Proc. Int. Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'95),
Las Vegas, NV, November 1995, pp. 13-22

Plasma Simulation on Networks of Workstations using the Bulk-Synchronous Parallel Model

Mohan V. Nibhanupudi Charles D. Norton Boleslaw K. Szymanski

Department of Computer Science
Rensselaer Polytechnic Institute
Troy, NY, USA 12180-3590
{nibhanum, nortonc, szymansk}@cs.rpi.edu

Abstract

Computationally intensive applications with frequent communication and synchronization require careful design for efficient execution on networks of workstations. We describe a Bulk-Synchronous Processing (BSP) model implementation of a plasma simulation and use of BSP analysis techniques for tuning the program for arbitrary architectures. In addition, we compare the performance of the BSP implementation with a version using MPI. Our results indicate that the BSP model, serving as a basis for an efficient implementation, compares favorably with MPI.

Keywords: BSP Model, Networks of Workstations, Parallel Computing, Plasma

1 Introduction

Complex scientific applications such as plasma simulation rely on parallel processing for efficiency. The General Concurrent Particle In Cell (GCPIC) method [2] partitions the particle and field data among the processors to distribute the computational load. However, this method leads to frequent communication and synchronization among the component processes because particle and field data must be exchanged between partitions. As a result, such applications do not execute efficiently on a network of workstations (NOW), which is a group of processors loosely coupled through a relatively slow communication medium. Nevertheless, NOWs are inexpensive, widely available, user friendly and often underutilized, hence it is desirable to be able to run scientific applications efficiently on them.

The objectives of the work presented in this paper are two-fold. First, we want to investigate techniques for implementing complex scientific applications involving frequent communication and synchronization efficiently on NOWs. We focus on minimizing the overall execution time (wall clock time), rather than the system or user time. Second, we want to investigate suitability of the Bulk-Synchronous Processing Model [9] for

implementing such applications. To this end, we compare the performances of a BSP library and the MPI message passing library on plasma simulation codes.

2 Overview of Plasma PIC Simulation

A material subjected to extreme heat, pressure or electric discharges undergoes ionization, where the electrons are stripped from the atoms, acquiring free motion. The created mixture of heavy positively charged ions and fast electrons forms an ionized gas called a plasma. Familiar examples of plasma include the Aurora Borealis, neon signs, the ionosphere, and solar winds. Fusion energy is also an important application area of plasma physics research. The plasma Particle In Cell simulation model [1] integrates in time the trajectories of millions of charged particles in their self-consistent electromagnetic fields. Particles can be located anywhere in the spatial domain; however, the field quantities are calculated on a fixed grid. Following [6], our simulation models only the electrostatic (coulomb) interactions.

The General Concurrent Particle in Cell (GCPIC) Algorithm [2] partitions the particles and grid points among the processors of the MIMD (multiple-instruction, multiple-data) distributed-memory machine. The particles are evenly distributed among processors in the primary decomposition, which makes advancing particle positions in space and computing their velocities efficient. A secondary decomposition partitions the simulation space evenly among processors, which makes solving the field equations on the grid efficient. As particles move among partitioned regions, they are passed to the processor responsible for the new region. For computational efficiency, field/grid data on the border of partitions are replicated on the neighboring processor to avoid frequent off-processor references.

As in [6], we perform a Beam-Plasma instability experiment in which a weak low density electron beam is injected into a stationary (yet mobile) background plasma of high density, driving plasma waves to instability. Experiments such as this can be used to verify plasma theories and to study the time evolution of macroscopic quantities such as potential and velocity distributions. Figure 1 shows an overview of the organization of the simulation program.

3 Bulk-Synchronous Processing Model and NOWs

The Bulk-Synchronous Processing model [9], introduced by Leslie Valiant, consists of *components* (processors) which perform processing or memory functions, a *router* that provides point to point communication between pairs of components, and a *synchronization mechanism* to synchronize all or a subset of the components at regular intervals. Computation consists of a sequence of *supersteps*. In each superstep, a component performs some local computation and/or communicates with other components. All components synchronize at the end of each superstep and the data communicated are not guaranteed to be available at the destination until the end of the superstep.

