Skeleton PIC codes for Parallel Computers

Viktor K. Decyk
Physics Department, UCLA
Los Angeles, CA 90024
and
Jet Propulsion Laboratory/California Institute of Technology
Pasadena, California 91109

Abstract

Simple Skeleton Particle-in-Cell codes designed for massively parallel computers are described. These codes are used to develop new algorithms and evaluate new parallel computers. Benchmark results from a number of MIMD parallel computers are presented.

I. Introduction

One, two and three dimensional skeleton particle-in-cell codes have been developed for parallel computers to provide a testbed where new algorithms can be developed and tested and new computer architectures can be evaluated. These codes have been deliberately kept to a minimum, but they include all the essential pieces for which algorithms need to be developed. Thus the codes advance the particles, deposit their charge, and solve for the fields. Only one particle species is kept and the only diagnostic used is the particle and field energies. The codes use the electrostatic approximation and magnetic fields are neglected. Boundary conditions are periodic. The lessons learned from the skeleton codes are being incorporated into a more elaborate gyrokinetic particle code being developed as part of the Numerical Tokamak Project [1-2], a High Performance Computing and Communications (HPCC) project sponsored by the U. S. Dept. of Energy (DOE).

Particle-in-cell codes have been used in the plasma physics community for several decades and there are a number of excellent textbooks available on the subject [3-4]. The general idea is the particles interact with electromagnetic fields created by themselves as well as those externally imposed. Thus the code has two distinct parts and data structures. In one part, particles interact with fields and this interaction is described by Newton's law:

$$d\mathbf{v}_i(t)/dt = (q_i/m_i)^*\mathbf{E}(\mathbf{x}_i(t))$$
$$d\mathbf{x}_i(t)/dt = \mathbf{v}_i(t)$$

where the subscript i refers to the ith particle. The algorithm generally used to solve this set of equations is a time-centered leap frog scheme:

$$\mathbf{v}_i(t+dt/2) = \mathbf{v}_i(t - dt/2) + (q_i/m_i)^*\mathbf{E}(\mathbf{x}_i(t))$$
$$\mathbf{x}_i(t+dt) = \mathbf{x}_i(t) + \mathbf{v}_i(t+dt/2)$$

where the electric field at the particle's position $\mathbf{E}(\mathbf{x}_i)$, is found by interpolation from electric fields previously calculated on a grid. The interpolation is a gather operation which involves indirect addressing and a substantial part of the computation time is spent here. Generally, linear interpolation is used, but in the skeleton codes, quadratic spline functions are used for interpolation because they have an operation count similar to that needed in the gyrokinetic code.

In the second part of the code, the fields created by the particles must be found. In the skeleton codes, only Poisson's equation is used:

$$\nabla \bullet \mathbf{E} = 4\pi\rho(\mathbf{x})$$

This equation is solved on a grid, using Fourier transform methods. Typically, the time for the field solver is not large. The source term $\rho(\mathbf{x})$ is calculated from the particle position by an inverse interpolation:

$$\rho(\mathbf{x}) = \Sigma\, q_i S(\mathbf{x} - \mathbf{x}_i),$$

where S is a particle shape function. This is a scatter operation which also involves indirect addressing and consumes a substantial part of time in the calculation.

II. Description of Parallel Codes

Since the dominant part of the calculation in a particle code involves interpolation between particles and grids, it is important for a parallel computer that these two data structures reside on the same processor. The algorithm used to achieve this is called GCPIC and is described in P. C. Liewer and V. K. Decyk [5]. The essential idea is that different processors are assigned different regions of space and particles are assigned to processors according to the spatial region they are in. As particles move from one region to another, they are moved to the processor which is associated with the new region. During the interpolation particles must also access neighboring regions of space, so that extra guard cells are kept in each processor which are then combined or replicated as needed after the particles are processed. The passing of particles from one processor is performed by a particle manager subroutine. The passing of field data between guard cells in performed by a field manager subroutine.

