CMU-CS-83-118

An Efficient Program for Many-Body Simulations (or, Cray Performance from a VAX)

Andrew W. Appel

March 1983

Abstract

The simulation of N particles interacting in a gravitational force field is useful in astrophysics, but such simulations become costly for large N. Representing the universe as a tree structure with the particles at the leaves and internal nodes labelled with the centers of mass of their descendants allows several simultaneous attacks on the computation time required by the problem. These approaches range from algorithmic changes (replacing an $O(N^2)$ algorithm with an $O(N \log N)$ algorithm) to data structure modifications, code-tuning, and hardware modifications. The changes reduced the running time of a large problem ($N = 10{,}000$) by a factor of four hundred. This paper describes both the particular program and the methodology underlying such speedups.

This research was supported by an NSF Graduate Student Fellowship and by ONR Grant #N00014-76-C-0370.

1. Introduction

Isaac Newton calculated the behavior of two particles interacting through the force of gravity, but he was unable to solve the equations for three particles. In this he was not alone [6, p. 634], and systems of three or more particles can be solved only numerically. Iterative methods are usually used, computing at each discrete time interval the force on each particle, and then computing the new velocities and positions for each particle.

A naive implementation of an iterative many-body simulator is computationally very expensive for large numbers of particles, where "expensive" means days of Cray-1 time or a year of VAX time. This paper describes the development of an efficient program in which several aspects of the computation were made faster. The initial step was the use of a new algorithm with lower asymptotic time complexity; the use of a better algorithm is often the way to achieve the greatest gains in speed [2].

Since every particle attracts each of the others by the force of gravity, there are $O(N^2)$ interactions to compute for every iteration. Furthermore, for the same reasons that the closed form integral diverges for small distances (since the force is proportional to the inverse square of the distance between two bodies), the discrete time interval must be made extremely small in the case that two particles pass very close to each other. These are the two problems on which the algorithmic attack concentrated. By the use of an appropriate data structure, each iteration can be done in $O(N \log N)$ time, and the time intervals may be made much larger, thus reducing the number of iterations required. The algorithm is applicable to N-body problems in any force field with no dipole moments.

Using an algorithm with a better asymptotic time complexity yielded a significant improvement in running time. Four additional attacks on the problem were also undertaken, each of which yielded at least a factor of two improvement in speed. These attacks ranged from insights into the physics down to hand-coding a routine in assembly language. By finding savings at many design levels, the execution time of a large simulation was reduced from (an estimated) 8000 hours to 20 (actual) hours. The program was used to investigate open problems in cosmology, giving evidence to support a model of the universe with random initial mass distribution and high mass density.

This paper describes the problem and its solution, considered from the point of view of a computer scientist approaching a software engineering problem. Thus, only a brief overview of the physics is given; the emphasis is on techniques of writing efficient software. Section 2 explains the nature of the cosmological questions that can be answered by many-body simulations. Section 3 describes some old algorithms for such simulations, Section 4 introduces the data structure and the algorithm to reduce the time per iteration, and Section 5 shows how to use the data structure to reduce the number of iterations. Section 6 shows how to create the structure and how to keep it from becoming distorted. Section 7 describes an implementation of

the algorithm. The techniques used to attain speedups at various design levels are described. These speedups are summarized, and the design methodology leading to them is discussed, in Section 8.

2. Applications in Astrophysics

The search for a faster algorithm to compute many-body interactions in a gravitational force field was motivated by two important questions in cosmology that can be investigated by simulating gravitational interactions of tens of thousands of galaxies. An efficient computer program has made it feasible to do such simulations. This section describes the cosmological applications, and the remaining sections describe the program.

2.1. How Did Galaxies Form?

It is generally believed that the early universe was radiation-dominated, that is, that most of the energy of the universe was in the form of photons, and the forces on a typical particle were primarily electromagnetic. The present universe, however, is mass-dominated, with most of the energy condensed into massive bodies (such as stars), and the primary interaction between these bodies being gravitational (the gravitational force between the Earth and the Sun, for example, completely dominates the "solar wind" of photons pushing the Earth away from the Sun).

The transition between a radiation-dominated and a mass-dominated universe probably took place relatively suddenly; after that, massive bodies such as galaxies began to form (they would have been torn apart in a radiation-dominated universe). Two of the competing theories describing the formation of galaxies [20] may be characterized as "top-down" and "bottom-up," respectively.

In the "top-down" theory [21], galaxy clusters formed as a result of long-range pressure waves left over from the radiation-dominated universe. A pressure wave contains alternating regions of high and low density. When the universe "condensed" and the radiation disappeared, there would be no medium to support the waves, but the regions of high and low mass-density would remain. It is proposed that the regions of high density became super-clusters of galaxies; that galaxies formed within these super-clusters; and that stars formed within the galaxies. Two-dimensional simulations under these assumptions have shown a cell-like structuring of the clusters [7]; it is not clear whether the dimensionality of the simulation is responsible. It may be that these cells exist in the present universe [12], but the observations at large distances are not conclusive.

In the "bottom-up" theory [16], there were no pressure waves, and the universe immediately after condensation consisted of randomly distributed hydrogen molecules. In a random distribution, there will be

local fluctuations in mass density, and as the universe expands, the denser regions will tend to cohere, while the regions of lower density will expand. This will tend to increase the size of the fluctuations, forming stars. More expansion will increase the size of the fluctuations to that of galaxies and eventually of clusters and super-clusters of galaxies. The clusters will have a more random structure than in the "top-down" model.

In both theories, the only significant interactions between galaxies after the condensation are gravitational. A simulation of the motion of many particles with gravitational interactions can therefore test these theories. A ten-thousand-galaxy, three-dimensional simulation testing the "bottom-up" theory (that is, starting with a uniform random distribution of particle positions) has been done using the techniques described in the remainder of this paper. The result of the simulation is clustering consistent with that observed by astronomers (see Figure 2-1 for a picture of the simulation's output). A similar test of the "top-down" theory has not yet been done, but since this theory differs from the "bottom-up" theory primarily in its specification of the distribution of the initial placement of the particles, it could be simulated easily using the same algorithm.

The large-scale simulations done using the program described in Sections 3 through 6 of this paper seem to imply that the bottom-up model can explain the present mass distribution of the universe quite well, without the complicated assumptions inherent in the top-down model.

2.2. Is the Universe Open or Closed?

One of the fundamental questions in cosmology is whether the universe will continue expanding forever, or whether it will eventually collapse in a gigantic reversal of the Big Bang. One way to answer this question is to look at the mass density of the universe. If the universe is below a certain "critical density" then expansion will continue forever; otherwise it will contract. Unfortunately, it is difficult to measure the mass density of the universe. Astrophysicists have been able to make estimates; most observational estimates put the mass density at about a tenth of the critical density. Since Truth, Beauty, and Simplicity demand that the density of the universe be equal to the critical density [15], a great astronomical search has been on for the "missing mass." The search is complicated by the fact that many forms of mass (such as black holes) are difficult to observe directly.

