3D Electromagnetic Plasma Particle Simulations on the Intel Delta Parallel Computer

J. Wang, P. C. Liewer, and P. Lyster

Jet Propulsion Laboratory, California Institute of Technology

V. Decyk

University of California, Los Angeles

Abstract

A three-dimensional elect.ror[~ag[wtic PIC code has been developed on the 512 node Intel Touchstone Delta MIMD parallel computer. This code is based on the General Concurrent PIC algorithm which uses a domain decomposition to divide the computation among the processors. The 3D simulation domain can be partitioned into 1-, 2-, or 3-dimensional subdomains. Particles must be exchanged between processors as they move among the subdomains. The Intel Delta allows one to use this code for very-large-scale simulations (i.e. over 10s particles and 10^6 grid cells). The parallel efficiency of this code is measured, and the overall code performance on the Delta is compared with that on Cray supercomputers. It is shown that our code runs with a high parallel efficiency of $\geq 95\%$ for large size problems. The particle push time achieved is 115 nsecs/particle/time step for 162 million particles on 512 nodes. Comparing with the performance on a Cray C90, this represents a factor of 58 speedup. The code uses a finite-diflcrcncc leap frog method for field solve which is significantly more efficient than fast Fourier transforms on parallel computers.

1. Introduction

Computer particle simulation has become a standard method in space and laboratory plasma physics research. A particle-in-cell (PIC) code simulates plasma phenomena by modeling a plasma as hundreds of thousands of test particles and following the evolution of the orbits of individual test particles in the self-consistent electromagnetic field. Each time step in a PIC code consists of two major stages: the particle push and the field solve. Since the particles can be located anywhere within the simulation domain but the macroscopic field quantities are defined only on discrete grid points, the particle push uses two interpolation (gather/scatter) steps to link the particle orbits and the field components.

While the particle simulation method allows one to study the plasma phenomena from the very fundamental level, the scope of the physics that can be resolved in a simulation study critically depends on the computational power. The computational time/cost and computer memory size restricts the time scale, spatial scale, and number of particles that can be used in a simulation. The cost of running three dimensional electromagnetic PIC calculations on existing sequential supercomputers limits the problems which can be addressed.

Recent advances in massively parallel supercomputers have provided computational possibilities that were previously not conceivable. The objectives of this study are to develop a three-dimensional electromagnetic PIC code for MIMD parallel computers and to test the full potential of using parallel computers for very-large-scale particle simulations. In section 2, our 3D PIC code is discussed. This code is implemented on the 512 node Intel Touchstone Delta parallel computer at Caltech using the General Concurrent PIC (GCPIC) algorithm[1]. Section 3 discusses the code performance. The parallel efficiencies of running the code for fixed problems and scaled problems will be discussed, and the overall performance of the code on the Intel Delta will be compared with that on Cray supercomputers. Section 4 contains a summary and conclusions.

Copyright © 1994 by the authors. Published by the American Institute of Aeronautics and Astronautics, INC. with permission.

2. A Parallel 3D Electromagnetic PIC Code

The Algorithm

The basic procedures of a generic electromagnetic PIC code are as follows:
(1) Define the initial conditions of the particles and fields;
(2) Distribute the charge and current of the particles to the nearby grid points to obtain the charge density ρ and current density $\vec{J}$ at each grid point;
(3) Solve the Maxwell equations

$$\nabla \cdot \vec{E} = \rho \quad (1)$$

$$\nabla \cdot \vec{B} = 0 \quad (2)$$

$$\frac{\partial \vec{E}}{\partial t} = c \nabla \times \vec{B} - \vec{J} \quad (3)$$

$$\frac{\partial \vec{B}}{\partial t} = -c \nabla \times \vec{E} \quad (4)$$

to obtain the electromagnetic field at each grid point;
(4) Interpolate the electromagnetic field on the particle position to obtain the force on each particle; and
(5) Update the particle velocity and position from the Newton's second law

$$\frac{d\gamma m\vec{V}}{dt} = q(\vec{E} + \vec{V} \times \frac{\vec{B}}{c}) \quad (5)$$

In our code, the relativistic equation of motion is used for particle push. The trajectory of each particle is integrated using the usual time-centering leapfrog scheme:

$$\vec{v}^{n+1/2} - \vec{v}^{n-1/2} = \frac{dt}{\gamma^n}[\frac{q}{m}\vec{E}^n + \frac{\vec{v}^{n+1/2} + \vec{v}^{n-1/2}}{2} \times \frac{q}{m}\frac{\vec{B}^n}{c}] \quad (6)$$

$$\vec{x}^{n+1} - \vec{x}^{n-1} = \vec{v}^{n+1/2} dt \quad (7)$$

where the superscripts $n + 1/2$ and $n + 1$ represents the time step, and the γ is the relativistic gamma.

