N87-26535

PHASE SPACE SIMULATION OF COLLISIONLESS STELLAR SYSTEMS ON THE MASSIVELY PARALLEL PROCESSOR

Richard L. White
Space Telescope Science Institute
Baltimore, MD

ABSTRACT

A numerical technique for solving the collisionless Boltzmann equation describing the time evolution of a self-gravitating fluid in phase space has been implemented on the Massively Parallel Processor (MPP). The code performs calculations for a two dimensional phase space grid (with one space and one velocity dimension). Some results from calculations are presented.

The execution speed of the code is comparable to the speed of a single processor of a Cray-XMP. Advantages and disadvantages of the MPP architecture for this type of problem are discussed. The nearest-neighbor connectivity of the MPP array does not pose a significant obstacle. Future MPP-like machines should have much more local memory and easier access to staging memory and disks in order to be effective for this type of problem.

Keywords: Stellar Dynamics, Phase Space, Galaxies, Star Clusters, Parallel Processing, N-body Techniques

INTRODUCTION

The dynamics of gravitating systems are of great interest because of their application to many astronomical systems such as clusters of stars, galaxies, and clusters of galaxies. These problems have two important characteristics which make them ideal for study using a parallel computer. First, the fundamental physics governing the evolution of the system is both simple and well-known: the force between any pair of bodies in the system is just determined by Newton's law of gravitation. Second, this simple law must be applied very many times during the course of each time step, and all of the force calculations could in principle be performed in parallel.

N-body Methods

Unfortunately, direct N-body methods, in which the force between every pair of bodies is calculated in each step, are not practical for large stellar systems such as galaxies. Galaxies have so many stars that the gravitational potential is very smooth, and gravitational encounters between individual stars are rare. Consequently, such systems are described as collisionless. The actual number of stars is far too large for direct N-body simulation ($N \approx 10^{11}$), and computations with feasible values of N ($\approx 10^4$) produce many more 2-body encounters (collisions) than occur in the real systems.

Most studies of the dynamics of collisionless stellar systems have used a modified N-body technique in which the density distribution is binned and smoothed before calculating the gravitational potential (*e.g.*, Ref. 1). The calculation can then include many more particles than direct N-body simulations ($N \approx 10^5$), and 2-body encounters are prevented because the potential is smoothed. Even this method, however, suffers from graininess due to the finite number of particles in each zone; this graininess manifests itself as a numerical "noise" in the calculation. It also limits the resolution which is achievable with a given number of particles. Another problem is that it is necessary to use a "softened" gravitation potential instead of the real $1/r$ potential to avoid serious instabilities. Sellwood (Ref. 1) has shown that this significantly changes even the linear behavior of the system.

Phase Space Methods

Numerical noise can be eliminated if the stellar system is treated as a fluid in phase space. The evolution of such a system is described by the collisionless Boltzmann equation. A Japanese group has done some calculations using this approach (Refs. 2–5). Their results are very interesting and indicate that the numerical noise in the N-body methods may have led to spurious results. For example, Nishida (Ref. 5) found that the bar instability of a thin stellar disk can be suppressed by the presence of a small bulge component; previous N-body calculations had led to

the opposite conclusion, apparently because their resolution was inadequate.

This paper reports on the implementation on the Massively Parallel Processor (MPP) of a new numerical technique for phase space techniques. The technique is briefly described, some results are presented, the MPP implementation is discussed, and some improvements are suggested for future MPP-like machines which would make them more effective for this type of problem.

THE NUMERICAL TECHNIQUE

The numerical method described here was developed by the author and P. R. Woodward (Refs. 6, 7). Phase space is divided into an Eulerian (fixed) grid. This grid is two-dimensional (2-D) for spherical systems without angular momentum, 3-D for spherical systems with angular momentum, or 4-D for disk systems. The grid need not be rectangular; for the disk problem, for example, a polar grid can be used. For each zone of the grid, we store the mean density in the zone and all the first order moments in the phase space coordinates. Thus, for a 2-D grid, there are 3 moments: the mean density, the x-moment, and the v-moment. The moments in a zone imply a unique distribution of density within that zone which is linear in all coordinates. For a 2-D grid, the density is $f(x,v) = \langle f \rangle + f_x x + f_v v$, where $\langle f \rangle$ is the mean density and f_x and f_v are proportional to the x and v moments.