This problem can be avoided by approaches that do not involve direct observation of the mass density. One such approach is through simulation of the gravitational interactions of galaxies under different assumptions about the mass density. Groth *et al.* [10] observed in small simulations that low mass densities will not lead to the amount of clustering actually observed, and that the critical density would lead to such clustering. The ten-thousand-body, three-dimensional simulation using the program described later in this

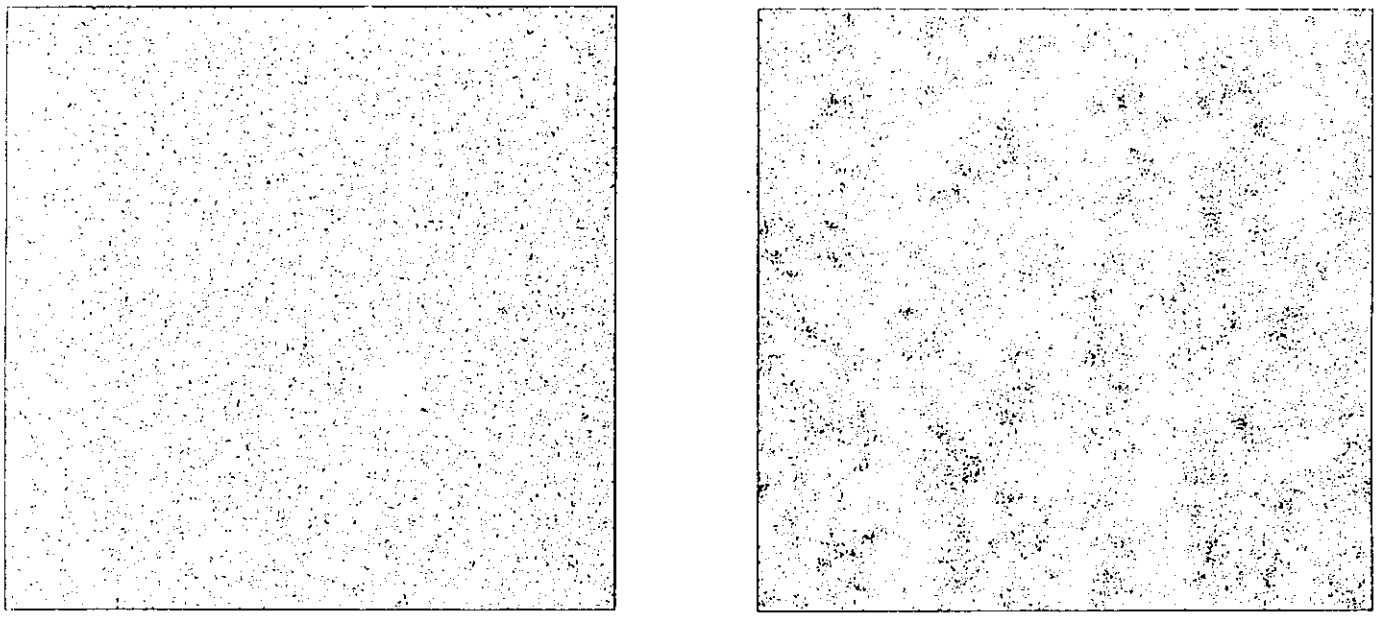

Figure 2-1: Result of a Simulation

An initial randomly generated configuration of 10,000 galaxies, and the result of simulating the gravitational interactions of this configuration as the universe expands by a factor of 7.12, with mass density $\rho = \rho_{crit}$ as a parameter of the simulation.

The particles are in a three-dimensional space which has been projected into two dimensions for this picture. A periodic coordinate system is used in which the two extreme points in each dimension are identified. The pictures are scaled to the expansion factor of the simulated universe.

paper was for the higher-density case; large-scale clustering was observed, lending support to this theory. The lower-density case can be examined by the same techniques.

3. Previous Algorithms

Because the N-body problem cannot be done in closed form, the calculation must be done numerically. That is, at each time t, the gravitational forces of each mass on each of the others may be computed by Newton's laws. (For an appropriate range of distances -- say, between one and a few hundred million light-years -- Newton's laws are a good approximation to General Relativity.) Using the inverse-square force law, an approximation to the true acceleration and velocity of each particle over a time dt can be computed. By many iterations of this method, the position of each particle after an arbitrary length of time may be found.

3.1. A Simple Algorithm

Newton's law of gravity states that the force between any pair of particles is proportional to the product of their masses divided by the square of the distance between them. Stated as a vector equation,

$$m_i \mathbf{r}''_i = \frac{G m_i m_j (\mathbf{r}_j - \mathbf{r}_i)}{|\mathbf{r}_j - \mathbf{r}_i|^3},$$

where $\mathbf{r}_i$ is the position vector of particle i, $\mathbf{r}''_i$ is the acceleration vector of particle i, and G is the universal gravitational constant.

When there are many particles, the acceleration of each particle is given by the sum of the accelerations (as computed by Newton's law) for all the other particles. This is simply a large set of differential equations. For two bodies, it is solvable in closed form; however, for more than two bodies no closed form solution exists.

The differential equation can be integrated numerically using a "naive" algorithm. At each iteration, compute the acceleration acting upon each particle; from this, compute a modified velocity over the next time increment, and then compute the position of each particle at the end of the time increment by calculating

$$\mathbf{r}_{\text{new}} = \mathbf{r}_{\text{old}} + \mathbf{v} \cdot dt,$$

The time increment dt must be made small enough that the accelerations do not greatly change between t and $t + dt$.

There are two problems with this algorithm. The first is that the number of interactions is large as a function of the number of particles. In particular, the gravitational action of each particle on every other particle must be computed every iteration, requiring a total of $N^2 - N$ operations. When N is large (physicists would like to simulate tens of thousands of particles, although they are rarely able to do so), an $O(N^2)$ algorithm is extremely costly to execute.

The second problem in many-body simulations is that it usually happens that some pairs of particles in such a system will pass very close to each other. Nearby particles in a gravitational field usually move at high speed with respect to each other; the combination of high velocities and small distances necessitates an extremely small time increment between iterations.

