THE TREE PARTICLE-MESH N-BODY GRAVITY SOLVER

PAUL BODE,1 JEREMIAH P. OSTRIKER,1 AND GUOHONG XU2

ABSTRACT

The tree particle-mesh (TPM) N-body algorithm couples the tree algorithm for directly computing forces on particles in a hierarchical grouping scheme with the extremely efficient mesh-based PM structured approach. The combined TPM algorithm takes advantage of the fact that gravitational forces are linear functions of the density field. This allows the use of domain decomposition to break down the density field into many isolated high-density regions containing a significant fraction of the mass but residing in a very small volume. At low redshift, ~1/3 of the particles in a typical large cosmological simulation can be assigned to ~10^4–10^5 separate groups occupying ~10^-2.5 of the total volume. In each of these high-density regions, the gravitational potential is computed via the tree algorithm supplemented by tidal forces from the external density distribution. For the bulk of the volume, forces are computed via the PM algorithm; time steps in this PM component are large compared to individually determined time steps in the tree regions. Since each tree region can be treated independently, the algorithm lends itself to very efficient parallelization using message passing. We have tested the new TPM algorithm (a refinement of that originated by Xu) by comparison with results from Ferrall & Bertchinger’s P^3M code and find that, except in small clusters, the TPM results are at least as accurate as those obtained with the well-established P^3M algorithm, while taking significantly less computing time. Production runs of 10^9 particles indicate that the new code has great scientific potential when used with distributed computing resources.

Subject headings: dark matter — methods: n-body simulations — methods: numerical

1. INTRODUCTION

In addition to the rapid increase of available computing power, the rise of the use of N-body simulations in astrophysics has been driven by the development of more efficient algorithms for evaluating the gravitational potential. Efficient algorithms with better scaling than ~N^2 take two general forms. First, one can introduce a rectilinear spatial grid and, taking advantage of fast Fourier transforms (FFTs), solve Poisson’s equation on this grid in Fourier space; this is the well-known particle-mesh (PM) method, which, while very fast, limits the spatial resolution to the grid spacing. To gain finer resolution, one can introduce smaller subgrids (e.g., the ART code of Kravtsov, Klypin, & Khokhlov 1997; see also Norman & Bryan 1999); alternatively, one can compute the short-range interactions directly (the particle-particle–particle-mesh, or P^3M method; Efstathiou et al. 1985; Ferrall & Bertchinger 1994). One widely used code (AP^3M) combines both of these refinements (Couchman 1991; Pearce & Couchman 1997). The second general approach is to approximate long-range interactions that are less important to an accurate determination of the force, by grouping together distant particles. These are known as tree methods, since a tree data structure is used to hold the moments of the mass distribution in nested subvolumes (Barnes & Hut 1986; Hernquist 1987). ART and AP^3M are discussed in more detail by Knebe et al. (2000); for a review of the field, see Couchman (1997).

All these algorithms are more difficult to implement on parallel computers with distributed memory than on single-processor machines. Gravity acts over long scales, and gravitational collapse creates highly inhomogeneous spatial distributions, yet with parallel computers one needs to limit the amount of communication and give different processors roughly equal computing loads. The problem is one of domain decomposition: locating spatially compact regions and deciding which data are needed to find the potential within that region.

Xu (1995) introduced a new N-body gravity solver that deals with this problem in a natural way. The tree particle-mesh (TPM) approach is similar to the P^3M method, in that the long-range force is handled by a PM code and the short-range force is handled by a different method—in this case using a tree code, with the key difference that the tree code is used in adaptively determined regions of arbitrary geometry. In this paper we describe several improvements to the TPM code, and compare the results with those obtained by the P^3M method. Our goal was to improve and to test the new algorithm while designing an implementation that could be parallelized efficiently and that was optimal for use as a coarse-grained method suitable for distributed computational architectures, including those having large latency. Section 2 describes the method, §3 the basis (density threshold) for domain decomposition, §4 the parallelism of the implemented algorithm (using message passing), and §5 tests and compares with the well-calibrated P^3M algorithm.

