete Event Systems: Modeling and Performance Analysis*. 1993.

[23] O.P. Judson. The rise of the individual-based model in ecology. *Trends in Ecology and Evolution*, 9:9–14, 1994.

[24] R. M. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):31–53, 1990.

[25] K. M. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, 5:440–452, 1979.

[26] D.R. Jefferson. Virtual Time. *Trans. Prog. Lang. and Syst.*, 7:404–425, 1985.

[27] E. Deelman, T. Caraco, and B. K. Szymanski. Parallel Discrete Event Simulation of Lyme Disease. *Pacific Biocomputing Conference*, 1996.

[28] J. S. Steinman, C. A. Lee, L. F. Wilson, and D. M. Nicol. Global Virtual Time and Distributed Synchronization. pages 139–148. 1995.

[29] E. Deelman, B. K. Szymanski and T. Caraco. Simulating Lyme Disease Using Parallel Discrete Event Simulation. *to appear in* Winter Simulation Conference, 1996.