A version of this scheme using second order moments has also been implemented; however, this scheme requires much more memory than the linear scheme, especially for higher dimensional problems. The limited memory of the MPP made it desirable to use only linear moments.

A time step consists of calculating new values for the moments in each zone. The x and v motion is split into two half steps; by taking steps in the order x-v-v-x, second order accuracy in the time step is achieved. Fluxes of each moment across the zone boundaries are computed from the acceleration and velocity of material in the zone. The moments are updated using the fluxes so that mass and momentum are conserved during the time step.

RESULTS OF CALCULATIONS

The figures show the results for two calculations of the gravitational collapse of a 1-D, spatially periodic system consisting of infinite sheets of stars. For both examples, the spatial period is 10 Jeans lengths, the mean space density is 1, the Gaussian velocity dispersion is 1, and the 2-D phase space grid is 128×128.

The initial conditions differ for the two examples. The example in Figure 1 has a density perturbation of the form $\delta\rho = A \cos 2\pi x/L$, where L is the length of the grid and $A = 0.01$. The example in Figure 2 has $\delta\rho = A[\cos(2\pi x/L) + 3\cos(6\pi x/L)]$.

The figures display the evolution of the system with time. One spatial period is shown in each figure. For each time, a contour plot of the phase space density is shown, with velocity on the vertical scale and position on the horizontal scale. The contours are logarithmically spaced.

At the beginning of the calculation, the material is concentrated near zero velocity and is almost uniformly distributed in space. The initial perturbation causes material to feel a gravitational acceleration toward the center. This increases (decreases) the velocity of material to the left (right) of center, so that it moves up (down) in the figure. The higher velocity material then moves toward the center, which leads to the winding up of the original phase space distribution. As time passes, the original material becomes so tightly wound that it is no long resolved by the grid; it then appears smooth.

These two calculations reveal the interesting feature that the final phase space density distribution is not identical for the two collapses, so that there is some "memory" of the initial perturbation. This is of interest in understanding the formation and evolution of galaxies; it hints that buried in the present-day structure of galaxies there may still be information about the conditions under which galaxies were formed.

IMPLEMENTATION ON THE MPP

The implementation of this numerical scheme on the MPP is relatively straightforward. The 2 dimensional phase space grid is mapped directly to the 128×128 array of processors in the MPP, with one zone per processing element. The nearest neighbor connectivity of the MPP is precisely what is required for the flux-based method described above. The programmable boundary conditions for the MPP array allow the boundaries of most problems to be treated easily.