One approach to these problems is to use an extremely fast computer. The Cray-1 computer is very fast at algorithms that have a "vectorizable" formulation: that is, problems which can be expressed in terms of element-by-element arithmetic operations on long arrays of numbers. The acceleration computation can be formulated in terms of such large vectors. If the vector instructions of the Cray-1 are used to advantage (either by hand-coding, or by using the Cray Fortran compiler with a good understanding of what sorts of

programs the compiler can generate efficient code for), the time required to calculate the acceleration between two bodies can be estimated at 100 clock cycles (40 of which are needed for a calculation of a periodic distance function peculiar to the many-galaxy problem [3]). The time for one clock cycle is 12 nanoseconds [18], and the number of pairs of bodies is $N^2/2$, so the time for one iteration can be estimated at $.6N^2$ microseconds. Using scalar instructions, or using vector instructions with inefficient pipeline behavior, would more than double the time taken per iteration.

Using a similar program to simulate ten thousand bodies over one thousand iterations requires approximately 8000 hours of VAX time (this was extrapolated from observations of 100-particle simulations). Table 3-1 gives the times required for various implementations of a straightforward simulator. Even on a fast vector processor like the Cray-1, this simulation takes several hours. The disadvantage to running the simulation on the Cray computer is that the Cray-1 is enormously expensive: at a cost of eight to ten million dollars it is about 40 times as expensive as a large minicomputer such as a VAX. A solution whereby the problem can be solved in tens of hours on the VAX would obviously be preferable to any of the points in the solution space described in the table below.

	VAX-11/780	Cray-1 (estimated)
Optimizing Compiler	8000	30
Hand-optimized	5000	16

Table 3-1: Running times, in hours, of an $O(N^2)$ program for 10,000 bodies over 1,000 iterations.

3.2. Other Algorithms in the Literature

Two approaches have been taken to reduce the computational cost of solving the N-body problem. One approach is to represent the problem in a position-velocity phase space, and transform the force field using a Fast Fourier Transform into a form where it can be applied in linear time [13, 14]. This takes $O(N \log N)$ time (dominated by the Fourier Transform) per iteration. However, the phase space must be discrete. This means that all positions must be multiples of some lattice size a, and that all velocities must be less than some maximum f. Thus, the (physically interesting) effects of tight clusters cannot be modelled.

Another approach is to keep track, for each particle, of the sets of "nearby" particles and "faraway" particles [1]. The "faraway" particles may be integrated with larger time-steps than the "nearby" particles. When the particles are uniformly distributed, this has an asymptotic complexity of $O(N^{1.5})$. Unfortunately, when clustering occurs, the number of "nearby" particles is in the same order of magnitude as the total number of particles, and the asymptotic complexity is again $O(N^2)$. The problem of small time-steps is attacked by using a special-case technique for close two-body interactions, but this technique cannot be

applied for tight clusters of three or more particles.

Another similar approach is to divide the universe into cells, computing the particle-particle interactions within the cell, and then the cell-cell interactions [11]. This also has complexity $O(N^{1.5})$ for a uniform distribution, and also degrades to a quadratic time-complexity when clustering occurs.

With none of these algorithms is the problem of the vanishingly small discrete time-step solved; in the discrete phase-space approach, the time steps cannot be made smaller and thus information is lost, while in the second and third approaches, the problem is essentially the same as with the "naive" algorithm.

4. Reducing the Complexity of Each Iteration

To compute the force of gravity on an apple exerted by the Earth, it suffices to treat the Earth as a point mass; it is not necessary to sum the forces exerted by each atom of the Earth. This is a consequence of the spherical symmetry of the Earth; Newton invented the integral calculus to prove this fact.

When an attracting body is not spherically symmetric, the result obtained by treating it is a point mass is no longer exact, but it is a good approximation. This approximation -- in which one attraction between a pair of point masses is calculated, rather than all the attractions between all their constituent particles -- is the key to reducing the asymptotic complexity of computing the accelerations from $O(N^2)$ to $O(N \log N)$.

4.1. The Monopole Approximation

A divide-and-conquer algorithm can solve the many-body problem in $O(N \log N)$ time per iteration, and requires significantly fewer iterations. This order time has not been proved, but a reasonable argument is given; furthermore, experience with an implementation of the algorithm has shown that it runs as quickly as expected.

The algorithm relies upon the following approximation: suppose there are two particles, m_1 and m_2, each no more than dr from their center of mass (see Figure 4-1). The gravitational attraction they exert upon an observer situated a distance r from the center of mass will be

$$\mathbf{g} = \frac{Gm_1(\mathbf{r}+\mathbf{dr}_1)}{|\mathbf{r}+\mathbf{dr}_1|^3} + \frac{Gm_2(\mathbf{r}+\mathbf{dr}_2)}{|\mathbf{r}+\mathbf{dr}_2|^3} = \frac{G(m_1+m_2)\mathbf{r}}{|\mathbf{r}|^3} + O(dr^2).$$

Because there is no term in dr in this equation, the approximation is good to first order.

Now consider the arrangement of masses shown in Figure 4-2, which we will suppose to be a subset of the particles in a many-body simulation. To compute the acceleration of each particle on every other, we may break the computation into three parts: those interactions of two particles which are in the left-hand clump,

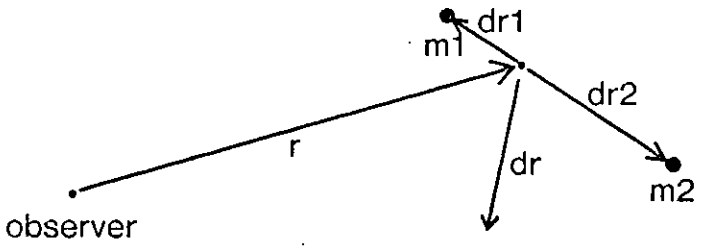

Figure 4-1: The Monopole Approximation

those interactions of which both particles are in the right-hand clump, and the interactions of a particle from each clump. The latter interactions may be approximated to order $(dr/r)^2$ by using the approximation described in the previous paragraph: by computing one interaction, as if each of the two clumps were one large mass. The number of computations required to calculate the inter-clump interaction has thus been reduced from $n_1 \cdot n_2$ to 1; the intra-clump calculation remains unchanged.

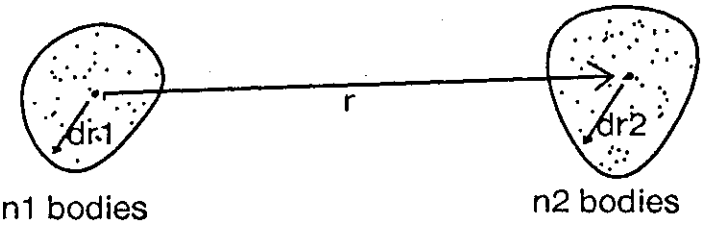

Figure 4-2: Two clumps to which the approximation can be applied

Had the two clumps been closer together, then the approximation would no longer have been as good, since it depends on the value of dr/r. In that case, more calculations would have had to be done.