The implementation presented in this paper is oriented toward a specific cosmological problem, the formation of large clusters, and we discuss it in that setting. However, this algorithm could be used for many particle-simulation applications, both in astrophysics and in other fields; it should be beneficial in situations where the density distribution allows one to divide the particles into many isolated groups. Thus, we will conclude this section with a brief summary of the specific cosmological context, for those unfamiliar with it. A large cubical volume is simulated with periodic boundary conditions. The simulation begins in the linear regime; particles are displaced slightly from a
uniform grid, giving Gaussian perturbations to a nearly constant density field. The particles are followed as they move under their mutual gravitational attraction. Over time, gravitational instability causes the initially small overdensities to collapse, forming highly dense halos (with central densities a factor of ~$10^3$ higher than the average). These halos are distributed along filaments surrounding large, low-density voids. The TPM algorithm was developed to deal with this highly inhomogeneous structure.

2. THE TPM ALGORITHM

The basic idea behind the TPM algorithm is to identify dense regions and use a tree code to evolve them; low-density regions and all long-range interactions are handled by a PM code. A general outline of the algorithm is:

1. Find the total density on a grid.
2. Based on the grid density, decompose the volume into a background PM volume and a large number of isolated high-density regions. Every particle is then assigned to either the PM background or a specific tree.
3. Integrate the motion of the PM particles (those not in any tree) using the PM gravitational potential computed on the grid.
4. For each tree in turn, integrate the motion of the particles, using a smaller time step if needed; forces internal to the tree are found with a tree algorithm (Hernquist 1987), added to the tidal forces from the external mass distribution taken from the PM grid.
5. Step global time forward, go back to step 1.

In this section we consider certain aspects of this process in detail, and conclude with a more complete outline of the algorithm.

2.1. Spatial Decomposition

We wish to locate regions of interest, which will be treated with greater resolution in both space and time; for the purposes of cosmological structure formation, this translates into regions of high density. It is also necessary that these regions remain physically distinct during the long PM time step (determined by the Courant condition), so that the mesh-based code accurately handles interactions between two such regions. The process we use can be thought of as finding regions enclosed by an isodensity contour. If one imagines the isodensity contours through a typical simulation at some density threshold $\rho_{\text{thr}} > \bar{\rho}$, space is divided into a large number of typically isolated regions with $\rho > \rho_{\text{thr}}$ plus a multiply connected low-density background filling most of the volume.

To locate isolated, dense regions, we begin with the grid density, which has already been calculated by the PM part of the code. Each grid cell that is above a given threshold density $\rho_{\text{thr}}$ is identified and given a unique positive integer key (the choice of $\rho_{\text{thr}}$ is discussed in §3). Cells are then grouped by a friends-of-friends approach: for each cell with a nonzero key, the 26 neighboring cells are examined, and if two adjacent cells are both above the threshold, they are grouped together by making their keys identical. The end result is isolated groups of cells, each separated from the other groups by at least one cell. If a wider separation between these regions is desired, one can examine a larger number of neighboring cells. The method is “unstructured” in the sense that the geometry of each region is not specified in advance, except insofar as it is singly connected. The shape of the region can be spheroidal, planar, or filamentary, as needed.

To assign particles to trees, the process used to find the density on the grid (described in the next section) is repeated. This involves locating the grid cell to which some portion of a particle’s mass is to be added, so it is easy to check at the same time whether that cell has a nonzero key, and if it does, to add that particle to the appropriate tree. Thus, any particle that contributes mass to a cell with density above the threshold is put into a tree. Because of the spatial separation of the active regions (they are buffered by at least one nontree cell), a particle will only belong to one tree even though it contributes mass to more than one cell.

An example of this in practice is shown in Figure 1. In the bottom panel, all particles in a small piece of a larger simulation are shown in projection. The grid and the location of active cells are shown in the top panel; each isolated region is indicated by a unique numerical key. In a couple of cases it appears that different regions are in adjacent cells, but in fact they are separated in the third dimension; the region shown is 10 cells thick. In the lower of the middle two panels, the particles assigned to trees are shown with different symbols indicating membership in different trees. In the other panel the residual PM particle positions are plotted, demonstrating their much lower density contrast as compared to those in trees.

2.2. Force Decomposition

As in Xu (1995), the force is decomposed into that internal to the tree and that due to all other mass:

$$ F = F_{\text{internal}} + F_{\text{external}}. \quad (1) $$

However, we do this in a different manner, described in this section, than was done in Xu (1995).