The field equations are most commonly solved by transform methods such as fast Fourier transform (FFT). However, transform methods are '(global" methods because the field information from every point in the simulation domain contributes to each single field harmonic. in general, global methods are not very efficient for parallel computers because they involve a large amount of interprocessor communications which may eventually become the bottleneck. For a code to run efficiently in parallel, a method that updates the field purely from the local data is preferred.

From the Maxwell's equations, one notes that eq(1) will always be satisfied as long as the charge conservation condition

$$\frac{\partial \rho}{\partial t} = - \nabla \cdot J$$

is satisfied, hence, the electromagnetic field can be updated from only the two curl Maxwell's equations (3) and (4) if one can enforce rigorous charge conservation numerically. A rigorous charge conservation method has been developed in the Magic and Quicksilver codes by Sandia National Laboratories[2] and the Tristan code by Buneman et al[3,4]. In this scheme, the electromagnetic field is updated locally by finite-difference leapfrogging:

$$\vec{E}^{n+1} - \vec{E}^n = dtc \nabla \times \vec{B}^{n+1/2} - dt\vec{J}^{n+1/2} \quad (8)$$

$$\vec{B}^{n+1/2} - \vec{B}^{n-1/2} = - dt[c \nabla \times \vec{E}^n] \quad (9)$$

This scheme also requires the use of a complex staggered grid mesh system in which $\vec{E}$ is defined at midpoints of cell-edges while $\vec{B}$ is defined at midpoints of cell-surfaces. This ensures that the change of B flux through a cell surface equals the negative circulation of E around that surface and the change of E flux through a cell surface (offset grid) equals the circulation of B around that surface minus the current through it. This frnite-difference leapfrogging scheme is used for our field solve.

Implementation on a MIMD Parallel Computer

There are basically two types of parallel computers: Multiple-Instruction Multiple-Data (MIMD) and Single-instruction Multiple-Data (SIMD). In a MIMD parallel computer, each processor may execute a separate stream of instructions while in a SIMD parallel computer, each processor executes the same instructions simultaneously.

Our 3D electromagnetic PIC code is implemented on a MIMD parallel computer, the Intel Touchstone Delta computer at Cal tech. The Intel Touchstone Delta system consists of an ensemble of nodes which are independent processors with its own memory connected as a two-dimensional mesh. There are 512 numerical nodes. Each numerical node is i860 chip based. The node operates at 40 MHz and has a peak speed of 80 single-precision Mflops and 60 double-precision Mflops. The available memory on Delta is about 12 Mbytes per node or an equivalent of 6.1 Gbytes on all 512 nodes.

The code is implemented using the general concurrent PIC (GCPIC) algorithm developed by Liewer and Decyk[1]. The GCPIC algorithm is designed to make the most computationally intensive portion of a PIC code, the particle computation, run efficiently on a MIMD parallel computer. The algorithm uses a domain decomposition to divide the computation among parallel processors. Each processor is assigned a subdomain and all the particles in it. When a particle moves from one

subdomain to another, it must be passed to the appropriate processors. For the code to run efficiently in parallel, the domain decomposition needs to be such that the subdomains have roughly the same number of particles for load balance. in our code, the computation domain can be partitioned into I-, 2-, or 3-dimensional subdomains ('(slabs", "rods", or '(cubes").

The code is written using Express-Fortran and compiled into a single object code. lath processor runs the object with a separate program counter. Each processor also has its own particle arrays and field arrays. The computations in each processor arc linked together through message-passing and global communications. Three major message-passing operations arc involved in the code: *particle trade, guard cell exchange*, and *guard cell* summation. Guard cells are the neighboring grids outside a processor's subdomain boundary, also stored by the processor, which are needed to insure that the interpolations (gather/scatter) can be performed locally (no interprocessor communication).

Particle trade passes the particles between processors. If a particle went out of bounds of a subdomain boundary, it is placed in a buffer. When all particles have been checked, the buffer is passed to the neighboring processors, and at the same time, incoming particle buffers arc received from the neighboring processors. *Guard cell exchange* and *guard cell* summation arc for communication of field information. When updating the field, the $\vec{E}$ and $\vec{B}$ field in guard cells need to be exchanged between neighboring processors so all processors have the updated conditions. When depositing the current, those particles near a subdomain boundary will contribute to the current on the grid points on both sides of this boundary. Hence, the guard cell currents need to be passed to the neighboring processors and added to the currents at the appropriate interior points of the neighboring processors.

Fig. 1 shows the flow chart of our parallel 3D electromagnetic PIC code. Note that the rounded blocks represent the steps in a sequential EM PIC code and the four rectangular blocks are the ncw steps needed in the parallel message-passing code.

3. Performance Analysis

The performance of our parallel 3D electromagnetic PIC code has been measured in three ways: 1) fixed problem size analysis; 2) scaled problem size analysis; and 3) comparison of the performance with that on Cray supercomputers.