A BSP computer is characterized by the following set of parameters: number of processors (p), processor speed (s), synchronization periodicity (L) and a parameter to indicate the global computation to communication balance (g). L is the minimal

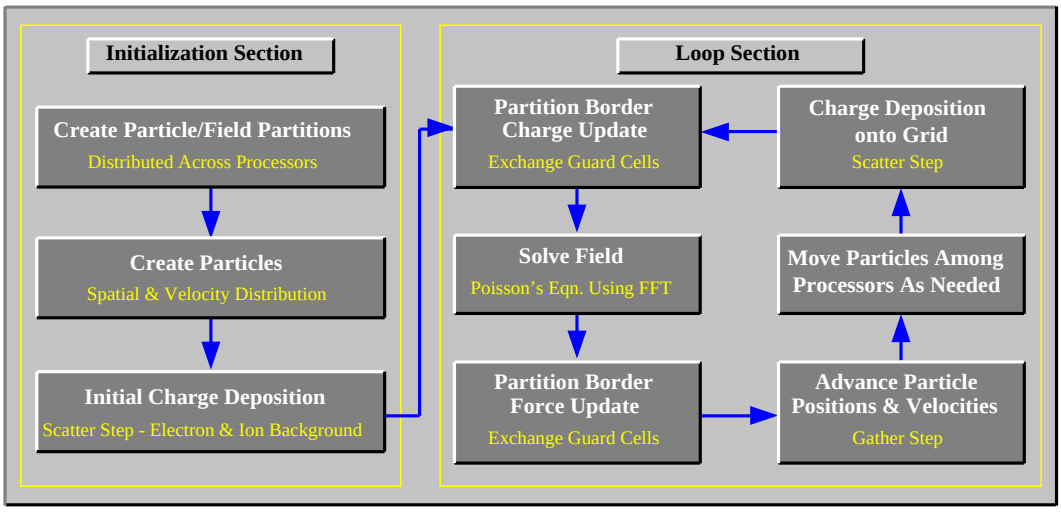

Figure 1: *Plasma PIC Computation Loop Overview. Reprinted with permission from* [6] *under* ©*Copyright 1995 by ACM Inc.*

number of time steps between successive synchronization operations. g is the ratio of the total number of local operations performed by all processors in one second to the total number of words delivered by the communication network in one second. It should be noted that the parameters L and g usually are also not constants, but functions of the number of processors p. These functions are in turn defined by the network architecture and the implementation of the communication and synchronization primitives.

BSP parameters allow the user to analyze an algorithm in a simple and convenient way. Using such analysis, an algorithm can be optimized for a particular hardware by changing the distribution of the data, thus allowing the programmer to concentrate on the characteristics of the application rather than on the architectural features of the target machine.

3.1 Network of workstations as a BSP Computer

In terms of the BSP parameters, parallel computers are often characterized by large values of s (fast processors) and low values of L and g (a communication network with low latency and large bandwidth). A general purpose network of workstations, on the other hand, is characterized by values of s that are somewhat lower and values of L and g that are much larger than the corresponding values for the parallel machines (high latency and low bandwidth due to the loosely coupled nature of these networks). For the BSP implementation on NOWs, parameters L and g can be expressed as the following functions of the number of workstations p: $L(p) = L_0 \log(p)$, and $g(p) = g_0 p$. The network of workstations used to implement the plasma simulation is composed of Sun machines with Sparc 10/30, Sparc 10/50, Sparc 20/50 and Sparc 4/630 architectures connected via Ethernet with a bandwidth of 10 Mbits per second. We measured the BSP parameters for a group of Sparc machines used for our simulations. L_0 is equal to the time taken by about 7000 floating point operations and g_0 is about 22 floating point operations per one floating-point word communicated (assuming large size messages).

The high values of L and g in a NOW environment require different algorithms than those which work efficiently on a parallel computer. For example, to broadcast data

from one processor to all others, processors of a parallel machine are often treated as the nodes of a tree. At each step, nodes at the currently active level of this tree pass on the data to the nodes at the next level. The communication is increasingly parallel as data move from the root of this tree to the leaves. We refer to this scheme as *logarithmic broadcast*. The communication in the opposite direction (from the leaves to the root) implements data gathering. Both operations take number of steps proportional to the logarithm of the number of processors involved. Using this scheme to broadcast large amount of data on NOWs will cause the interconnection network to sequentialize message flow from/to the active processors. Hence, the cost of logarithmic broadcast on NOWs is

$$L\log(n) + g(n-1)h = L_0\log^2(n) + g_0(n-1)h,$$

In the *linear broadcast*, the broadcasting node simply communicates the data to all other nodes in a single superstep. Hence, the cost of the linear broadcast is

$$L + g(n-1)h = L_0\log(n) + g_0(n-1)h$$

Comparing the two costs shows that, unlike in a parallel computer environment, linear broadcast is always faster in a NOW environment. When logrithmic gather is used, computations can be performed in parallel at the nodes of the tree. Conversely, linear gather restricts computations to be delayed until all of the data arrives at the processor. This feature may make logarithmic gather more attractive than linear gather under some circumstances.