The field solver uses Fourier transform methods. Therefore a parallel real-to-complex FFT was developed. For 2D and 3D data, a transpose method is used. The 2D case is described in R. Ferraro et. al. [6]. The field data is first distributed in the y coordinate and transformed in x (since each processor has all the x data for some y). Then the data is transposed so that the x coordinate is distributed across processors and transformed in y. The 3D case is similar, and is described in E. Huang et. al. [7]. In this case the data is first distributed across z, transformed in x and y, then transposed so that it is distributed across x and transformed in z. The maximum number of processors which can be used is limited by the maximum number of grid points in any one coordinate, but this is not a severe constraint at present since the Numerical Tokamak is designed for systems where this number is about 512. The transpose of data is performed by a transpose subroutine.

In the 1D code, the FFT is fully distributed, as described in V. K. Decyk, et. al. [8]. Two and three dimensional FFTs can also used this algorithm, but they are always slower than the transpose method in those cases when the transpose method can be used. To make the 2D and 3D FFTs more flexible, they have been designed so that if invoked with more processors than grids, they will only use only as many processors as grids. This allows the possibility to use more processors in pushing particles (which is the dominant part of the calculation) and fewer during the field solve.

During the transpose, each processor sends one piece of data to every other processor. There are a number of ways to accomplish this, but the safest way is to always send one message, receive one message, and so on. Flooding the computer with many messages then receiving them tends to overflow system resources and is not always reliable.

The skeleton codes always use uniform one dimensional partitions. Although algorithms for non-uniform[6] and multi-dimensional partitions[9] have been developed and will probably be necessary for the Numerical Tokamak, they have not been included in keeping to the spirit of simplicity of the skeleton codes.

The structure of the main loop of the skeleton code is illustrated in Figure 1 and is summarized as follows:

1. Particles coordinates are updated by an acceleration subroutine
2. Particles are moved to neighboring processors by a particle manager subroutine.

3. Particle charge is deposited on the grid, by the deposit subroutine.
4. Guard cells are combined by a field manager subroutine.
5. Real-to-complex FFT of charge density is performed by FFT subroutine.
6. Electric fields are found in Fourier space by a Poisson solver subroutine.
7. Complex-to-real FFT of electric field is performed by FFT subroutine.
8. Electric field is distributed to guard cells by a field manager subroutine.

This structure has the beneficial feature that the physics modules, items 1, 3, and 6 contain no communication calls (except for a single call to sum energies across processors). Thus they can easily be modified by physicists who do not have special knowledge of parallel computing or message passing. The modules which do communication, items 2, 3, and 8 do not perform any calculation, just management of data, and can be used by physicists as black boxes, where only the input and output must be understood. The FFT, items 5 and 7, are the usual sequential FFTs, with the addition of a single transpose subroutine embedded, which can also be used as a black box. One could actually incorporate the field manager into the FFT, but this was not done.

Since most message passing libraries are quite similar, it is a relatively simple matter to move from one distributed memory computer to another, by replacing the message passing calls in the communications modules with new ones. This primarily involves changing subroutine names and the order of the arguments. In a production environment this would be done by a preprocessor, if a message passing standard has not become widespread by them. The main exception to this simplicity is pvm, which is a more primitive environment than the message passing environments provided by all the other vendors. Specifically, pvm has no global operations, such as global sums, no simple synchronization mechanisms, and nodes are addressed by task identifiers (essentially network addresses). So a lot of missing pieces and translations must be supplied by the user himself.

III. Benchmark Results

The skeleton codes have been ported to a number of parallel MIMD computers and their performance tested. The process of porting the code to multiple platforms has been useful in uncovering bugs and weaknesses in the code. The simplicity of the skeleton codes has made this process less onerous.

All of the benchmarks were run in 64 bit precision and on the MIMD computers the subroutines have been optimized for scaler processors. The default safe level of optimization was used (usually -O). The total time reported excludes the initialization time. This is because the initialization was not parallelized in order that all the particles always have the same initial variables regardless of the number of processors and therefore the answer (energy values) calculated was always the same. This has proved useful in uncovering subtle bugs and compiler errors. Push time reported includes particle acceleration, passing particles between processors, and depositing the charge (items 1, 2, and 3 in the list above), for one particle for one time step. The field time includes the remaining items (4-8) in the list above. Normally, the field time is a small percent of the total time, but the transpose in the FFTs involves substantial communication, so that if communication on the parallel system is slow, this time would start to become significant.