An important measure of the performance on a concurrent computer is the parallel efficiency ϵ which measures the effects of communication overhead and load imbalance[5]. If there were no communications involved and the processor loads were perfectly balanced, the parallel efficiency would be $\epsilon = 100\%$. In this paper wc shall focus only on the effect duc to communication overhead. The simulation runs used in this section all have near-perfect load balance because the particle distributions arc nearly uniform. (Dynamic load balance for non-uniform particle distributions has been investigated in a 2D PIC code by Ferraro et al[6].)

Fixed Problem Size Analysis

In a fixed problem size analysis, we compare the times to run the same problem on an increasing number of processors. Since the total problem size is fixed, the problem size on each individual processor decreases as the number of processors increases.

Let us define $T(N)$ to be the time elapsed on a parallel computer with N nodes. For a problem that can be fit into a minimum of N_{min} processors, the parallel efficiency $N > N_{min}$ processors is defined by

$$C(N) = \frac{T(N_{min})N_{min}}{T(N)N} \quad (10)$$

For the fixed problem size analysis, we have considered two problems. The size of the first problem (F1) is 2.22 x 10^5 particles and $32^3 = 32768$ grid cells (~ 7 particle/cell). This problem can be fit on a single processor on Delta. The second problem (F2) has 1.4 x 10^7 particles and $64^3 = 2.62 \times 10^5$ cells (~ 54 particle/cell). F2 requires a minimum of 64 processors to run. F1 and F2 were run using processors from $N_p = N_{min}$ to $N_p = 512$. 3D domain partitions are used. The parallel efficiencies for F1 and F2 as a function of N_p are shown in Fig. 2a and the run times for different portions of the code are shown in Fig. 2b.

The results show that the efficiency for F1 drops significantly as the processor number is increased ($\epsilon(512) \simeq 24\%$). This is not surprising because the size of F1 is too small to run on the parallel computers. For instance, when we divide the computation in a 3D partit ion using 8 x 8 x 8 = 512 processors, each processor will only have a computation domain of 4^3 grid points and about only 430 particles. With such a small size problem on each node, the computation time becomes smaller than the internode communication time. The low efficiency simply reflects the fact that for F1 the code is dominated by internode communications. On the other hand, we find that F2 performs much better then F1 on multiple nodes because of its much larger problem size. (When F2 is divided into 512 processors, each node has a computation domain of 8^3 grid points

and about 2.77 x 10^4 particles.) The parallel efficiency for F2 stays at $\geq 95\%$. This demonstrates that a parallel computer is best suited only for large size problems.

Scaled Problem Size Analysis

We now study in detail the parallel efficiency for scaled problem size. In a scaled problem size analysis, we keep the problem size on each individual processor fixed while increasing the total number of processors. The total problem size is then proportional to the number of processors used, The parallel efficiency in a scaled problem size analysis is defined as

$$\epsilon(N) = \frac{T(1)}{T(N)N} \qquad (11)$$

We consider two cases for scaled problem analysis. In the first case (S1), each node has 32^3 cells and about 2.22 x 10^5 particles ($\sim$ 7 particles/cell). When S1 is loaded to all 512 nodes, the size of the total problem becomes 256^3 (1 6.8 million) cells and 114 million particles In the second case (S2), each node has 16^3 cells and 3.15 x 10^5 particles ($\sim$ 77 particles/cell). 'The size of S2 on all 512 node is then 128^3 (2.1 million) cells and 162 million particles. We note that the memory size required to run S1 and S2 on each node are 10.4 Mbytes and 11.6 Mbytes respectively. Considering the memory limit of 12 Mbytes per node, S2 represents about the largest problem that one can fit onto the Touchstone Delta system. When S2 is loaded to all 512 nodes of the Delta, the total memory size is an equivalent of 5.9 Gbytes.

The parallel efficiencies for S1 and S2 as a function of the processor number are shown in Fig. 3. The results show that a high parallel efficiency of $\epsilon \geq 95\%$ has been achieved.

The run times used by different portions of the code for S1 and S2 are shown in Fig. 4 as a function of the processor number. We find that the times the code spends on particle move, field update, and current deposit within each node ($T_{move}, T_{current}, T_{fldupdate}$) stay as a constant. This is because these three code portions do not involve internode communications, and our domain decomposition has assigned an equal amount of calculation to each processor. The times spent by the code portions that involve internode communications (T_{trade}, T_{gdfl}, and T_{gdsm}) increases somewhat as the node number increases. However, due to the large problem size on each node, the run time is dominated by "productive" calculations. For the problems considered here, the most computation intensive portion is particle push within each node. Hence, the increase of communication only has a minimum effect on the overall code performance. As Fig, 4 shows, the total run time is approximate a constant as the processor number is increased.

In our calculations, the guard cell number and the size of communicated message is independent of the processor number. However, the timing results in Fig. 4 shows that the guard cell communication time increases as the processor number increases. This is apparently a result of the Delta Mesh contention since the number of messages and message size exchanged by each processor is constant. We also note that, in both cases, the field solve time represents only a very small fraction of the total time ($T_{field}/T_{tot} \leq$ 2.4% for S1 and $T_{field}/T_{tot} \leq$ 0.5% for S2). As a test, in some other simulations we have used $\sim$ 5 particles/ceil. Even at such a low particle number/grid ccl] ratio, wc find the field solve still takes $\leq$ 4% of the total time. This demonstrates that the finite difference field solve is extremely efficient for parallel computers.