The first step in obtaining the particle accelerations is to obtain the PM gravitational potential. The masses $m_p$ of the $N$ particles (including those in trees) are assigned to the grid cells using cloud-in-cell (CIC) weighting:

$$ \rho_{\text{all}}(i, j, k) = \sum_{p=1}^{N} m_p w_i w_j w_k, \quad (2a) $$

$$ w_i = \begin{cases} 1 - |x_p - i| & \text{for } |x_p - i| < 1, \\ 0 & \text{otherwise} \end{cases} \quad (2b) $$

where $x_p$ is a particle’s $x$-coordinate in units where the grid spacing is unity. The potential $\Phi_{\text{PM,all}}$, assuming periodic boundary conditions, is then found by solving Poisson’s equation using the standard FFT technique (Hockney & Eastwood 1981).

Once a tree has been identified, we wish to know the forces from all the mass not included in that tree; thus the contribution of the tree itself must be removed from the global potential. This step will have to be done for each tree in turn. The density is found exactly as before, except now summing over only the particles in the tree:

$$ \rho_{\text{tree}}(i, j, k) = \sum_{\text{tree}} m_p w_i w_j w_k. \quad (3) $$

Using this density, we solve Poisson’s equation again, except that nonperiodic boundary conditions are used (Hockney & Eastwood 1981). The resulting potential, $\Phi_{\text{NP,tree}}$, is the contribution the tree made to $\Phi_{\text{PM,all}}$ without
counting the ghost images due to the periodic boundary conditions of the latter. The force on a tree particle exerted by all the mass not in the tree (including the periodic copies of the tree) is then

\[ F_{\text{external}} = \sum_{i,j,k} w_i w_j w_k \nabla \Phi_{\text{PM,all}} - \sum_{i,j,k} w_i w_j w_k \nabla \Phi_{\text{NP,tree}}. \]  

(4)

Thus, tidal forces within a tree region are computed on the mesh scale in a consistent manner, with interpolation used as required to find the forces on individual particles.

Calculating the nonperiodic potential with FFTs involves using a grid that is 8 times larger in volume than that containing the actual mesh of interest, but since trees are compact and isolated regions, the volume of the larger grid that is nonzero is quite small. Thus, the FFT computed for each tree can be done on a smaller grid as long as the grid spacing remains the same as for the larger periodic FFT; we do this by embedding the irregular tree region in a cubic subgrid, padding with empty cells as needed.

The final step is to calculate the internal forces, \( F_{\text{internal}} \), for each tree. We do this with the tree code of Hernquist (1987). Since the periodic nature of the potential was taken
into account in finding the external forces, no Ewald summation is needed. Time stepping is handled in the same manner as in Xu (1995). That is, the PM potential is determined at the center of the large PM time step, and each tree has its own, possibly smaller, time step. There are a couple of slight differences: in equation (15) of Xu (1995) we use the parameter \( \beta = 0.05 \), and we decrease \( \delta t_{\text{tree}} \) so that 97.5% of the tree particles satisfy \( \delta t_{i} \geq \delta t_{\text{tree}} \).

2.3. Detailed Outline

To sum up this section, we give a more detailed outline of the code. All particles begin with the same time step, \( \Delta t = \Delta t_{\text{PM}} \); the velocities are given at time \( t \) and the positions at time \( t + \Delta t/2 \) (as described in Xu 1995).

1. Using the density from the previous step, we identify all particles belonging to trees, and to which tree (if any) each particle belongs (§ 2.1).
2. The time step for each tree is computed, and particle positions are adjusted if \( \Delta t \) has changed for that particle (Hernquist & Katz 1989). This can occur if a particle joins or leaves a tree, or if the tree time step has changed.
3. The total density due to all particles at time \( t + \Delta t_{\text{PM}}/2 \) is found on a grid using equation (2). The potential \( \varphi_{\text{PM,all}} \) is found from this density, and the PM acceleration at midstep is found for each particle.
4. Each tree is then dealt with in turn. First, the tree contribution to the PM acceleration is removed, as described in § 2.2. Next the tree is stepped forward with a smaller time step using the tree code of Hernquist (1987), with the external forces included.

5. All particles not in trees are stepped forward using the PM acceleration. The global time and cosmological parameters are updated, completing the step.