4.2. A Data Structure

A method is needed for finding subsets of the particles for which the approximation can be made. This is made easier by the introduction of an appropriate data structure -- a binary tree whose leaves are particles and whose internal nodes represent clumps of particles. Every node will have an associated mass and position. The leaves will have the mass and position of the particles they represent; each internal node will have a mass equal to the sum of the masses of its two child nodes, and a position equal to the center of mass of its child nodes. Also associated with each clump (internal node) will be the approximate radius of the clump.

It is now a simple matter to compute all of the gravitational interactions between two clumps that are small

relative to their separation, that is,

$$dr_1/r < \delta \quad \text{and} \quad dr_2/r < \delta$$

for some fixed criterion of accuracy δ. The parameters dr_1 and dr_2 are stored in the tree; the positions need only be subtracted and multiplied by the total masses of each clump (also stored in the tree).

If the accuracy criterion is not satisfied, that is, if the clumps are large and close together, then the calculation of the interaction of each of the two subclumps of one clump with each of the two subclumps of the other clump must be made. It is not always necessary to "break up" both clumps for this calculation; see Figure 4-3 for an example in which one clump satisfies the criterion and need not be split, while the other clump is split into two pieces.

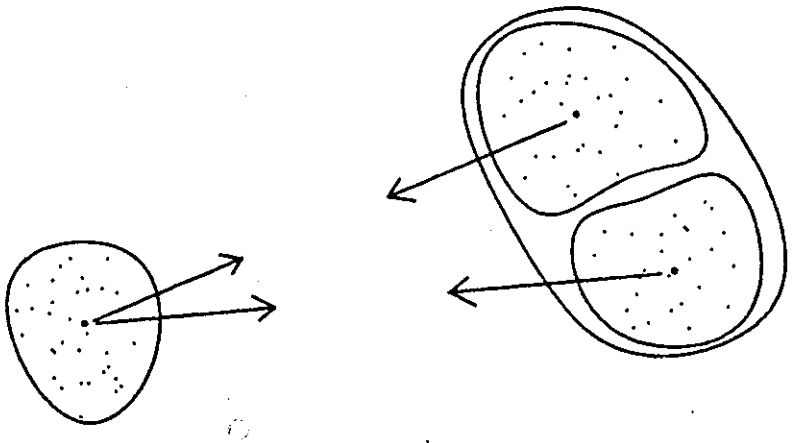

Figure 4-3: An example of the calculation of clump interaction

4.3. The Algorithm

This algorithm can be coded as the following pair of pseudo-Pascal recursive procedures -- procedure ComputeAccel computes all of the accelerations internal to one clump, and procedure TwoNode computes the interactions between two clumps.

```
procedure ComputeAccel ( B )
        begin if B is a nontrivial clump
                then begin ComputeAccel(B_left-child)
                           ComputeAccel(B_right-child)
                           TwoNode(B_left-child, B_right-child)
                     end
        end
```

```
procedure TwoNode(A,B)
        begin d ← r_B − r_A
            if (dr_A/d > δ) and (dr_A > dr_B)
              then begin TwoNode(A_left-child, B)
                         TwoNode(A_right-child, B)
                   end
              else if dr_B/d > δ
                     then begin TwoNode(A, B_left-child)
                                TwoNode(A, B_right-child)
                          end
                     else begin Acc_A ← Acc_A + Gm_B d/d^3
                                Acc_B ← Acc_B − Gm_A d/d^3
                          end
        end
```

One detail that for clarity has so far been omitted from the description of the algorithm pertains to the representation of position, velocity, and acceleration vectors. Rather than storing at each node the absolute position of the clump associated with that node, the position vector from the node's parent to the node is stored. (The same applies to velocities and accelerations.) This is done in order to minimize round-off errors in subtractions, which will be discussed in section 7. The absolute position of a particle or clump may be computed by taking the sum of the position offsets of all its ancestors up to the root, though it is rarely necessary to compute absolute positions. Note that the algorithm assigns accelerations throughout the data structure, taking advantage of the relativization of acceleration vectors.

4.4. Analysis of Time Complexity

If the parameter δ is set to zero, then the TwoNode procedure will always recur down to the level of individual particles, and the accelerations assigned to the internal nodes will be zero. If δ is not equal to zero, then the absolute acceleration of a single particle will be an approximation to the true acceleration. For values of δ between 0 and 1, the time complexity of ComputeAccel is estimated (and observed) to be $O(N \log N)$.

To see this, consider the number of times a particle X is compared with other clumps for the purposes of adding to an acceleration vector. Suppose there is a spherical shell around X of radius r and thickness $\delta \cdot r$. If this shell is filled with clumps of diameter $\delta \cdot r$, then there will be $4/\delta^2$ clumps in the shell. The smallest sphere will have a size such that the expected number of galaxies contained within it is 1; the largest will enclose a volume such that the expected number of galaxies within it is N. The quotient of the radii of the largest and smallest spheres will therefore be $N^{1/3}$. This will be equal to $(1+\delta)^k$, where k is the number of shells. Then

$k = \log(N)/3\log(1+\delta)$, and the number of clumps for which there must be calculation of accelerations with respect to particle X is approximately

$$\frac{4\log N}{3\delta^3\log(1+\delta)}.$$

Note that this number overestimates the number of calculations done, in that some of the calculation will involve not the comparison of X with another clump, but the comparison of an enclosing clump of X with another clump. That calculation would also be counted in this analysis as a calculation for X's sibling clump, and all other subclumps of the encompassing clump. However, this will do no more than change the constant of proportionality: for each of the N galaxies, $O(\log N)$ calculations must be done, giving a total execution time -- for fixed δ -- of $O(N\log N)$.

4.5. Accuracy of the Algorithm

The parameter δ is a measure of the accuracy of the calculation. When one clump is compared with another, and the ratio of diameter to separation is less than δ, then the computed acceleration will have a fractional error less than δ^2. When all the accelerations that clump X feels from other clumps are summed, the error in acceleration should be proportional to δ^2 divided by the square root of the number of clumps compared with (assuming random directions of the error vector). A more intuitive explanation of this statistical argument is that larger clumps will tend to approach some sort of spherically symmetric distribution, simply because of the large number of randomly positioned particles. In a perfectly spherical distribution, the error made in assuming that all the mass is positioned at the center is exactly zero. Thus the error in acceleration, on the average, should be significantly less than δ^2.