One of the most important measure of a PIC code's performance is the particle push time per particle per time step. The particle push time includes the times spent on moving particles, depositing currents, and related interprocessor communications (i.e. particle trade and guard cell summation): $T_{push} = T_{move} + T_{trade} + T_{current} + T_{gdsm}$. For S1 and S2, the particle push times on the 512 node Delta arc as follows:

$T_{push} \simeq$ 119 nsecs/particle/time step
for S1(114 million particles, 256^3 grid cells)
$T_{push} \simeq$ 115 rw42cs/particle/tirrle step
for S2(162 million particles, 128^3 grid cells).

Performance on Delta vs. Performance on Cray

Finally, we compare the overall performance of the code on Delta with that on Cray supercomputers. Two Cray computers were used for this analysis. The first one is the Cray Y-MP at JPL. The memory limit on the JPL Cray Y-MP is 16 Mwords or 128 Mbytes. The second one is one of the larges Cray supercomputer available, the Cray C90 at NASA Ames (The Von Neuman). The memory limit on the NASA Ames Cray C90 is 128 Mwords or 1.024 Gbytes. The memory limit on Intel Delta is about 6 times larger than that of the Cray C90.

Other than the message-passing and global communications, the Cray version of the code is identical to the parallel version, The Cray version of the code is compiled using the Cray Fortran compiling system's automatic vectorization and optimization. However, no rewriting was done to optimize the gather/scatter for the Cray. All the Cray runs were carried out on a single CPU.

In Fig. 5 we plot the total run times for the S1 and

S2 cases as a function of the "problem size". The unit of the problem size is defined as the problem size on 1 node of the Delta computer. For S1 (Fig. 5a), the size unit is 2.22 x 10^5 particles and 32^3 grid cells. For S2 (Fig. 5b), the size unit is 3.16 x 10^5 particles and 16^3 grid cells. Due to the memory limits on the Cray supercomputer, not all S1 and S2 problems can be run on the Cray. For instance, the largest S1 problem we ran on the Cray C90 was size $\simeq$ 134.6 (164^3 grid cells and 2.98 x 10^7 particles) and the larges S2 problem we ran was size $\simeq$ 91.13 (72^3 grid cells and 2.9x 10^7 particles).

To compare the performance, we shall define the Delta speedup as

$$S = \frac{(T_{tot}/size)_{Cray}}{(T_{tot}/size)_{Delta}} \qquad (12)$$

For small problems, the Cray supercomputer performs much better than the parallel computer. Comparing to Cray C90, the speedup factors at size = 1 are $S \simeq$ 0.10 for S1 and $S \simeq$ 0.12 for S2. However, as the problem size increases, the time spent on the Cray is approximately linear in the Log scale. While on the Delta, due to the high parallel efficiency, the total run times for S 1 and S2 stay almost an constant as both the problem size and processor number are increased. At size = 64, we find the speedup of the Delta over the Cray C90 has becomes S $\simeq$ 4.9 for S1 and S $\simeq$ 7.42 for S2. Extrapolating the run times on Cray to size = 512, if one had a Cray C90 large enough to run the size = 512 problems, then the speedup of Delta over Cray C90 would be about S $\simeq$ 49 for S1 and S $\simeq$ 58.7 for S2.

5. Summary and Conclusions

A MIMD parallel 3D electromagnetic PIC code has been developed on the 512 node Intel Touchstone Delta system. This code is based on the General Concurrent PIC (GCPIC) algorithm[1] which uses a domain decomposition to divide the computation among the processors. Three major message-passing operations, *particle trade*, *guard cell exchange*, and *guard cell summation*, are used to link the computations in different processors together, With 12 Mbytes memory limit per node and a total of about 6 Gbytes on all 512 nodes, the Intel Delta system allows our code to run simulations using over 10^8 particles and 10^6 grid cells. The parallel efficiency of this code is evaluated using both fixed problem analysis and scaled problem analysis. It is shown that our code runs with a high parallel efficiency of $\epsilon \geq 95\%$ for large size problems. The particle push time we have achieved is 115 nsecs/particle/time step for 162 million particles on 512 nodes. The overall performance of the code on the Delta is also compared with that on Cray supercomputers. Comparing with the runs on a Cray C90, our code has achieved a factor of 58 speedup on the Delta.

In the code, the electromagnetic field is updated locally using a rigorous charge-conservation finite-difference leap frog method. We find, for parallel computers, a finite difference field solve is significantly more efficient than fast Fourier transforms. Our results shows that the finite difference field solve generally takes $\leq$ 1% of the total CPU time for problems with about $\sim$ 77 particles/cell and $\lesssim$ 4% even for problems with $\sim$ 5 particles/cell.