3. THE DENSITY THRESHOLD

In § 2.1 the threshold density, \( \rho_{\text{thr}} \), was introduced to demarcate dense regions that would be followed with higher resolution. The best choice of this parameter depends on a number of considerations. One could set \( \rho_{\text{thr}} \) to be such a low value that nearly all particles are in trees, or so that only one large tree exists, thereby destroying the efficiency that the TPM algorithm is designed to give. On the other hand, too high a value would leave many interesting regions computed at the low resolution of the PM code. When modeling gravitational instability, one must also keep in mind that the density evolves from having only small overdensities initially to a state in which there are a few regions of very large overdensity; thus, the ideal threshold will evolve with time. With these considerations in mind, we base \( \rho_{\text{thr}} \) on the grid density as

\[
\rho_{\text{thr}} = A \bar{\rho} + B \sigma ,
\]

where \( \bar{\rho} \) is the mean density in a cell, and \( \sigma \) is the standard deviation of the cell densities. With this equation, the first two moments of the density distribution are used to fix \( \rho_{\text{thr}} \) in an adaptive manner. The coefficient \( A \) is set to prevent the selection of too many or too large trees when \( \sigma \) is small; its value will be near unity. The choice of \( B \) will determine what fraction of particles will be placed in trees when \( \sigma \) is large. This choice depends on the parameters of the simulation, such as the cosmological model (including the choice of \( \sigma_{\text{M}} \) and the size of a grid cell). We choose a value of \( B \) that will place \( \sim 1/3 \) of the particles in trees at the end of the simulation.

Figure 2 shows how tree properties vary over the course of a large low-density cold dark matter (LCDM) simulation, using \( A = 0.9 \) and \( B = 4.0 \) in equation (5). The value of \( \sigma \) begins at 0.1, so at high redshift \( \rho_{\text{thr}} \lesssim 1.5 \bar{\rho} \). This leads to a large number of trees that are low in mass and diffuse. As time goes on, these slight overdensities collapse and merge together, resulting in denser concentrations of mass. In addition, \( \sigma \) becomes larger (increasing to 4.1 by the end of the simulation), so a larger concentration of mass is needed before a region is identified as a tree. Thus, the original distribution of trees evolves into one with fewer trees, but at higher masses (although at any given time the masses of trees roughly follow a power-law distribution). The typical volume within tree regions also increases with time, but the total volume covered by trees (measured by the number of cells above \( \rho_{\text{thr}} \)) decreases. Given the roughly lognormal distribution of density resulting from gravitational instability, the total volume in tree regions is less than 1% even when they contain \( \sim 30\% \) of the mass. The rise in \( \rho_{\text{thr}} \) means that the size of the smallest tree found also rises, from 4 to 40 particles over the course of this run. This raises an issue that must be noted when understanding the results of a TPM run: the choice of \( \rho_{\text{thr}} \) introduces a minimum size below which the results are no better than in a PM code. This is discussed in more detail in § 5.

4. PARALLELISM

One of the strengths of the TPM algorithm is that after the PM step, each tree presents a self-contained problem:
given the particle positions, velocities, and tidal forces, the tree stepping can be completed without the need to access any other data, since the effect of the outside universe is summarized by the tidal forces in the small tree region. This makes the tree part of the code intrinsically parallel. What makes such a separation possible is that during the multiple time steps required to integrate particle orbits within a dense tree region, the tidal forces can be deemed constant; the code is self-consistent in that the density on the PM grid is only determined on the Courant timescale for that particle distribution.

Our parallel implementation of the TPM method uses a distributed-memory model and the MPI message-passing library in order to maximize the portability of the code. The PM portion of the code is made parallel in a manner similar to that described in Bode, Xu, & Cen 1996. This scales well, and takes a small fraction of the total time as compared to the tree portion of the code.

Two steps are made to ensure load balancing the tree part of the code. First, trees are distributed among processors in a manner intended to equalize the amount of work done. The time it takes for a particular tree to be computed depends on the size of the tree, the cost of computing the force scaling roughly as $N \log N$. As trees are assigned to processors, a running tally is kept of the amount of work given to each node, and the largest unassigned tree is assigned to the processor given the least amount of work. The tree particles are then distributed among the processors, and each processor deals with its assigned trees, moving from the largest to the smallest. There is also a dynamic component to the load balancing: when a node has completed all of its assigned trees, it queries another process to see if that one is also finished. If that process still has an uncomputed tree remaining in its own list, it sends all the necessary tree data to the querying node. That node then evolves the tree and sends the final state back to the node that had the tree originally. Thus nodes that finish earlier than expected do not remain idle.