3.2 The Oxford BSP Library

The Oxford BSP Library [4, 5], developed by Richard Miller, is used to implement the plasma simulation on NOWs. It is based on a slightly simplified version of the model presented in [9]. The programming model uses a SPMD program style, and static allocation of processors. The most significant feature of the library is its use of remote assignment semantics for non-local data access, which simplifies debugging. The library is small, simple to use, yet robust. Figure 2 lists the library functions for C programs.

The Oxford BSP Library

- Process Management
  void bspstart(int argc, char* argv[], int maxprocs, int* nprocs, int* mypid);
  void bspfinish();
- Synchronization Management
  void bspsstep(int sstep_id);
  void bspsstep_end(int sstep_id);
- Communication Management
  void bspfetch(int pid, void* src, void* dst, int nbytes);
  void bspstore(int pid, void* src, void* dst, int nbytes);

Figure 2: *The Oxford BSP Library C Functions.*

Table 1: *Execution Times of the Plasma Simulation (Distributed Grid version) on a NOW (All times shown are in seconds).*

Number of Processors	20K Particles		
	Real Time	User Time	System Time
4	373.8	128.1	41.6
8	775.7	91.1	130.9

4 Plasma Simulation on NOWs using BSP Model

Norton et. al. [6] describe an object oriented implementation of the plasma simulation for distributed memory machines based on the GCPIC method. Their implementation partitions both the particles and the field data among the processors. As particles advance in space, they need to be redistributed to appropriate partitions. Redistribution of particles is achieved by a series of synchronous steps in which particles are passed from one neighbor processor to the next until they reach their destinations. Their implementation runs efficiently on parallel machines like IBM SP2, and Intel Paragon, which provide fast communication between the processors.

Our first attempt to implement the simulation on NOWs using the BSP model was to replace the message passing communication primitives with the communication and synchronization primitives from the Oxford BSP Library. No attempt was made to change either the data distribution or the simulation routines to improve the communication and synchronization performance of the implementation. This version was tested on a network of Sun Sparc workstations. As can be seen from Table 1, this implementation does not scale well. Increasing the number of processors from 4 to 8 reduces the computation time per processor (user time) by approximately 30%. However, the wall clock time for 8 processors is approximately twice that of 4 processors. These results confirmed our intuition: the distributed grid implementation is not suitable for the range of BSP parameter values that characterize NOWs.

It is important to understand the reasons for the poor performance of distributed grid implementation of plasma simulation on NOWs, since they are relevant to many other applications as well. According to the measurements on IBM SP2 by Norton et. al. advancing the particles' positions and velocities accounts for about 90% of the overall simulation time and does not require inter-processor communication. Thus, the volume of the communication (total number of bytes delivered by the communication network) does not appear to be a problem even after taking into account the low bandwidth of the Ethernet. This relatively low volume of communication is spread over a large number of short messages, giving rise to a large number of communication calls and supersteps in the initial implementation on NOWs. This affects the performance in two ways. First, each communication call contributes an additional delay equivalent to the latency of the communication network. Second, every additional superstep contributes a synchronization point and adds synchronization delay to the overall execution time.

To better fit the BSP model, the distributed grid implementation can be improved by reorganizing the data distribution. Interprocessor communication can be reduced if all the interactions between the particle and the field data are local to each processor. Since the bulk of the work involves computations on particle data, the particles must

remain partitioned among the processors to support workload sharing. However, the grid can be replicated on each processor allowing the particle/field interactions to remain local. Replicating the grid requires maintaining consistency of the grid data across all the processors. This is is achieved by merging the modified copies of the grid from individual processors. All the processors send charge depositions to their local grid, to one processor. This processor combines the individual charge depositions and broadcasts the final grid to all the processors.