In addition to the parallel benchmarks, a number of serial benchmarks were performed for comparison. The code used on these machines was a simpler non-parallel version. On the vector processors (Cray, NEC, and IBM ES/9000 computers), all the important subroutines were vectorized, which meant primarily that special versions of the charge deposition subroutine were used.

The physical problem being run is a beam-plasma instability, where 10% of the electrons are streaming with a velocity five times the thermal velocity of the remaining electrons. The problem is partitioned in a worst case scenario, that is, so that streaming particles are moving normal to the partition surfaces. This was done in order to stress the computers. The particles bunch up during the peak of the instability, so that some processors have more particles than others and the maximum load imbalance observed was about 10%.

The 2D benchmark problem uses 3,571,712 particles and a 128x256 mesh and runs for 325 time steps. The raw benchmark results are summarized in Table I and Figure 2. In Table I, Total Time = 3571712*325*Push Time + Field Time

The 3D benchmark problem uses 7,962,624 particles and a 64x32x128 mesh and runs for 425 time steps. The raw benchmark results are summarized in Table II and Figure 3. In Table II, Total Time = 7962624*425*Push Time + Field Time

IV. Comments on Specific Computers

Intel IPSC/860. This machine is a hypercube topology, and has been around for quite a while. It is very stable. The NX communication library was used. Although the network speed is slower than on the Paragon, the hypercube topology means each node has more connections and in some cases, such as the transpose, the IPSC/860 will perform better than the Paragon. This effect can be seen by comparing the field time for the IPSC/860 and the Paragon in Table I. One observes that as the number of processors increases, the field time for the IPSC/860 continues to improve, in contrast to the Paragon, until the IPSC/860 is actually faster.

Intel Paragon. This machine has a 2D mesh topology, with a faster processor and network speed than the IPSC/860. The operating system, OSF/1, is somewhat unstable, however, and best results are obtained if the user access is controlled. The NX communication library was used for the benchmarks, although pvm3 was tested and found to give the same results as NX. In running pvm, only the host/node model is recommended in order to avoid having compute processes running on the service partition.

Thinking Machine CM-5. This machine supports multiple programming models, but the best results were obtained using fortran 77 and the CMMD message passing library. This mode of operation means that only the sparc processor was used in the benchmark, and this processor is quite a bit slower than the RISC processors used by the other MIMD computers benchmarked. With the 1d code, the other programming models were also tried. A shared memory version written using global cmfortran [8], which did not use domain decomposition, gave results that were about 3 times slower than using fortran 77 with message passing. When cmfortran on a node is used together with message passing, the vector processors can be used, which should be more than 25 times faster than using the sparc processor alone. However, when this was tried it was found that the results were actually slower than using the sparc processor. Thus these alternate programming models on the CM-5 were abandoned in favor of fortran 77 with message passing. On the small system tested (32 processors), the machine was very stable. It was not possible for the user to vary the number of processors used by the code when a partition had been established at the time the machine was configured.

IBM SP1. This is a relatively new machine which is in essence a collection of RS/6000 workstations in a cabinet connected by ethernet as well as a high speed switch. Multiple communication libraries are available. The MPL message passing library was used in the benchmarks, running with the switch. MPL can also be used with ethernet. In addition, pvm3 and pvm2.4 are supported. There are also experimental versions of pvm available. In running pvm, both host/node and node only programming models are supported. The alternate libraries were tested for correctness, but not timed. Timings on this machine were obtained running stand alone and measuring wall clock time. This was necessary because each node can be running many Unix processes simultaneously by many users. Although only one user can use the switch at a time, there is no control over how many users can be using ethernet or running non-parallel jobs. Per processor, this machine gave the best results for the push time, although not for the overall time. This is due to the relatively slow performance in the field time, which is much more sensitive to communication speeds than the push time.