Acknowledgments

The authors are grateful to E. Huang for her help in implementing the code on the Intel Delta parallel computer and for many useful suggestions through out this research. We would also like to acknowledge useful discussions with F. Tsung. The simulations on Cray C90 were carried out with the help of B. Mackey. This work was carried out by the Jet Propulsion Laboratory under contracts from Sandia National Laboratory and US Department of Energy through agreements with NASA. Access to the Intel Touchstone Delta System, which is operated by Caltech on behalf of the Concurrent Supercomputing Consortium, was provided by JPL. Access to the Cray supercomputers was provided by the JPL Supercomputing Project, which is sponsored by JPL and NASA Office of Space Science and Applications,

References

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

[2] M. Kiefer et al, Architecture and Computing Philosophy of the QUICKSILVER, Proceedings of Conference on Codes and the Linear Accelerator Community, (1990).

[3] J. Villasenor and O. Buneman, Rigorous Charge Conservation for Local Electromagnetic Field Solvers, *Computer* Physics *Communications*, *69*, 306-316 *(1992)*.

[4] *O.* Buneman et al, Solar Wind-Magnetosphere Interaction as Simulated by a 3-D EM Particle Code, *IEEE Trans. Plasma* Science, 20(6), 810816 (1992).

[5] G. Fox et al, Solving Problems on Concurrent Processors V.], Prentice-Hall, New Jersey (1988).

[6] R. D. Ferraro, P. C. Liewer and V. K, Decyk, Dynamic Load Balancing for a 2D Concurrent PIC Code, *J. Computational* Physics, (to be published, 1994).

Figure Captions

Figure 1: 3D electromagnetic GCPIC code flow

Figure 2: Code performance for fixed problems. a) Parallel efficiency. b) Times for various code portions.

Figure 3: Code performance for scaled problems: parallel efficiency.

Figure 4: Code performance for scaled problems: times for various code portions.

Figure 5: Run time on Delta vs. run time on Cray.