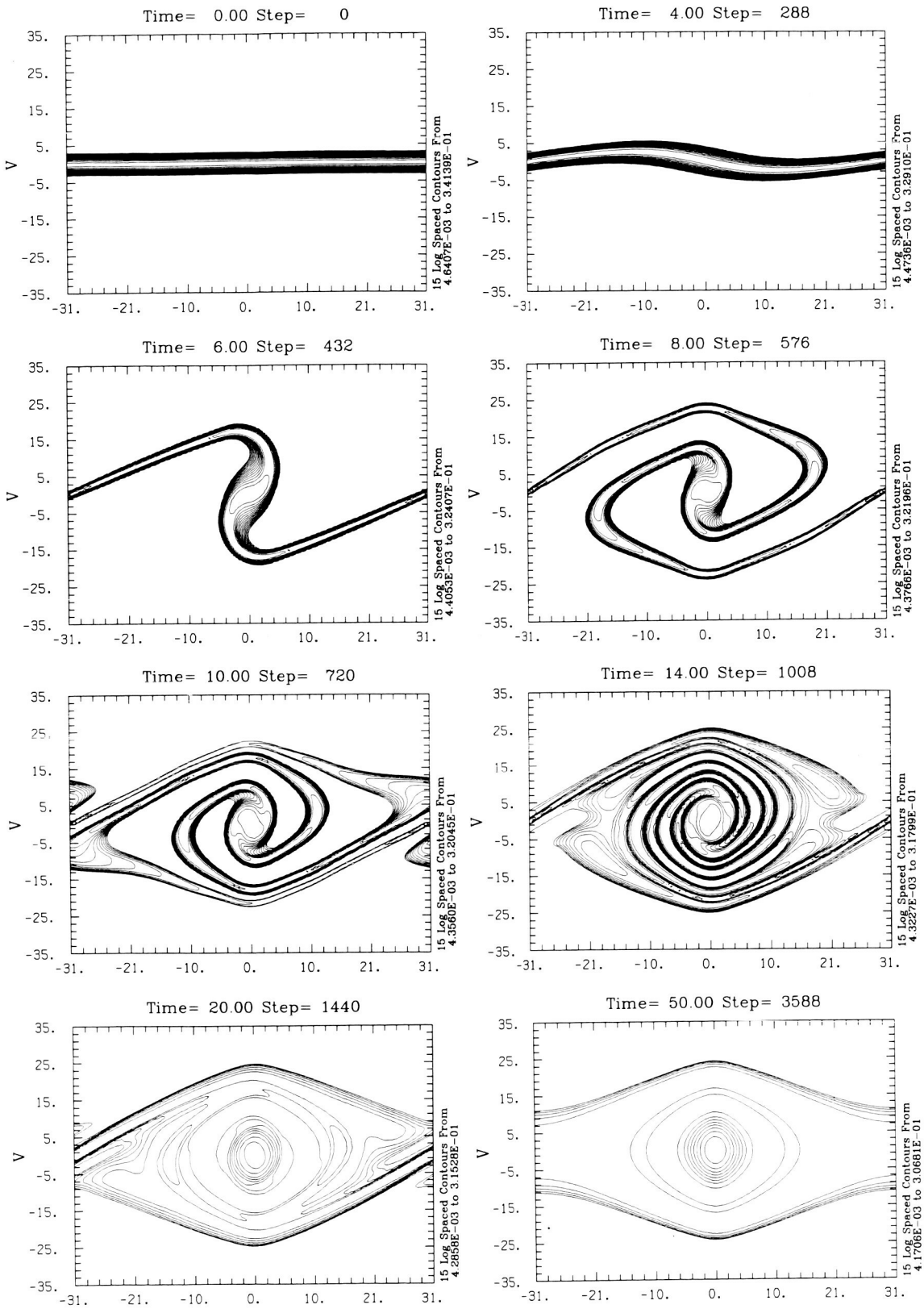

Figure 1. Evolution of phase space density for Example 1 (see text for details).

The MPP architecture is very effective for problems like the one described here which require intensive floating point computations. The MPP is divided into a scalar processing unit (the MCU) and a parallel processing array; the MCU places requests for array operations into a call queue. The time for a floating point add or multiply in the array is long compared to any computations which must be formed in the MCU; consequently, the MCU keeps the call queue full, and the array is working continuously. The array-side code consists of primitives which can (in principal) be fully optimized; the user code resides in the MCU and need not be optimized at all except in its organization of calls to the array, because the execution time of the MCU code has negligible effect on the total MPP time required for the calculation. The net effect of all this is that execution time on the MPP is very close to what one would calculate from simply counting the number of floating point operations and multiplying by the time per operation.