The scaling of the code is shown for two different size problems in Figure 3; the times shown are for when the underlying LCDM model is at low redshift ($z = 0.5$), meaning that clustering is significant and calculating tree forces dominates the CPU time. At higher redshift, when the trees are less massive and more diffuse, the timing would be more like that of a PM code (this can be seen from Table 1). These timing tests were run on an SGI Origin 2000 with 250 MHz chips; the scaling on a PC cluster with a fast interconnect was found to be quite similar. The 512$^3$ model is the one shown in Figure 2; it scales reasonably well up to the largest number of processing elements (NPE) we attempted; compared to NPE = 32, the efficiency is better than 90% at NPE = 128, and 80% at NPE = 256. When using 32 nodes, the code required 512 MB per node, so we

![FIG. 3.—Timing as a function of the number of processors, for models at $z = 0.5$. The labels give the number of particles, which equals the number of cells in each case. The thin dotted lines show the slope expected for a perfect scaling of $\sim NCPU^{-1}$. The time shown is for one PM step; the number of steps for individual trees varies, up to 10 steps for the larger ones.](image)

### Table 1

| $N$   | NPE  | $z = 9$ | $z = 0.5$ |
|-------|------|---------|-----------|
|       |      | PM     | DD   | Tree | PM     | DD   | Tree |
| 256$^3$ | 4    | 21.9   | 70.8 | 13.8 | 19.3   | 117.1| 2695.0 |
|       | 8    | 11.1   | 35.8 | 7.0  | 9.7    | 68.8 | 1350.0 |
|       | 16   | 5.6    | 18.6 | 3.8  | 5.5    | 37.8 | 700.0  |
|       | 32   | 2.9    | 10.0 | 2.0  | 3.4    | 160.1| 339.5  |
|       | 64   | 2.0    | 6.8  | 1.0  | 2.4    | 140.1| 175.5  |
|       | 128  | 1.4    | 5.7  | 0.5  | 2.0    | 234.0| 84.5   |
|       | 256  | 1.6    | 11.7 | 0.2  | 3.1    | 252.6| 38.2   |
| 512$^3$ | 32   | 27.0   | 80.2 | 8.4  | 24.5   | 133.5| 1085.0 |
|       | 64   | 13.8   | 48.5 | 4.2  | 14.4   | 67.2 | 545.0  |
|       | 128  | 7.6    | 33.4 | 2.1  | 9.9    | 44.7 | 275.5  |
|       | 256  | 13.6   | 38.4 | 1.1  | 12.9   | 38.8 | 144.5  |
| 1024$^3$ | 256  | 69.17  | 136.9| 9.5  | 87.1   | 200.8| 1433.0 |

Note: PM: The PM portion of the code. DD: Time spent preparing trees, including domain decomposition, tidal force calculation, and any load imbalance. Tree: The potential computation for tree particles.
did not try any smaller runs. The $256^3$ times are for the same LCDM model, but with a smaller box size (150 Mpc $h^{-1}$) and $\rho_{th} = 0.85\rho + 4.0\sigma$. Since the largest nonlinear scale is a larger fraction of the box size, a greater fraction of particles (37%) are placed in trees, and the largest tree contains a greater proportion of the mass. This $256^3$ model scales extremely well from 4 to 16 processors, but drops to 70% efficiency at 32 nodes, and beyond 64 nodes does not speed up at all. The reason for this is that the largest tree in this simulation contains 1% of all particles, which means that this one tree takes a few percent of the entire CPU time devoted to trees. As NPE is increased, the time it takes to complete this one tree becomes the major part of the total time. The solution to this problem is to allow more than one processor to work on the same tree, which is quite possible (e.g., Davé, Dubinski, & Hernquist 1997 and references therein; see also Xu 1995).