Cray T3D. This machine is very new. It uses a 3D mesh topology and is designed to support multiple programming models, but a distributed memory message passing model is the only one currently implemented. A message passing library is available based on pvm3 but with considerable extensions to make it easier to use. Except in the initialization these extensions were not used in order to maintain portability. Another nonportable feature was the use of 64 bit integer words. Per processor, this machine gave the best total time. Although the push time was not quite as fast as the IBM SP1, the field time performance was substantially better, indicating the T3D has much better communication speeds. There was insufficient usable memory on the JPL T3D to run the 3D benchmark.

V. Conclusions

Of the parallel computers the IBM SP1 and Cray T3D, the newest machines, give the best performance per processor for PIC simulation. Of course, which gives the best price performance depends on the price one can negotiate. The performance of the SP1 begins to degrade more rapidly than the T3D as the number of processors increases. The biggest disappointment is the poor performance of the cmfortran node compiler on the CM-5. When used with message passing, this compiler produces such poor code that the four vector processors actually run slower than a single sparc. There is potentially great room for improvement here.

Acknowledgments

I wish to acknowledge access to the Intel IPSC/860 at Sandia National Laboratory, New Mexico, and assistance from Mark Kiefer.

Access to the Intel Paragon at the Jet Propulsion Laboratory, Pasadena, California, was provided by NASA's Office of Aeronautics, and access to the Cray T3D at JPL was provided by NASA's Office of Mission to Planet Earth, Office of Aeronautics, and Office of Space Science. I wish to acknowledge assistance from Edith Huang, Paulett Liewer, and Robert Ferraro from JPL.

For large runs, the Intel Paragon located at the Center for Computational Sciences at Oak Ridge National Laboratory, Oak Ridge, Tennessee, was used.

Access to the CM-5 was provided by the Joint Institute for Computational Science, University of Tennessee, Knoxville, Tennessee, and I wish to acknowledge assistance by Jean-Noel Leboeuf of ORNL and Siddharthan Ramachandramurthi of U. Tenn.

This research was conducted in part using the IBM SP1 at the Cornell Theory Center, Ithaca, New York, which receives major funding from the National Science Foundation (NSF) and New York State. Additional funding comes from the Advanced Research Projects Agency (ARPA), the National Institutes of Health (NIH), IBM Corporation, and other members of the center's Corporate Research Institute. I wish to acknowledge assistance provided by Steve Lantz.

In additional, smaller runs were performed on the IBM SP1 at UCLA, and I wish to acknowledge assistance provided by Paul Hoffman, Gaber Mohamed and Ross Bollens.

Access to the vector Crays were provided by National Energy Supercomputer Center, in Livermore, California.

Access to the IBM ES/9000 was provided by the UCLA Office of Academic Computing, Los Angeles, California, and I wish to acknowledge assistance by Aeint de Boer.

Finally, access to the NEC SX-3 was provided by the National Institute for Fusion Science, Nagoya, Japan, and I wish to acknowledge help provided by T. Sato and K. Watanabi, and H. Takamaru.

The research described in this paper was carried out in part at UCLA and was sponsored by USDOE and NSF. It was also carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the National Aeronautics and Space Administration.

References

[1] J. M. Dawson, V. Decyk, R. Sydora, and P. Liewer, "High Performance Computing and Plasma Physics," Physics Today, vol. 46, no. 3, p. 64 (1993).

[2] V. K. Decyk and R. D. Sydora, "The Numerical Tokamak Project," Proceedings of the International Workshop on Plasma Physics, Pichl, Austria, February, 1993, in Current Topics in Astrophysical and Fusion Plasma Research, M. F. Heyn, W. Kernbichler, and H. K. Biernat, ed. [dbv-Verlag Graz, Austria, 1993], p. 32.

[3] C. K. Birdsall and A. B. Langdon, Plasma Physics via Computer Simulation, [McGraw-Hill, New York, 1985].

[4] R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles, [McGraw-Hill, New York, 1981].

[5] P. C. Liewer and V. K. Decyk, "A General Concurrent Algorithm for Plasma Particle-in-Cel Codes," J. Computational Phys. 85, 302 (1989).

[6] R. D. Ferraro, P. C. Liewer, and V. K. Decyk, "Dynamic Load Balancing for a 2D Concurrent Plasma PIC Code," J. Computational Phys. 109, 329 (1993).