3D Electromagnetic GCPIC Code Flow

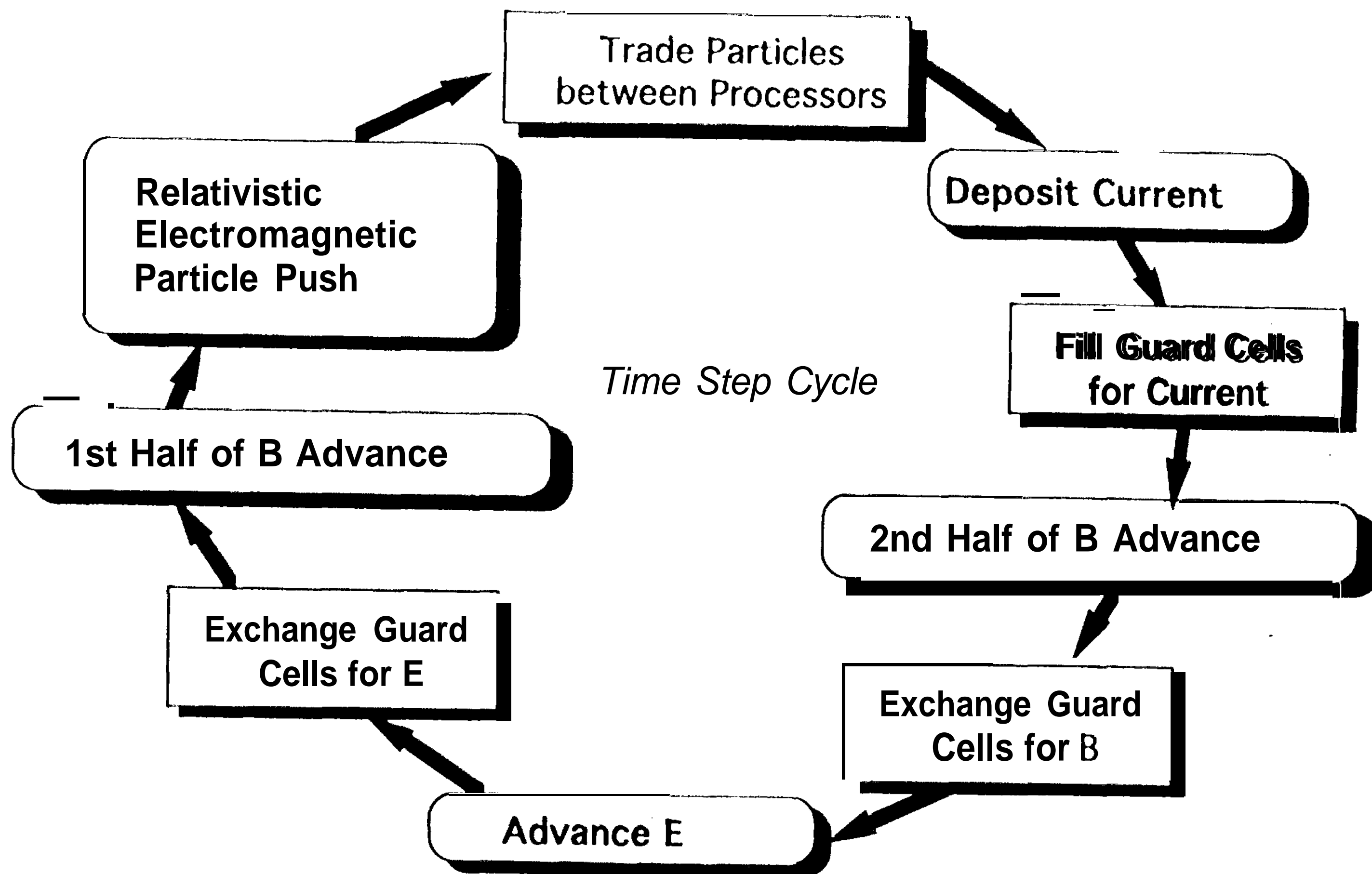
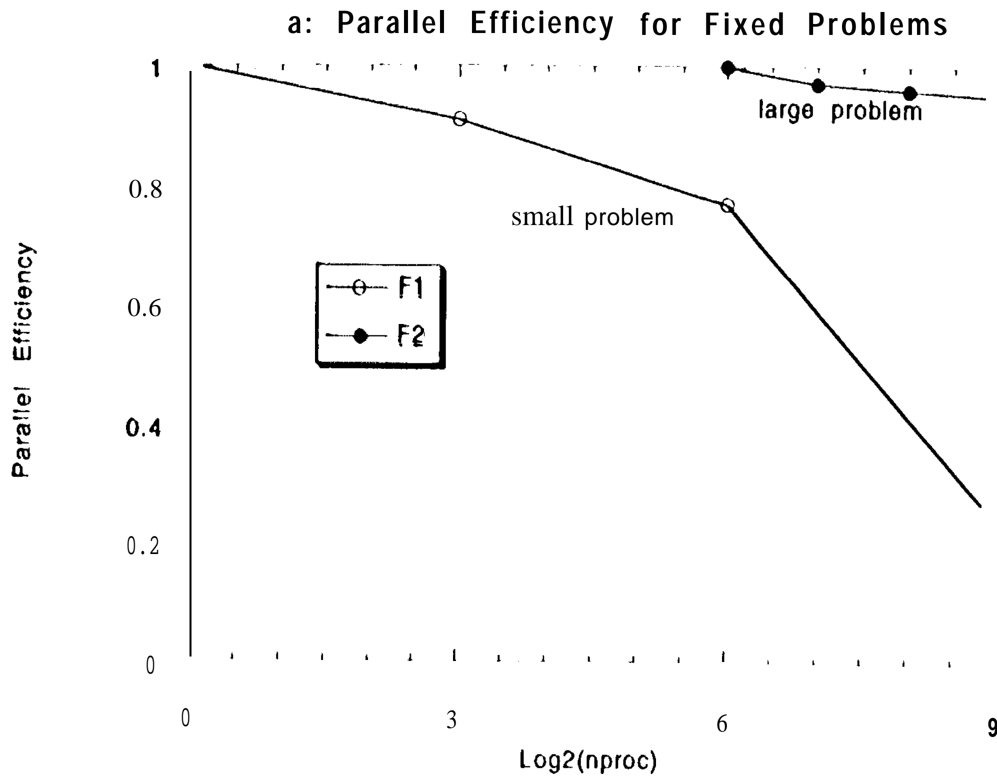