The division of the total time between different components of the code is shown in Table 1 for both low and high redshift. At low redshift, the tree calculations dominate the total time (as long as this part of the code is load balanced; the rise in overhead for the $256^3$ model when NPE $\geq 32$ is due to imbalance, as discussed above). At high redshift the trees are smaller, so the overhead related to domain decomposition takes a large fraction of the total time; the main difference between the two redshifts is the rising cost of the tree calculations as trees become more massive and require more time steps. Comparison with the $P^3M$ code of Ferrell & Bertschinger (1994; made parallel by Frederic 1997) shows that TPM (with 30% of the particles in trees) takes slightly less time than $P^3M$ if all the trees keep to the PM time step. Allowing trees to have individual time steps speeds up the TPM code by a factor of 3–4. In the present implementation, particles within the same tree all use the same time step; implementing multiple time steps within trees could further save a significant amount of computer time (roughly another factor of 3) without loss of accuracy.

The memory per process used by our current implementation is $20N/NPE$ reals when there is one cell per particle. This includes for each particle $x$, $v$, $a$, and three integer quantities (a particle ID number, a tree membership key, and the number of steps per PM step). The remaining space is used by the mesh part of the code, and reused as temporary storage during the tree stepping. Because the grid density from the previous step is saved, the memory used could be reduced to $17N/NPE$ at the cost of computing the density twice per step.

The $1024^3$ point shown in Figure 3 is for the same cosmological model and box size as the $512^3$ run, but with 8 times as many particles. This run shows the great potential of the TPM algorithm. At lower redshifts, over 80% of the computational time is spent finding tree forces—precisely the portion of the code that involves no communication; thus, a run of this size would be able to efficiently utilize even more processors. This does not necessarily mean using a larger supercomputer; rather, one could use networked PCs or workstations. These distributed resources could be used to receive a single tree or small group of trees, do the required time-stepping in isolation, and send back the final state. The time required to evolve a single tree varies from less than a second to a couple of minutes, so even in situations with a high network latency the cost of message passing need not be prohibitive.

5. Tests of the Code

To test how the code performs in a standard cosmological simulation, we ran both TPM and the $P^3M$ code of Ferrell & Bertschinger (1994) with the same initial conditions. The test case contains $128^3$ particles in a 150 Mpc $h^{-1}$ box, with a flat LCDM cosmological model close to that proposed by Ostriker & Steinhardt (1995): $\Omega_m = 0.37$, $\Lambda = 0.63$, $H_0 = 70$ km s$^{-1}$ Mpc$^{-1}$, $\sigma_8 = 0.8$, and tilt $n = 0.95$. The softening length of the particles is $\epsilon = 18.31$ kpc $h^{-1}$. The number of mesh points in the PM grid was $256^3$ for the $P^3M$ run and $128^3$ for TPM. The TPM threshold density was $\rho_{th} = 0.85\rho + 4.0\sigma$, so 1/3 of the particles were contained in trees by $z = 0$. In the tree code, an opening angle of $\theta = 0.5$ was used.

Figure 4 shows projected particle positions at the final redshift, $z = 0$, for a portion of the volume around the largest halo that had formed. One important difference between the two codes can be seen by examining this figure. It is clear that the largest structures are quite similar in both cases; however, note that a number of small halos can be identified in the $P^3M$ snapshot that are not present in TPM. To verify this visual appearance in a more quantitative manner, bound halos were identified with DENMAX (Gelb & Bertschinger 1994). The resulting mass functions for the two codes are shown in Figure 5. The agreement is good for trees with more than 100 particles, but the TPM model has fewer small halos with less than 100 particles, confirming the visual impression.

The cause of this difference arises from the choice of $\rho_{th}$. Those objects that collapse early, which through merger and accretion will end up having higher masses, are identified when only slightly overdense and thus are followed at higher resolution throughout their formation. As $\rho_{th}$ rises, a halo must reach a higher overdensity before being followed with the tree code, so objects that collapse at late times are simulated at lower resolution. In this test case, the smallest tree at $z = 0$ contains 66 particles, so it is unsurprising that TPM has fewer halos near and below this size.
as the P³M, and in fact is higher for \( r_{12} < 10 \). This demonstrates clearly that the lower TPM correlation function in the former case is an effect of the higher force resolution of P³M in small halos and other regions where \( \rho < \rho_{\text{halo}} \). Within TPM halos followed as trees, the resolution is as good as (or better than) in P³M; the difference in \( \xi \) computed for halo particles only is most likely due to differences in softening (the tree code uses a spline kernel, while P³M uses a Plummer law) and in the time stepping.