Performance of the MPP Code

The execution speed of the MPP code is about 50 times faster than a VAX 8600. This is comparable to the speed of the same method on a single processor of a Cray-XMP/48, even though the much greater vector length of the MPP (16384 versus 64) requires about 5 times more multiplies than the Cray within the inner loop. For a grid with uniform spacing, at each time step there are many coefficients which need be calculated only once for each row of the grid. In the Cray version of the code, the short vector length allows these coefficients to be calculated outside the inner loop, but on the MPP the coefficients must be re-calculated for every zone. This greatly increases the number of operations within the inner loop, so that even though the MPP can perform more multiplies per second than the Cray, the speeds of the machines are similar for this scheme.

For a problem using a grid with non-uniform spacing, the MPP program would be faster than the Cray by a factor of 3 or more, because nearly all the calculations would have to be performed within the loop by both the Cray and the MPP.

It appears that a 3 or 4 dimensional phase space calculation will also run several times faster on the MPP than on a Cray. However, higher dimensional problems require much more memory than is available within the MPP array. They can be implemented on the MPP if some limitations which currently exist can be alleviated.

Limitations of the MPP

The code development time was much greater on the MPP than on the Cray, and the MPP program is more difficult to modify (*e.g.*, to use grids larger or smaller than 128×128). This can be attributed to limitations of the MPP which fall into two classes: those which are determined by the hardware (which probably cannot be fixed for the current machine), and those which are determined by software (which *can* be fixed).

Hardware Limitations – Hardware limitations of the MPP include:

(1) the lack of direct I/O facilities, and

(2) the severe shortage of local memory for each processing element in the array.

The MPP is slowed significantly compared to a Cray if many intermediate results must be stored on disk. On the MPP, all input and output (I/O) is performed through the front-end VAX. The rate at which the VAX can respond to MPP I/O requests is often the limiting factor in the performance of the program. In contrast, the Cray has dedicated, high speed disks so that the I/O usually has little effect on the execution time for the program.

Each node in the array has only 1024 bits of local memory. Some of this memory is required for the system, leaving room for about 25 32-bit floating point numbers for all variables and temporary storage. This local memory is so small that it is difficult to implement a modestly complicated algorithm such as the one discussed in this paper. It is necessary for the programmer to manage memory very carefully, with temporary variables being constantly re-used within modules. This sort of coding is highly prone to errors, and the resulting bugs are difficult to find and eradicate.

Even using the staging memory, which contains a much larger reservoir of storage, some algorithms will be very inefficient. The need for numerous temporary variables may force one to the unpleasant prospect of having to roll temporary variables in and out of staging memory frequently during each time step.