5 Plasma Simulation using a replicated grid

P. C. Liewer et. al. [3] implemented the plasma simulation on a 32 node JPL Mark III hypercube. The particle computations were done in parallel; each processor had a copy of the field data. Thus their implementation used a replicated grid, as a first step towards parallelizing the particle simulation codes. Our decision to use a replicated grid comes from an analysis of plasma simulation using the BSP model on NOWs. Replicating the field data on all processors has two positive effects:(i) it eliminates communication caused by particle-field interactions and (ii) it groups together communication needed to update field information on each processor. It also has one disadvantage. The field computation is not parallelized. Since it constitutes 10% of overall computation, replicating the grid limits the potential speedup to about 10.

There are other sources of execution inefficiency on a network of workstations. Parallel computer implementation [6] redistributes particles in a series of synchronous steps in which each processor exchanges particles with its neighbors. Such synchronization steps are very expensive on NOWs. In our current approach, the processors maintain a set of buffers, one for each processor. A particle which must be transferred to another processor is stored in a buffer corresponding to its destination. When this buffer is full or when all particles are processed, each processor sends its non-empty buffers directly to their destinations in a single superstep. Finally, replicating the grid allows for total elimination of particle redistribution. The evaluation of efficiency of such a solution versus the solution with buffered particle redistribution is given in the following section.

6 Analysis of the BSP implementation of plasma simulation

As the particles move in space, the region of the grid to which they deposit charge varies. Based on this criterion there are two approaches to organizing the particle data among the processors. In one approach (*scheme 1*) we redistribute particles after each advance operation, so that charges of the particles assign to a processor are always deposited to the same portion of the grid based on the particle partitioning (approximately about $1/p$ of the grid in size). Only these portions of the grid from each processor need to be combined to construct the complete globally consistent grid. This also improves the cache performance of the simulation since only this portion of the grid needs to remain in the cache on each processor during charge deposition. However, redistributing the particles may cause the load to become unbalanced. In the other approach (*scheme 2*), we allow the particles to remain on the same processor throughout the simulation. This

avoids the cost of redistributing the particles while maintaining the initial load balance. Since particles can now deposit charge anywhere, constructing the globally consistent grid requires combining the whole grid data from each processor. In addition, cache performance may also suffer during charge deposition stage.

The total cost of a BSP implementation is the sum of the computation, communication and synchronization costs. In the plasma simulation, computation consists of advancing the position and velocity of the particles, depositing charge from each particle onto the grid, and solving the field equations. Field equations are solved by a sequential FFT with a complexity of $O(N_g \log(N_g))$, where N_g is the number of grid points. The charge deposition and particle advance operations are performed for each particle in constant time. Hence, assuming ideal load balance, the complexity of this part of the computation is $O(\frac{N_p}{p})$, where N_p is the number of particles in the simulation and p is the number of processors. Another computational step adds together the charge depositions contributed by the individual processors. In the first scheme, charge depositions to the grid points in the guard columns must be added to the corresponding grid partition. For the 2D simulations discussed in this paper, the height of the guard columns is approximately $\sqrt{N_g}$. Assuming that there are c_w overlapping guard columns per processor, the cost of summation is $(n-1)c_w\sqrt{N_g}$. In our implementation $c_w = 3$. In the second scheme, entire grids are added during the parallel gather operation, so the associated cost is $N_g \log(p)$.

To analyze communication costs, let c_{pm} be the average fraction of particles that cross an arbitrary selected plane perpendicular to the x-axis of the space in one simulation step. c_{pm} is dependent on the characteristics of the simulation (the speed distribution of the particles). With p processors, the average fraction of particles that change partitions is $c_{pm}p$. Let c_g be the size of data associated with a grid point and c_p be the size of data associated with a particle. Let $k = \frac{N_p}{N_g}$ be the average number of particles per cell.

In the first scheme, we use a linear gather for combining the grid data and a linear broadcast to return the grid to the processors. Particles are sent directly to the destination processor in one superstep. The communication cost in this case is

$$g_0(n-1)c_g(\frac{N_g}{p} + c_w\sqrt{N_g} + N_g) + g_0 N_p c_p c_{pm} p.$$

For the second scheme, there is no particle redistribution and the only cost is that of gathering the grid data from each processor into the global grid and broadcasting the resulting grid to all processors. We use logarithmic gather to combine the grid data, because summation of the data from individual processors can be done on the nodes of the tree in parallel. We use linear broadcast for sending the final grid to the processors. Following the analysis of the logarithmic and linear broadcasts in the previous section, the communication cost is $2g_0(N_g c_g)(p-1)$.