[7] E. Huang, P. C. Liewer, V. K. Decyk, and R. D. Ferraro, "Concurrent Three-Dimensional Fast Fourier Transform Algorithms for Coarse-Grained Distributed Memory Parallel Computers," Caltech CRPC Report 217-50, Dec., 1993 (unpublished).

[8] V. K. Decyk, F. S. Tsung, P. C. Liewer, P. M. Lyster, and R. D. Ferraro, "Particle Simulation on Distributed Memory Parallel Computers," Proc. of the IAEA Technical Committee Meetings on Advances in Simulation and Modelling of Thermonuclear Plasmas, Montreal, Canada, June, 1992 [IAEA, Vienna, 1993], p. 237.

[9] P. M. Lyster, P. C. Liewer, V. K. Decyk, and R. D. Ferraro, "Implementation and Characterization of a Three-Dimensional Particle-in-Cell Code on MIMD Massively Parallel Supercomputers," submitted for publication.

Figure Captions

Figure 1. Structure of the main loop of the skeleton codes.

Figure 2. Total Time versus Number of Processors on Log-Log scale for various machines for the 2D benchmark.

Figure 3. Total Time versus Number of Processors on Log-Log scale for various machines for the 3D benchmark.

Table I

2D Particle Benchmarks

The following are times for a 2D particle simulation, using 3,571,712 particles and a 128x256 mesh for 325 time steps. Parallel codes use Domain Decomposition.

Push Time is the time to update one particle's position and deposit its charge, for one time step. Total Time is the total time for running the simulation minus the initialization time. Field Time is time for solving for the fields, including FFTs.

Computer	**Push Time**	**Total Time**	**Field**
Time			
Cray C-90, 1 proc:	944 nsec.	1105.6 sec.	9.4 sec.
Cray Y-MP, 1 proc:	1937 nsec.	2264.9 sec.	16.4 sec.
Cray 2, 1 proc:	2650 nsec.	3107.8 sec.	31.2 sec.
IBM ES/9000, M900, 1 proc:	3428 nsec.	4017.4 sec.	38.1 sec.
IBM RS/6000, M580, 1 proc:	6197 nsec.	7255.4 sec.	61.8 sec.
Sun Sparc1000, 1 proc:	21641 nsec.	25299.7 sec.	178.5 sec.
Intel IPSC/860, 64 proc:	362 nsec.	447.1 sec.	26.8 sec.
Intel IPSC/860, 32 proc:	702 nsec.	851.9 sec.	37.0 sec.
Intel IPSC/860, 16 proc:	1383 nsec.	1650.0 sec.	44.3 sec.
Intel IPSC/860, 8 proc:	2753 nsec.	3261.7 sec.	66.4 sec.
Intel Paragon XP/S, 64 proc:	274 nsec.	348.2 sec.	29.9 sec.
Intel Paragon XP/S, 32 proc:	541 nsec.	659.2 sec.	31.6 sec.
Intel Paragon XP/S, 16 proc:	1099 nsec.	1304.4 sec.	29.2 sec.
Intel Paragon XP/S, 8 proc:	2235 nsec.	2640.9 sec.	46.0 sec.
IBM SP1, w/MPL, 32 proc:	188 nsec.	270.2 sec.	51.9 sec.
IBM SP1, w/MPL, 16 proc:	349 nsec.	422.8 sec.	17.4 sec.
Cray T3D, w/PVM, 128 proc:	47 nsec.	73.5 sec.	19.3 sec.
Cray T3D, w/PVM, 64 proc:	98 nsec.	126.8 sec.	13.4 sec.
Cray T3D, w/PVM, 32 proc:	200 nsec.	241.9 sec.	10.1 sec.
Cray T3D, w/PVM, 16 proc:	411 nsec.	487.1 sec.	10.1 sec.
TMC CM-5, f77 w/CMMD, 32 proc:	1242 nsec.	1514.4 sec.	73.2 sec.