In fact, the distribution of errors, shown in Figure 4-4, is such that there is a maximum absolute error range, such that for most particles the error is quite small on an absolute scale. For particles with large accelerations, the proportional error is practically zero. Figure 4-4 was computed by taking a random distribution of particles and using the (exact) results computed by running the algorithm with $\delta = 0$ as the "Actual" acceleration components, and using the results computed with $\delta = 0.3$ as the "Computed" acceleration components. The absolute errors are the deviations from the line $y = x$, the scatterplot shows a good bound on the absolute error.

In those calculations where the exact final positions of the particles is not as important as statistics about their configurations, a relatively large value of δ can be used (such as $\frac{1}{2}$), greatly reducing the constant factor in the running time of the $O(N\log N)$ program.

It is useful to note that although the $O(N^2)$ algorithm has theoretically complete accuracy in computing

accelerations, the fact that the time intervals must be made discrete introduces approximations into any numerical calculation of the N-body problem. By choosing the parameters so that the errors introduced by each part (the clump approximation and the discrete-time approximation) are equal, the resulting error is about equal to that of the standard algorithm.

Since the use of a clumping algorithm to study the formation of galaxy clusters might conceivably be a cause of systematic error, the result of a simulation using this algorithm in which no clustering occurred is of interest. In this simulation, the galaxies were given higher initial velocities than predicted by theory, and no measurable clustering occurred (as seen both by the human eye and by a correlation function of interparticle distance).

5. Reducing the Number of Iterations

When two particles come very close to each other in an inverse-square force field, their accelerations become extremely high. To model their behavior accurately, extremely small time steps are required. In any simulation with a large number of particles, there are bound to be a few such pairs at any given time; these pairs require the time increments of the simulation to be so small that the number of iterations required to integrate over a significant interval of time becomes prohibitively large.

One widely used solution to this problem modifies the force law to limit the accelerations at small distances. The inherent problem with this approach in the modelling of galaxy clustering is that the clustering occurs (and should be examined by the simulation) over all distance scales. To tamper with the force law at small distances makes any conclusions about clustering at these distances suspect.

Fortunately, the data structure introduced in the previous section leads to a solution to this problem that preserves the inverse-square properties of the force law at all distance scales. In section 5.1 an aspect of the calculation open to algorithmic attack is described, and the attack itself is explained in sections 5.2 and 5.3.

5.1. Characteristic Times

The time increment dt between iterations is determined after each iteration. The usual approach is to use a global dt for all particles. In order to avoid gross inaccuracies at very small distances, the minimum characteristic time over all particles must be used for dt. The characteristic time of an object is a measure of how long it takes for that object's acceleration to change significantly; the time will be much shorter for a particle tightly orbiting a neighbor. The occasional tight pairs and threesomes require an expensively small value for dt in the naive algorithm.

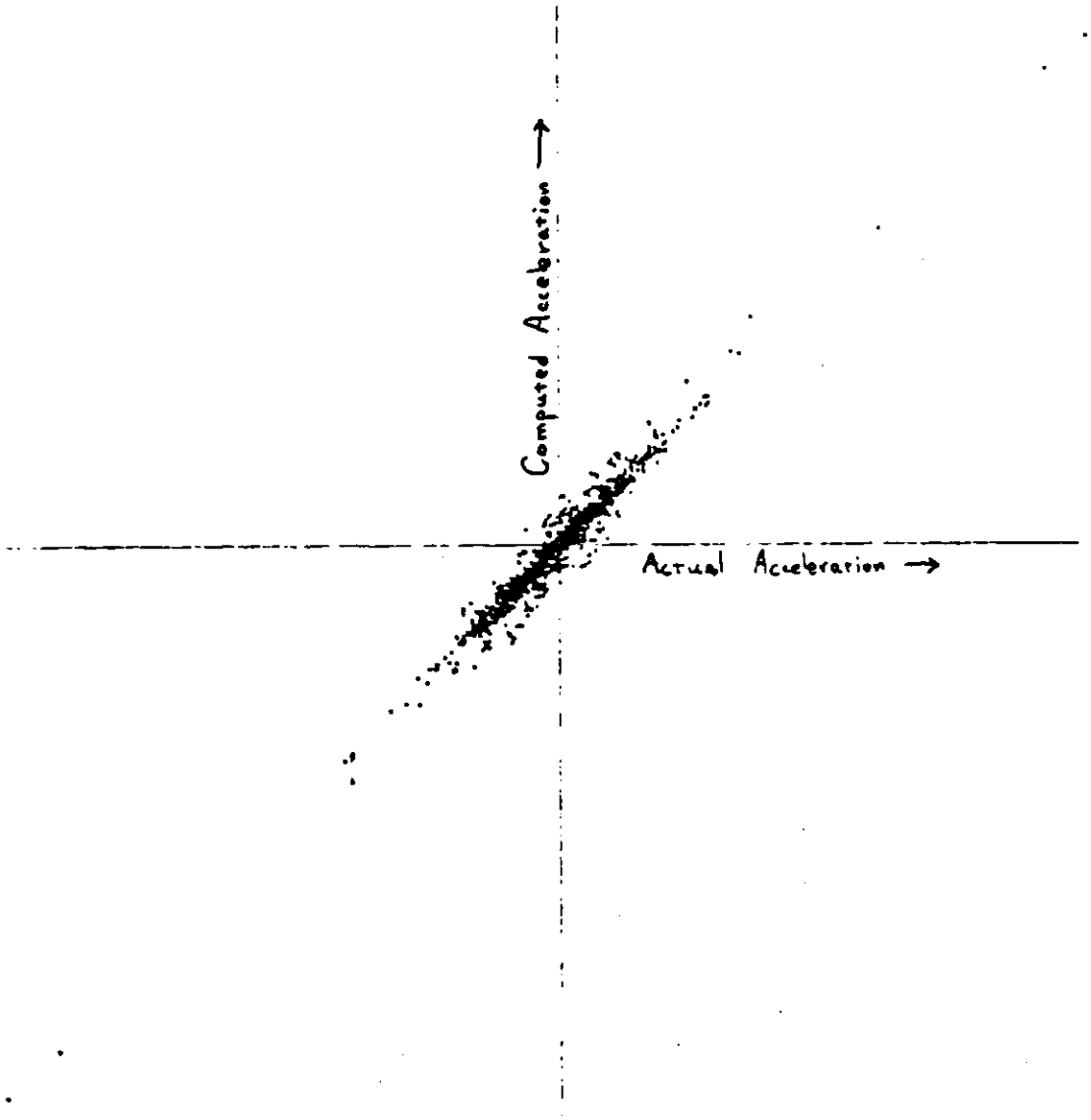

Figure 4-4: Scatterplot of components of
actual vs. computed accelerations for $\delta = 0.3$

The characteristic time for a clump C is the time in which a child of C will move a distance of approximately δ times the child's distance from C's center of mass. This is easy to calculate, since the position vector of each is stored as the vector from (the center of mass of) C. So the characteristic time of C is the min over both children of t_v and t_a, where

$$\delta \times |P| = t_v \times |V|$$

$$\delta \times |P| = |A| \times \tfrac{1}{2} t_a^2.$$

(Note that P, V, and A are the position, velocity, and acceleration vectors of the children relative to the center

of mass of *C*.) In each iteration, the accelerations are computed by ComputeAccel, the minimum characteristic time *dt* is found, and then Move calculates the new positions and velocities. Calculating the minimum characteristic time over the entire universe leads to an exceedingly small *dt*, however. Suppose two or three galaxies get into a tight orbit around each other; their characteristic time may be an order of magnitude shorter than the characteristic time of any other object in the universe.