By adding the computation and communication costs derived above with the synchronization cost, we obtain the cost function for the first scheme (which uses particle redistribution):

$$\begin{aligned} T_1 &= c_1\frac{N_p}{p} + c_2 N_g \log(N_g) + c_w\sqrt{N_g}(p-1) + g_0(\frac{N_g}{p} + c_w\sqrt{N_g} + N_g)c_g(p-1) \\ &\quad + g_0 N_p c_p c_{pm} p + 3L_0 \log(n) \end{aligned} \tag{1}$$

Table 2: *Execution Times of Plasma Simulation (Replicated Grid version, with no particle redistribution) on a NOW (All times shown are in seconds).*

No of Procs	20K Particles			300K Particles			3.5M Particles		
	Real Time	User Time	System Time	Real Time	User Time	System Time	Real Time	User Time	System Time
4	510	143	9	5959	1919	32	68075	24352	232
8	410	102	20	3731	1036	77	27318	10225	322
16	504	92	53	3155	686	182	25619	5835	711

where c_1, c_2 represent the operation counts for charge deposition/force computation and field computation, respectively. Similarly, the cost function for the second scheme is

$$T_2 = c_1\frac{N_p}{p} + c_2 N_g \log(N_g) + N_g \log(n) + 2g_0 N_g c_g(p-1) + L_0\log^2(n) + L_0\log(n) \quad (2)$$

Particle redistribution is more efficient than broadcasting the whole grid when (2) exceeds (1). That is when

$$\begin{aligned} N_g \log(p) + L_0\log^2(p) + g_0 N_g c_g(p-1) \;>\; & c_w\sqrt{N_g}(p-1) + g_0(\frac{N_g}{p} + c_w\sqrt{N_g})c_g(p-1) \\ & + g_0 N_p c_p c_{pm} p + 2L_0\log(p) \end{aligned} \quad (3)$$

For large simulations using a large number of processors, the above inequality can be approximated by $N_g c_g > N_p c_p c_{pm}$. That is, redistributing particles is advantageous when the size of the grid data that needs to be shared by the processors is larger than the size of the particle data that needs to be exchanged among the processors. In our simulations, $c_g = 1, c_p = 4$ and c_{pm} ranges from 0.002 for 300,000 particles to 0.0009 for 3.5 million particles. With this data, the above inequality can also be written as $k < 130$ for 300,000 particles and $k < 280$ for 3.5 million particles. The value of k usually grows faster with the growth of the problem size than the value of c_{pm} decreases. Hence, the solution without particle redistribution should become more efficient than the alternative for very large simulations. Figure 3 shows a plot of the cost function for this solution. The values of the coefficients $c_1 = 458$ and $c_2 = 33$ have been estimated from theoretical analysis of the algorithms and experimental data. The plot indicates that the execution time continues to decrease up to 20 processors, even though the improvement may not be significant after 16 processors. In a synchronous application, computation proceeds at the speed of the slowest machine. As the number of processors increases, the variation in the load (due to other users) also increases. The mean value of the random variable that represents the execution rate of the slowest machine decreases with increasing number of processors. This implies that the performance of the algorithm in a heterogenous network will decrease statistically based on the overall system load. This explains why the observed execution times from Table 2 are larger than the theoretical cost function values of Figure 3. However, the trends in execution times shown in Table 2 correspond well with the cost function plot.