Figure 1



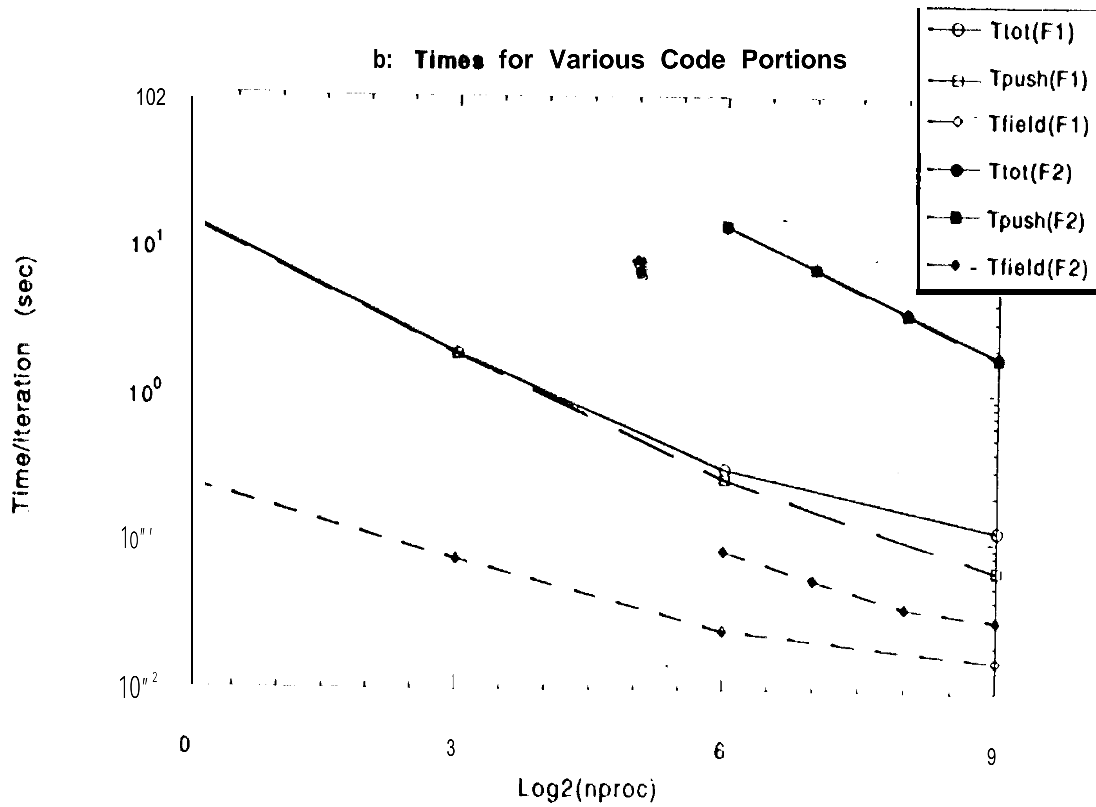


Figure 2

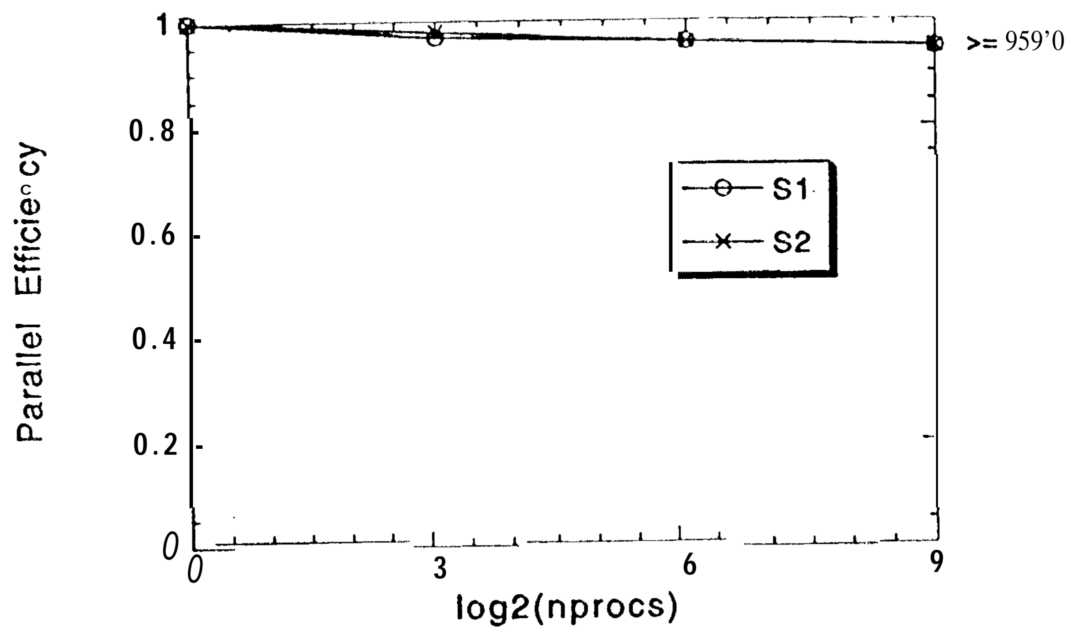


Figure 3

S1: 2.22E5 particles 32768 gird cells per processor

Time/Iteration (sec)

10^2

Total

Particle Push

10^1

Particle Move

Current Deposit

10^0

Particle Trade + BC

Field Solve

Update Field

Field Guard Cell + BC

10^{-1}

Current Guard Cell + BC

10^{-2}

0 3 6 9

Log2(nproc)

Legend: Ttot, Tpush, Tmove, Tcurrent, Tfield, Tfldupdate, Ttrade, Tgdfl, Tgdsm

S2: 3.16E5 particles 4096 grid cells per processor

Time/Iteration (sec)

10^2

Total

Particle Push

10^1

Particle Move

Current Deposit

10^0

Particle Trade + BC

10^{-1}

Field Solve

Field Guard Cell + BC.