Table I

3D Particle Benchmarks

The following are times for a 3D particle simulation, using 7,962,624 particles and a 64x32x128 mesh for 425 time steps.
Parallel codes use Domain Decomposition.
Push Time is the time to update one particle's position and deposit its charge, for one time step. Total Time is the total time for running the simulation minus the initialization time.
Field Time is time for solving for the fields, including FFTs.

Computer	Push Time	Total Time	Field Time
NEC SX-3, 1 proc:	1966 nsec.	6784.4 sec.	131.5 sec.
Cray C-90, 1 proc:	2388 nsec.	8264.4 sec.	184.1 sec.
IBM ES/9000, M900, 1 proc:	20149 nsec.	68821.0 sec.	633.7 sec.
Intel Paragon XP/S, 64 proc:	1072 nsec.	3842.7 sec.	216.2 sec.
Intel Paragon XP/S, 32 proc:	1971 nsec.	6884.5 sec.	211.7 sec.
IBM SP1, w/MPL, 32 proc:	690 nsec.	2615.9 sec.	294.6 sec.
IBM SP1, w/MPL, 16 proc:	1363 nsec.	4892.2 sec.	280.4 sec.
TMC CM-5, f77 w/CMMD, 32 proc:	5867 nsec.	20053.1 sec.	198.6 sec.

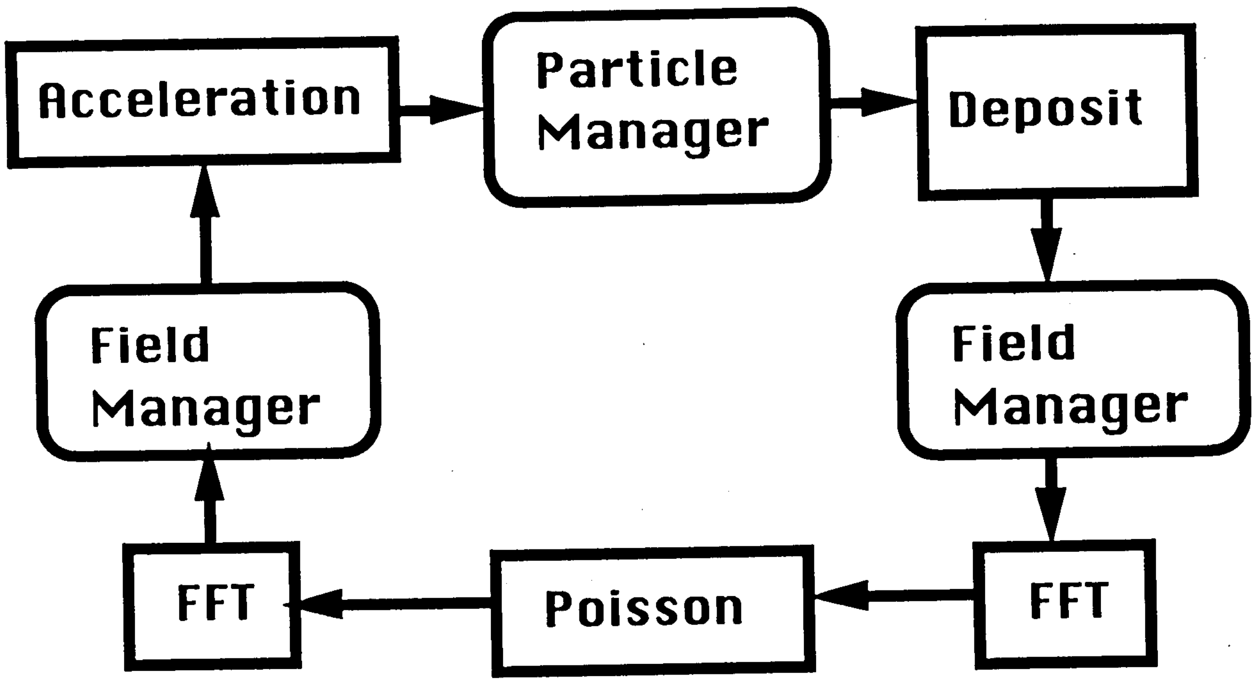
Acceleration
Particle Manager
Deposit
Field Manager
Field Manager
FFT
Poisson
FFT