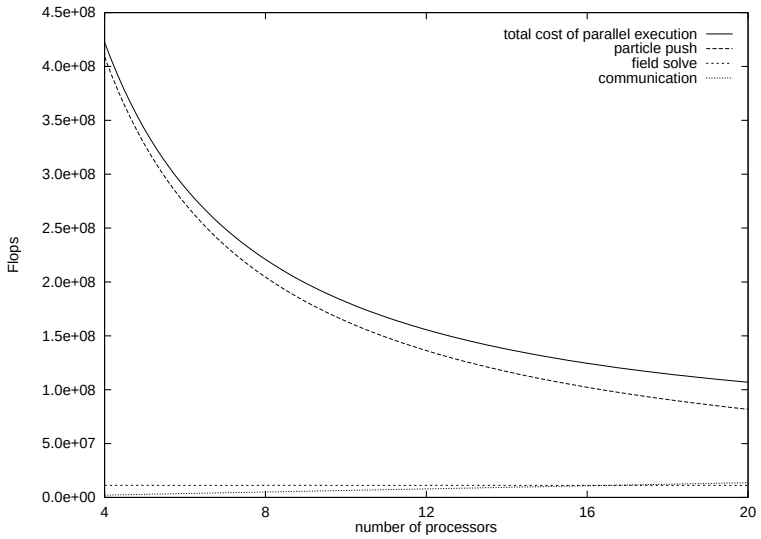

Figure 3: *Cost function for plasma simulation using no particle redistribution. The simulations uses 3571712 particles and a grid of size 32768 ($k = 109$).*

Table 3: *Comparison of BSP implementation versus MPI implementation. The timings are for a 300K particle simulation. These simulations are run on a different set of machines than the simulations shown in Table 2 (All times shown are in seconds).*

Number of Processors	MPI			BSP		
	Real Time	User Time	System Time	Real Time	User Time	System Time
4	3592.9	1919.4	296.7	4207.5	1838.5	10.7
8	2695.1	1040.1	43.9	2646.6	1006.3	20.5

7 BSP and MPI Performance Results on NOWs

Execution times of plasma simulation using the replicated grid are shown in Table 2. Plasma simulation using the replicated grid achieved impressive efficiencies for a small number of processors for 300,000 and 3.5 million particle simulations. The execution time (real time) initially decreases as the number of processors is increased, until it reaches a minimum value. Beyond this point, the execution time increases as the number of processors is increased for a constant problem size. For the simulation runs with 20,000 particles the minimum execution time occurs for 8 processors, however for 300,000 and 3.5 million particles, the minimum occurs for 16 processors.

We have implemented the replicated grid version of plasma simulation using Version 1.0.8 of the MPI message passing library, jointly developed by Argonne National Laboratory and Mississippi State University [8]. While the BSP implementation uses a Bulk Synchronous approach, the MPI version uses asynchronous communication and introduces no other synchronization than inherently present in the application. Both versions use a replicated grid with particle redistribution. Table 3 presents a comparison between the two implementations. Unfortunately, the MPI library for workstations sometimes failed during the simulation. (Because of these difficulties, the timings shown for 8 processors in Table 3 were extrapolated from a partial run). These partial results

indicate that the BSP library performs comparable to the MPI library for larger simulations.

8 Conclusion

Our work shows that the Bulk Synchronous Parallel model can be suitable for implementing scientific applications. We have used the model to implement the plasma simulation on a network of workstations. Using the BSP model allows us to concentrate on the characteristics of the application rather than the hardware features of the target machine. It also facilitates the analysis of reorganization of data distribution to improve performance. We demonstrated that it is possible to implement the plasma simulation on a network of workstations efficiently using the BSP model, and that this implementation compares favorably with an MPI implementation. We are currently investigating the performance characteristics of plasma simulations of larger size (simulations involving several million particles, which will be comparable to simulation runs on supercomputers).

References

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

[2] 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.

[3] P. C. Liewer, V. K. Decyk, J. D. Dawson, and G. C. Fox. Plasma Particle Simulations on the Mark III Hypercube. *Mathematical and Computer Modelling*, 11:53–54, 1988.

[4] Richard Miller. A Library for Bulk-synchronous Parallel Programming. *Proceedings of the British Computer Society Parallel Processing Specialist Group workshop on General Purpose Parallel Computing*, December 1993.

[5] Richard Miller and Joy Reed. The Oxford BSP Library Users' Guide Version 1.0. *Technical report, Programming Research Group, University of Oxford*, 1993.

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

[7] C. D. Norton, B. K. Szymanski, and V. K. Decyk. Parallel Object Oriented Implementation of a 2D Bounded Electrostatic Plasma PIC Simulation. In *Proc. Seventh SIAM Conference on Parallel Processing for Scientific Computing*, pages 207–212, San Francisco, California, February 15–17, 1995.

[8] W. Gropp E. Karrels E. Lusk P. Bridges, N. Doss and A. Skjellum. User's Guide to mpich, a Portable Implementation of MPI. 1995.