The distribution of velocities is also sensitive to resolution effects. To examine this, particle pairs were divided into 30 logarithmically spaced bins, with bin centers between 50 kpc and 20 Mpc; for each pair, the line-of-sight velocity difference \( v_{12} \) was computed. Histograms showing the distribution of \( v_{12} \) in selected radial bins are shown in Figure 7. If only particles in the 100 largest halos are considered, the two codes are indistinguishable. Again, however, a difference becomes noticeable as more particles are included; the P³M halos begin to show more pairs with a small velocity difference (\( v_{12} < 250 \text{ km s}^{-1} \)). Since the P³M code is following smaller halos with higher resolution, these halos have smaller cores and a cooler velocity distribution than TPM halos with the same mass.

In order to compare the properties of individual collapsed objects, we selected a group of halos as follows. First, we chose those DENMAX halos without a more massive neighbor within 2 Mpc \( h^{-1} \). The spherically averaged density profile, \( \rho(r) \), was found for each halo, and a fit to the NFW profile (Navarro, Frenk, & White 1997) was computed by a \( \chi^2 \) minimization; those with less than 99.5% likelihood were excluded from further analysis. This fitting procedure repositioned the centers onto the densest region of the halo; we removed those halos for which the positions found in the two models differed by more than \( r_{200}/3 \), in order to be sure that the same halo is being examined in both cases. Figure 8 shows the \( \rho(r) \) for a few halos selected

![Figure 5](image1.png)

**Figure 5.** Mass function (shown as number of halos containing more than \( N \) particles) resulting from the TPM code (dashed line) and the P³M code (solid line). Details of the simulation are described in the text.

![Figure 6](image2.png)

**Figure 6.** Particle-particle correlation function for halo particles from the TPM (dotted line) and P³M (solid line) simulations. The bottom pair of lines shows the correlation function for all particles; the upper pair was calculated using only those particles in the 1000 most massive halos.

![Figure 7](image3.png)

**Figure 7.** Histograms of the line-of-sight velocity difference between pairs of particles with a given separation \( r_{12} \). Solid lines: Pairs from the 1000 most massive P³M halos. Dashed lines: Pairs from the 100 most massive P³M halos. Dotted lines: Corresponding values from the TPM simulation.
in this manner; the agreement is quite good, and within statistically expected fluctuations. If the TPM code had a lower resolution, then a broader halo profile with a lower density peak would result, but this is not seen.

Comparisons of other derived halo properties are shown in Figure 9. In each case we plot the fractional difference of the two models: $[\rho_{\text{TPM}} - \rho_{\text{P}^3\text{M}}]/0.5[\rho_{\text{TPM}} + \rho_{\text{P}^3\text{M}}]$. The top panel shows the number of particles within 1.5 Mpc $h^{-1}$ of the center, and the second panel shows the velocity dispersion. The agreement in both cases is good; the dispersion is 7% and 9% respectively, with no systematic offset or discernible trend with halo mass. The third panel of Figure 9 compares $r_{200}$ from the NFW fits, which also agrees quite well, the dispersion being 5%. At the low-mass end there are some TPM halos with sizes more than 20% larger, but these are also the ones with the smallest $r_{200}$. The final panel compares the core radius, $r_s$, resulting from the NFW profile fits, which shows the most variation between codes. There are a number of TPM halos with substantially larger cores (particularly at low mass), but the average TPM core size is smaller by 10% than that in $P^3M$. It appears that most TPM cores have in general been followed with the same as or higher resolution than that obtained with the $P^3M$ code, but a few have not. Examination of those halos with the largest differences often shows substructure or high ellipticity, but this is not always the case.

6. Summary

In the current environment, those wishing to carry out high-resolution simulations must tailor their approach to exploit parallel and distributed computing architectures. In this paper we have presented an algorithm for evolving cosmological structure formation that is well suited to such machines. By suitable domain decomposition, one large volume is broken up into a large number of smaller regions, each of which can be solved in isolation. This simplifies balancing the load between different processes, and makes it possible to use machines with high latency (e.g., a large number of physically distributed workstations) efficiently. Furthermore, it ensures that higher resolution in both space and time is applied in only those regions that require it.