It would be nice to be able to iterate small, very tight clusters at shorter time intervals than the rest of the universe, saving a large amount of calculation. This is not too difficult; what is needed is a concise criterion to distinguish such clumps.

5.2. Indivisible Clumps

Let such a clump be considered to be one object, indivisible, of nonzero radius. Indivisibility will be defined as follows: a clump is indivisible if for *all* clumps outside it, its ratio of size to distance is less than δ. What indivisibility effectively means is that an indivisible clump will never -- throughout the course of the acceleration calculations for one iteration -- be "split" by procedure TwoNode to calculate accelerations of its subclumps with respect to any other clump. This is easy to detect -- simply mark clump *A* in the first **then** clause or clump *B* in the second **then** clause of procedure TwoNode. Any clump that is never marked during the process of computing all the accelerations is indivisible.

The reason that this criterion is chosen is that it characterizes very well the set of clumps such that the external gravitational field acting upon them is an almost constant function of position within the clump. In fact, the monopole approximation has the effect of assuming that this field *is* constant, and the improved moving algorithm described below takes advantage of this fact.

Procedure Move, procedure ComputeAccel, and the procedure that determines *dt* will be altered so that they never look at the internal structure of such a clump. Note that TwoNode need not be altered, since the way indivisible clumps are defined implies that TwoNode never looks at their internal structure. Now the problem is gone: the small, tight cluster of galaxies has become a point (although with nonzero radius). The time increment *dt* will be much larger than it could have been otherwise.

The internal motions and accelerations of these tight clumps will have to be computed every iteration, and in fact it will take several local iterations of the tight clump to compute its motion over the time interval *dt*. However, these iterations of three or four objects are replacing iterations over the entire universe.

5.3. Closed Form Calculations

When an indivisible object itself is a clump containing two indivisible subclumps (these will usually be simply individual galaxies), then its orbit may be solved in closed form. In this case, the calculations to resolve internal motion may be postponed until another clump gets near enough to see the internal structure of the object. This may be many iterations of the universe later -- and many times more iterations of the tight pair, which typically has a much shorter characteristic time. Only one calculation needs to be made in closed form to replace these many iterations; furthermore, this calculation will be exceedingly accurate, since no approximations are being made internally to the subsystem.

Since indivisibility may occur at several distance scales (indivisible clumps may contain clumps which themselves contain indivisible clumps, and so on)), the tight-clump calculations (of which the two-body closed form calculation is a special case) may done recursively.

6. Managing the Data Structure

The efficiency of all parts of the algorithm depends on having the structure of the tree of clumps accurately reflect the structure of the particles in the simulated space. Under the influence of gravity, the particles move, distorting the tree. The structure must be maintained and the distortions removed regularly. Fortunately, this can be done in a simple way.

6.1. Reorganizing the Tree

After moving clumps that are not indivisible, the coordinates of a clump will no longer correspond exactly to the center of mass of the two subclumps. This is due to a nearby object attracting one subclump more strongly than the other. It is a simple matter, however, to adjust the position of each clump after its subclumps have been moved. Sometimes, however, another subclump will intrude into a clump so that the clumps no longer represent disjoint (in the simulated three-space) clusters. In this case, it is necessary that the clumps be rearranged (while keeping the actual galaxies fixed). The condition to aim for is this: for every clump C, the closest clump to C external to C shall be its parent clump. Let Closest(C) be the nearest clump with which C is compared during the execution of procedure TwoNode. If the distance from C to Closest(C) is less than the distance from C to its parent, then a new clump W will be formed, which will become the subclump of Parent(C) in place of C. W will contain as subclumps C and Closest(C). Now the old parent clump of Closest(C) has only one subclump, so it can be liquidated, "promoting" its subclump. This process is represented in Figure 6-1.

These adjustments (which shall be known as *Grabs*) take place immediately after procedure ComputeAccel finishes running. Each Grab is a purely local phenomenon in the data structure (only affecting four nodes),

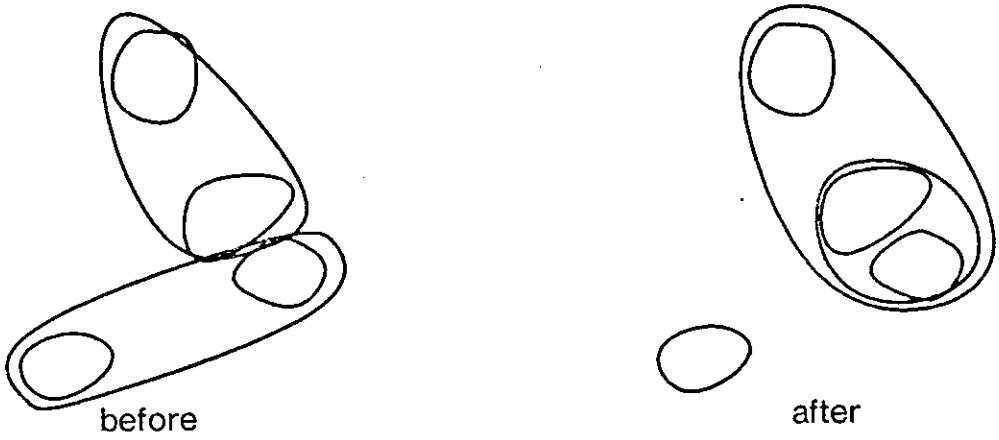

Figure 6-1: Rearrangement of Clumps

and preserves the positions, velocities, accelerations, and all other important data of the clumps involved. The process of Grabbing guarantees that close pairs will be subclumps of the same clump, and that the clumps will be close to optimally arranged for quickly computing accelerations. Although the Grab algorithm does not find the "best" arrangement of clumps, it has been observed to do a fairly good job in a very short time (see Figure 6-2).

The TwoNode procedure that calculates the accelerations throughout the tree also stores information which is used by the rearrangement algorithm in finding candidates for Grabs. The rearrangement is done after every iteration; it takes time linear in the number of particles.