Current Guard Cell + BC

Update Field

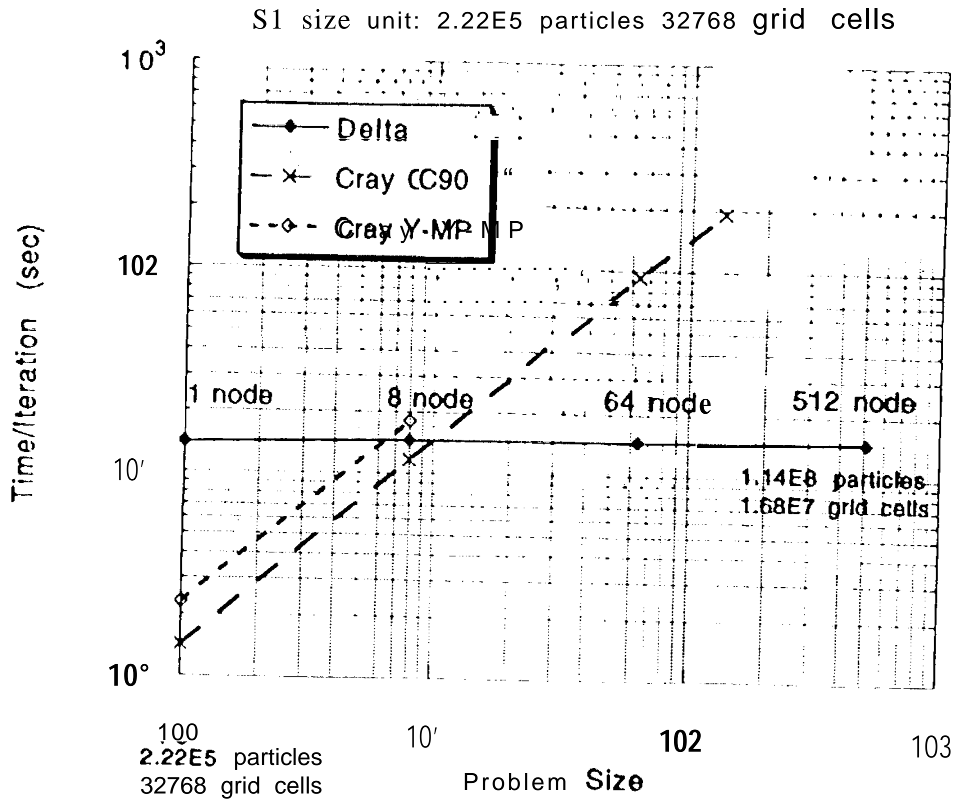
10^{-2}

0 3 6 9

Log2(nproc)

Legend: Ttot, Tpush, Tmove, Tcurrent, Tfield, Tfldupdate, Ttrade, Tgdfl, Tgdsm

Figure 4



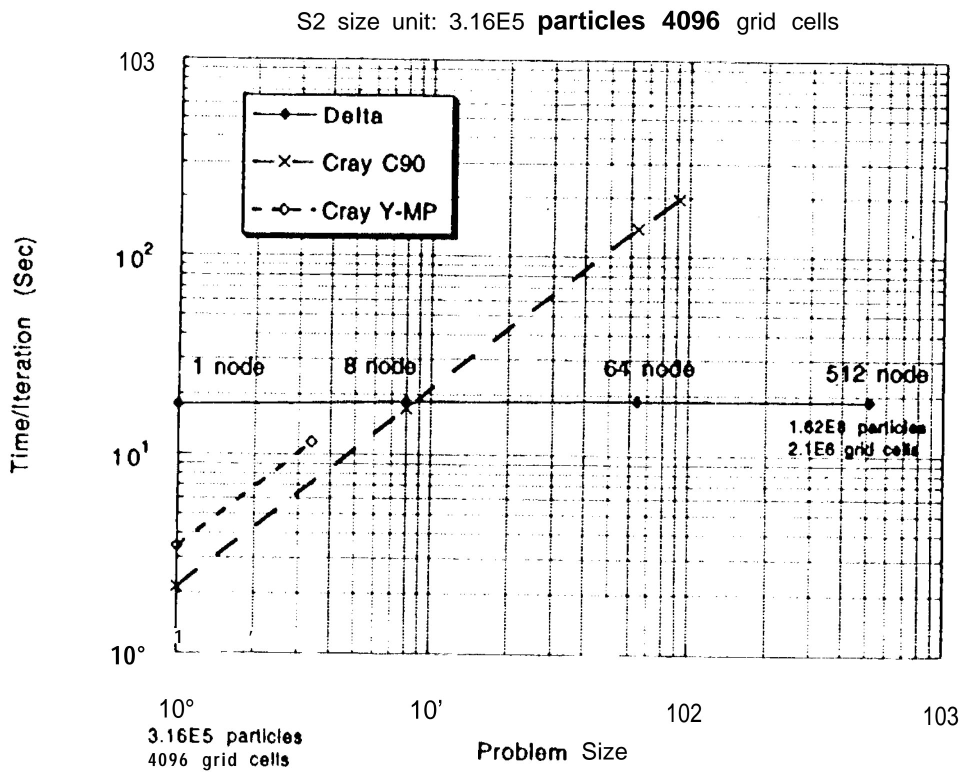


Figure 5