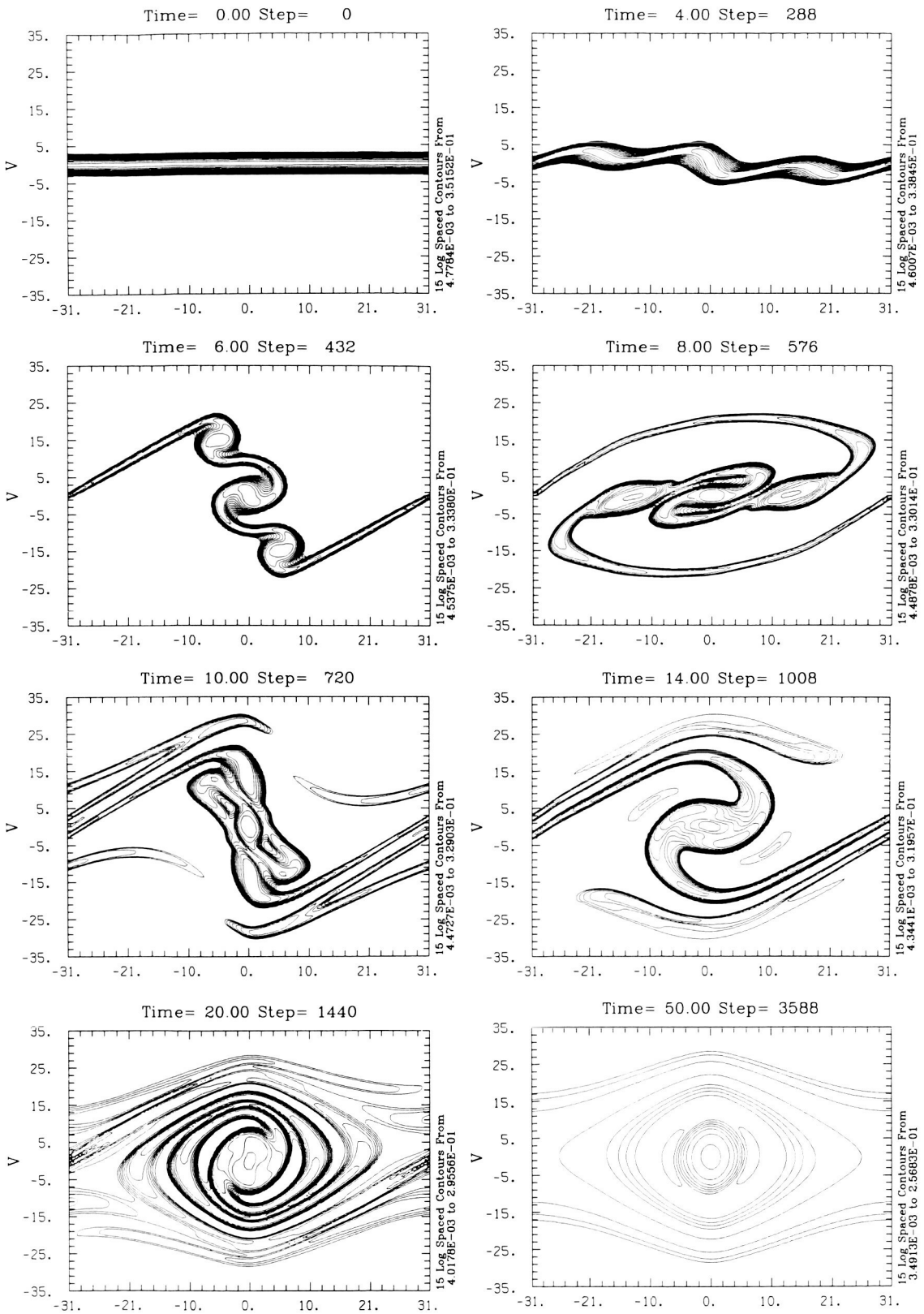

Figure 2. Evolution of phase space density for Example 2 (see text for details).

Software Limitations – The hardware problems are exacerbated by shortcomings of the software that currently exists on the MPP. Neither disk I/O nor the staging memory are integrated into the high-level language MPP Pascal, which was used for the implementation of the phase space method. The existing utility routines do not make use of many of the hardware capabilities of the MPP. For example, the hardware is capable of transferring data to and from the staging memory in parallel with array operations and with no interaction with the VAX; this is not implemented in current high level routines.

Software limitations of the MPP Pascal compiler also make the shortage of local memory harder to deal with. The compiler uses far too many temporary locations in memory for any reasonably complicated assignment statements. Consequently, the programmer is required to do the compiler's job of breaking each equation down into binary operations and managing the intermediate results which are generated. This is an error-prone process which would be much more efficiently left to the compiler.

The local memory of the array is so small (when used for floating point numbers) that it might be best to view it as similar to the general purpose registers in many scalar computers. There are usually not enough registers to hold all of one's data for the duration of the problem; instead, the data is stored in main memory (staging memory, for the MPP) and is brought in a piece at a time for processing. In the MPP, approximately four floating point multiplies can be performed in the time required to move a 32-bit number from the stager to the array; this is slow compared to many scalar computers, but it still may be short enough for the "register model" of array memory to be viable. The time required for this transfer becomes negligible if it can be carried out in parallel with array operations, as is permitted by the hardware.