An important parameter in the TPM code is the density threshold. By tying this parameter to the first and second moments of the density distribution, it is possible to follow initially small overdensities as they collapse and thus simulate halo evolution with as high a resolution as the more common $P^3M$ code. However, it is best to consider only those halos that contain twice as many particles as the smallest tree. Recently, Bagla (1999) introduced a different method of combining gridded and tree codes, called TreePM. This algorithm computes both a PM and a tree force for every particle, which has the advantage of uniform resolution for all particles. The performance of TPM in lower density regions can always be improved by lowering the density threshold, although this may lead to unacceptably large trees. Another possibility, which we intend to investigate, is to create a "TP$^3$M" code, which uses $P^3M$ rather than PM in the non-tree volume. This could be quite practicable, since the particle-particle interactions are not expensive to compute when the density is low.

However, it may be that increased force resolution in low-density regions is not a true improvement. Melott et al. (1997) and Splinter et al. (1998) showed that discreteness and two-body scattering effects become problematic when the force resolution outstrips the corresponding mass resolution. This led to a recent investigation by Knebe et al. (2000), who concluded that strong two-body scattering can lead to numerical effects, particularly when the local inter-particle separation is large or the time step is too long; slowly moving pairs of particles may suffer interactions that do not conserve energy. The TPM code will be less prone to such effects because low-density regions use lower force
resolution; only as the local mass resolution increases does the force resolution become higher, and simultaneously the time step will tend to become smaller.

This research was supported by NSF grants AST 93-18185 and AST 98-03137 (under subgrant 99-184), and the NCSA Grand Challenge Computation Cosmology Partnership under NSF cooperative agreement ACI-9619019, PACI subaward 766. Many thanks to to Ed Bertschinger for use of his P3M code, and to Lars Hernquist for supplying a copy of his tree code.

REFERENCES

Bagla, J. S. 1999, preprint (astro-ph/9911025)
Barnes, J., & Hut, P. 1986, Nature, 324, 446
Bode, P., Xu, G., & Cen, R. 1996, Supercomputing '96: Proc. 1996 ACM/IEEE Supercomputing Conference, Pittsburgh (New York: IEEE)
Couchman, H. M. P. 1991, ApJ, 368, L23
——. 1997, in ASP Conf. Ser. 123, Computational Astrophysics, ed. D. A. Clarke & M. J. West (San Francisco: ASP), 340
Davé, R., Dubinski, J., & Hernquist, L. 1997, NewA, 2, 277
Efstathiou, G., Davis, M., Frenk, C., & White, S. D. M. 1985, ApJS, 57, 241
Ferrell, R., & Bertschinger, E. 1994, Int. J. Mod. Phys. C, 5, 933
Frederic, J. J. 1997, Ph.D. thesis, MIT
Gelb, J. M., & Bertschinger, E. 1994, ApJ, 436, 467
Hernquist, L. 1987, ApJS, 64, 715
Hernquist, L., & Katz, N. 1989, ApJS, 70, 419
Hockney, R. W., & Eastwood, J. W. 1981, Computer Simulation Using Particles (New York: McGraw Hill)
Knebe, A., Kravtsov, A. V., Gottlöber, S., & Klypin, A. A. 2000, MNRAS, submitted (preprint astro-ph/9912257)
Kravtsov, A. V., Klypin, A. A., & Khokhlov, A. M. 1997, ApJS, 111, 73
Melott, A. L., Splinter, R. J., Shandarin, S. F., & Suto, Y. 1997, ApJ, 479, L79
Navarro, J. F., Frenk, C. S., & White, S. D. M. 1997, ApJ, 490, 493
Norman, M. L., & Bryan, G. L. 1999, in Numerical Astrophysics, ed. S. Miyama, K. Tomisaka, & T. Hanawa (Dordrecht: Kluwer), 19
Ostriker, J. P., & Steinhardt, P. J. 1995, Nature, 377, 600
Pearce, F. R., & Couchman, H. M. P. 1997, NewA, 2, 411
Splinter, R. J., Melott, A. L., Shandarin, S. F., & Suto, Y. 1998, ApJ, 497, 38
Xu, G. 1995, ApJS, 98, 355