Figure 1

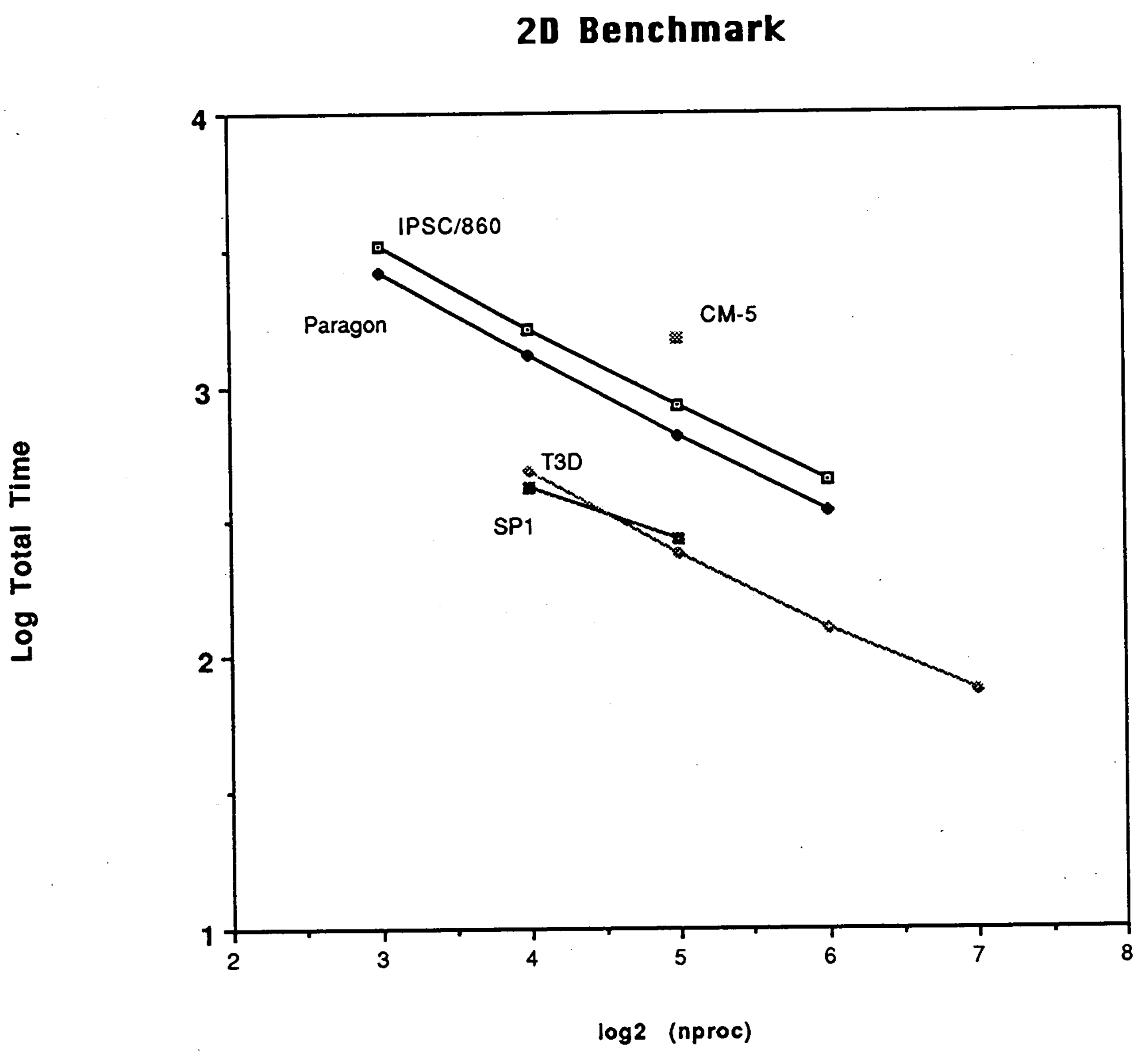
2D Benchmark
Log Total Time
log2 (nproc)
IPSC/860
Paragon
CM-5
T3D
SP1
1
2
3
4
5
6
7
8