Requiring the programmer to manage these data transfers on the MPP is equivalent to requiring the programmer to write in assembly language on a scalar machine. An advantage of treating array memory as registers is that a great deal of effort has been devoted to developing compilers which make efficient use of registers. Many of the techniques that have been developed for compilers on scalar computers thus might be fruitfully applied to memory management on the MPP. This would dramatically reduce the software development time on the MPP.

FUTURE MPP-LIKE MACHINES

Future MPP-like machines should be orders of magnitude faster than future short vector machines like the Cray. They will probably always be more difficult to program than the Cray, but the rewards for the effort will be great.

Many of the features of the current MPP are beautifully suited to large numerical calculations like the one described here. Such features should be retained in future MPPs:

(1) The separation of the array and the scalar processor allows very efficient use of the array, as discussed above.

(2) The staging memory is an extremely useful device which does much to compensate for the nearest neighbor connectivity of the array. It allows problems with more than 2 dimensions to be handled effectively.

(3) Programmable boundary conditions for the array allow the efficient treatment of many problems. Some additional boundary conditions might be useful (*e.g.*, a twisted connection in which the top of the last column is connected to the bottom of the first column.)

On the other hand, there are some aspects of the MPP architecture that could definitely be improved for this type of problem:

(1) The local memory needs to be much bigger.

(2) Some directly connected, high speed I/O devices are needed. These should include direct disk storage and an image display.

(3) As the array gets bigger, it will be important to have faster ways to load data into it and to shift data across it.

(4) Finally, the hardware is important, but the accompanying software development must not be forgotten.

Much of the current research in parallel computer architectures is directed toward developing machines with more sophisticated connectivity than the MPP. Machines with butterfly or hypercube connections allow easy communication between any pair of processors. This is indeed of critical importance for many algorithms, which simply could not be implemented on the MPP.

For example, the most promising N-body methods organize the particles into a binary tree and communicate information about masses and positions along the branches of the tree. The MPP would be very poorly suited for such an algorithm.

On the other hand, the MPP is very well suited to large numerical problems involving fluid flow, in which the connectivity of the physics underlying the calculation is usually local. It is also efficient for image processing, the task for which it was designed. It would be very premature to conclude that the nearest neighbor architecture of the MPP should be abandoned. The simplicity and easy expandability of MPP-like arrays should make such machines attractive as number-crunching engines for the foreseeable future.

CONCLUSIONS

Phase space techniques have several advantages over N-body calculations for collisionless stellar systems: there is no need for a "softened" gravitational potential; there is no numerical noise; and the resolution can be made higher by choosing the grid appropriately. It is also much easier on machines like the MPP to implement phase space fluid schemes than to implement efficient N-body schemes.

Phase space techniques do have one significant disadvantage compared to particle methods: they require more computational time and memory for multi-dimensional problems. For example, a 3-D problem requires a 6-D phase space calculation. Since most of phase space is empty, much of the computing time in such a problem would be wasted. By comparison, a particle calculation represents the ultimate Lagrangean grid calculation: all of the computational effort is expended where the matter is located.

The MPP is very well suited to fluid schemes, though some suggestions for improvements in hardware and software have been made. The most important change for future MPPs would be the addition of much more local memory for each node.

REFERENCES

1. Sellwood, J. A. 1983, *J. Comp. Phys.*, **50**, 337.
2. Fujiwara, T. 1981, *Pub. A. S. J.*, **33**, 531.
3. Watanabe, Y., *et al.* 1981, *Pub. A. S. J.*, **33**, 541.
4. Nishida, M. T., *et al.* 1981, *Pub. A. S. J.*, **33**, 567.
5. Nishida, M. T. 1986, *Ap. J.*, **302**, 611.
6. Woodward, P. R., and White, R. L. 1986, *J. Comp. Phys.*, in preparation.
7. White, R. L. 1986, in proceedings of the Princeton meeting on *The Use of Supercomputers in Stellar Dynamics*, ed. P. Hut.