6.2. Creating the Tree

While grabbing is very useful in maintaining the clump structure in the face of distortions, it will not be able to create one in the first place from a randomly arranged set of galaxies. This will be done as follows. The universal clump -- which contains all the galaxies -- will be divided initially into two subclumps chosen so that the first contains all galaxies whose x coordinate is less than the median x coordinate, and the other subclump will contain all galaxies with x larger than or equal to the median x.

Each of those subclumps will be divided into two sub-subclumps using the median y as the splitting criterion. Each lower level of clump will be split on z, then x, then y, then z, until the clumps consist of one galaxy each. Note that this procedure does not require that the number of clumps be a power of two, although that might seem most natural.

This structure is known as a k-d tree [4]. It has a variety of applications in multidimensional problems, including searching, nearest-neighbor calculations, classification, numerical integration, and computing

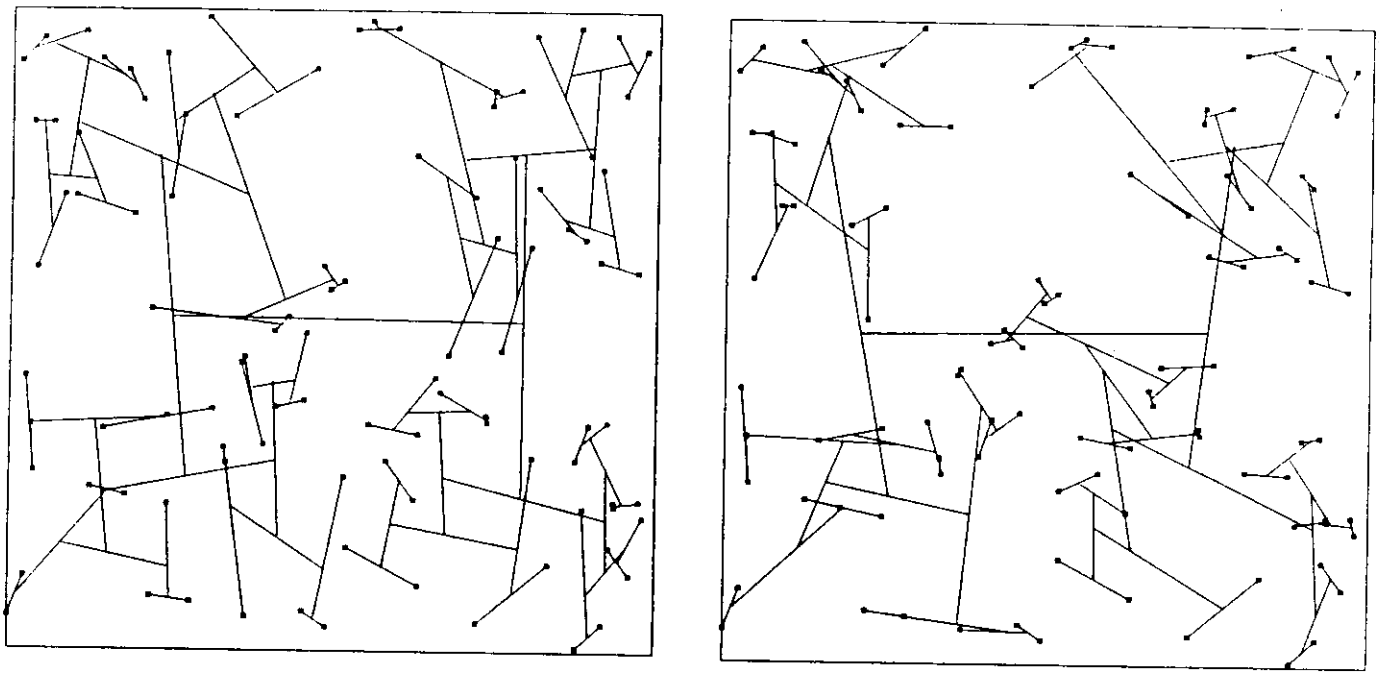

Figure 6-2: Effect of the Grab Algorithm

Figure 6-2 illustrates the effect of the grab algorithm on a two-dimensional universe. The diagram on the left depicts the clump structure as first created, by alternately splitting at the median x and y. The diagram on the right shows the structure after several iterations of Grab. Note that the particles are in the same positions, but the structure is cleaner -- close pairs are now all linked directly together. This improved structure may be measured by the fact that the acceleration calculation on the improved structure is empirically observed to be about twice as efficient as on the original structure.

minimum spanning trees [8, 5, 9]. For a many-body application, a standard k-d tree will be far from optimal -- nearby objects will not be in the same clump much of the time. The Grab procedure, though its behavior is difficult to analyze theoretically, has been observed to do a very good job of cleaning up the structure in just two or three iterations (see Figure 6-2).

7. Implementation of the Program

Various algorithmic attacks using the center-of-mass tree structure have been described in the preceding sections. It is inappropriate to stop seeking reductions in running time after a good algorithm has been found, however; significant efficiencies can be achieved in the implementation of a given algorithm.

The algorithm as described was first implemented in about 1200 lines of Pascal on a VAX-11/780. For a problem size of 10,000 galaxies, this first implementation runs in about forty minutes per iteration, and about 500 iterations are required to simulate the expansion of the universe by a factor of 100. Under the General Relativistic assumptions made, letting t run from 1 to 1000 causes the distance scales to run from 1 to 100,

because distance is proportional to $t^{2/3}$. Accelerations are transformed at each iteration to correspond with the changing distance scale. [3] Thus, 340 hours of execution time would be needed for this program, as opposed to 8000 for the $O(N^2)$ algorithm. The times given throughout this paper are for a slight modification of the algorithm to simulate a periodic distance function, which was necessary in the initial application. This adds a small constant factor to all distance calculations (eighteen floating point instructions, or about 25% of the running time).

A profiler was used to identify those parts of the program that consumed most of the processing time [19]. The profiler operates by asynchronously sampling the computer's program counter 60 times per second and incrementing the appropriate bin of a distribution function. The results showed that all but two percent of the execution time was spent in the TwoNode procedure. This is not unexpected, as TwoNode is the only part of the algorithm with an order time of $O(N \log N)$; the rest of the procedures run in $O(N)$ time. Since TwoNode is relatively small, hand optimization of the machine code was an obvious step. Writing this procedure in assembly language resulted in a speedup by a factor of two and a half. This rewriting used standard techniques, such as keeping more quantities in registers, putting procedure calls in-line, and using the addressing modes of the VAX more effectively.

At this point we found that the use of the Floating Point Accelerator option on the VAX significantly improves the performance of the program. The program was sped up by a factor of two by moving the calculations to a VAX on which an Accelerator had been installed.