Figure 2

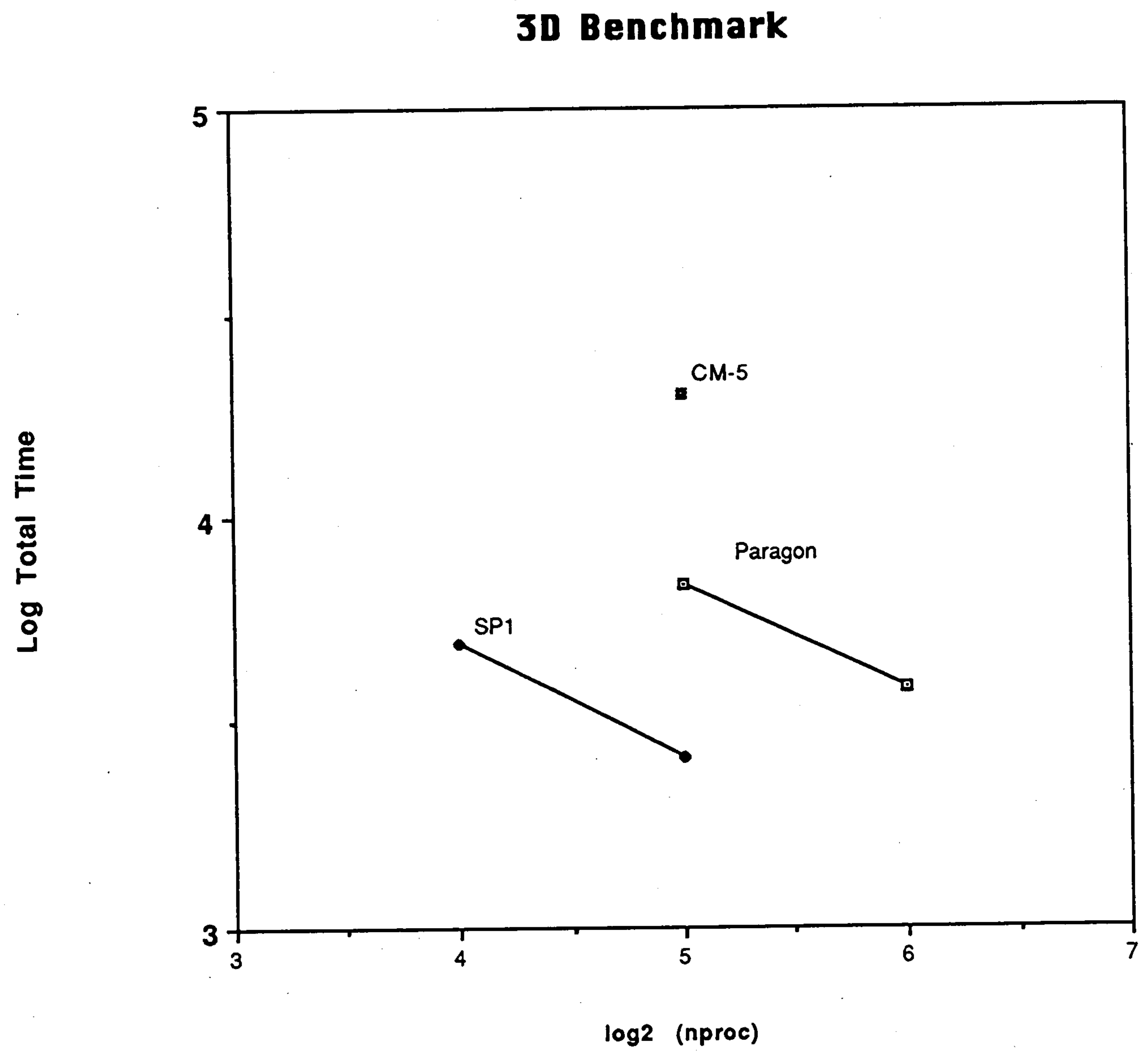
3D Benchmark
Log Total Time
log2 (nproc)
CM-5
Paragon
SP1
3
4
5
6
7

Figure 3