Many-body calculations usually require double-precision arithmetic because of the wide range of distances involved. Close orbits are often more than four orders of magnitude -- a dozen binary digits -- closer than the distance to a far-away galaxy. Since the improved algorithm stores all positions relative to the parent clump, this problem disappears -- typically only one order of magnitude, or less, is involved in the difference between the size of a clump and the size of its parent clump. The use of 32-bit floating point numbers in place of 64-bit floating point halved the running time of the algorithm.

The factors of two in speed from the use of the Floating Point Accelerator and from the use of single precision are approximate and interdependent. Table 7-1 shows the running time as a function of these variables.

Since tight, "indivisible" clumps are recognized and their small time constant does not affect the time constant of the universe, far fewer iterations are required. Typically, such clumps are iterated about four times for each iteration of the universal clump. Each of those iterations would have been a global iteration in the straightforward algorithm, as in that algorithm there is no way to detect a tight clump. A conservative

	32-bit Floating Point	64-bit Floating Point
With FPA Hardware	16	28
Without FPA Hardware	25	74

Table 7-1: Running times, in seconds, of an acceleration calculation for 1,000 bodies on a VAX-11/780.

estimate of the number of iterations saved is 50% -- a factor of two speedup.

In the astrophysical applications described in Section 2, in which galaxy clustering occurs, the development of clusters among the particles simulated leads to greater opportunities for procedure TwoNode to apply its approximation. The resulting gain is an empirically observed two-fold speedup in computation of accelerations.

The program that resulted from these modifications to a very simple iteration method succeeded in reducing the running time of a simulation from 4000 hours to 20 hours -- a factor of two hundred (for ten thousand bodies, with $\delta^2=0.3$). This saving was achieved by attacking the problem from several angles at once.

8. Conclusions

It is often very difficult to make one change in a program that makes it faster by more than one order of magnitude. In this case, even a change that reduced the order time of the algorithm from $O(N^2)$ to $O(N \log N)$ increased the efficiency for a typical problem size by only a factor of 12 -- one order of magnitude. The four-hundred-fold reduction in running time was the product of savings at all levels of the conceptual hierarchy, from the idea that some galaxies are in systems by themselves, to the idea that keeping certain pointers in registers saves memory references (see Table 8-1). They are in some sense independent -- improving the efficiency of one level of the hierarchy does not preclude improving the efficiency of another. Most importantly, all of the savings are multiplied together.

Reddy and Newell [17] have characterized the type of problem for which this multiplicative speedup can be expected: such a problem has four to eight layers of implementation, such as computer technology, architecture, algorithm, *et cetera.* This paper has been concerned with ways to avoid changes in the technology and architecture layers (i.e., using a Cray-1) because of their expense. Rather, the algorithms, "knowledge sources," and implementation layers have been attacked.

This brings the running time of the algorithm on a relatively small and inexpensive computer such as the VAX down to what it would be on a large, extremely fast, and expensive Cray-1. Of this speedup, a factor of

Level	Speedup Factor	Description
Algorithm	12	Changing to the $O(N \log N)$ algorithm
Problem-Specific Knowledge	2	Iterating indivisible clumps by themselves and using closed-form solutions, thus halving the number of global iterations.
Algorithm (Problem-Specific)	2	Clustering behavior in the simulation produces a clump structure well-suited to the algorithm
System-Independent Code Tuning	2	Use of single precision floating point rather than double precision, made possible by the data structure
System-Dependent	2.5	Hand-coding the routine where most of the time was spent
Hardware	2	Use of the Floating-Point Accelerator

Table 8-1: Summary of the speedups attained at various levels

about two was attributable to technology (the use of the Floating Point Accelerator) and two to implementation (hand coding a critical routine) -- these could be done for any program, probably with similar results. The other factor of a hundred (for ten thousand bodies) came from the exploitation of the data structure in various ways. The use of a good data structure to provide an asymptotically fast algorithm is especially important for large problems.

Since the layers of the problem are relatively independent, the technology and architecture layers are still available for additional speedup factors. If the program were run on a Cray-1 or a Cyber 7600, the 20 hours of runtime might be reduced to 1 or 2 hours, since most of the efficiency improvements described in this paper are machine-independent, and these computers are much faster than the VAX (and almost proportionally more expensive).

The data structure is a variant of one already known in the literature (the k-d tree), but the reorganization of the tree with the Grab procedure changes it substantially -- it loses the useful (for some applications) property of being split along planes of constant x,y, and z, and gains the useful (for this application) property of joining mutually nearest neighbors at all levels of the hierarchy. For the simulation of gravitational attractions, this turned out to better than halve the number of calculations. Reorganized trees may have other applications as well; for example, the recognition of individual objects from the point set obtained from a television camera might be facilitated by an algorithm that could group points together in $O(N \log N)$ time. Some sorts of nearest-neighbor searching might also be made easier.

It is difficult to analyze the properties of the Grab algorithm. It is low-level in nature: when two points are found to be closer to each other than to their parent nodes, a local rearrangement is done without regard for the global structure of the tree. That it works as well as it does was difficult to predict. Its behavior is dependent on δ, since these closest pairs are detected during the TwoNode procedure; the question of what δ to use to most efficiently produce a reorganized tree (independent of gravitational considerations) might be investigated if reorganized trees are found to be useful in other applications.

References

[1]
Sverre J. Aarseth, J. Richard Gott III, and Edwin L. Turner, *N-body Simulations of Galaxy Clustering: I. Initial Conditions and Galaxy Collapse Times,* Astrophysical Journal, 228 (1979), pp. 664-683.

[2]
Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

[3]
Andrew W. Appel, *An Investigation of Galaxy Clustering Using an Asymptotically Fast N-body Algorithm,* Undergraduate Thesis , Princeton University, Princeton, NJ, April 1981.

[4]
Jon Louis Bentley, *Multidimensional binary search trees used for associative searching,* Comm. ACM, 18 (1975), pp. 509-517.

[5]
Jon Louis Bentley and Jerome H. Friedman, *Fast Algorithms for Constructing Minimum Spanning Trees in Coordinate Spaces,* IEEE Transactions on Computers, C-27 (1978), pp. 97ff.

[6]
Edward A. Desloge, *Classical Mechanics,* John Wiley & Sons, New York, NY, 1982.

[7]
A. G. Doroshkevich, E. V. Kotok, I. D. Novikov, A. N. Polyudov, Yu. G. Sigov, and S. F. Shandarin, *Dvumernaya model obrazovaniya krupnomasshtabnoi struktury vselennoi (A Two-Dimensional Model of the Formation of Large-Scale Structures of the Universe),* Preprint 83, IPM AN SSSR (Institute for Problems of Mechanics, Academy of Science, USSR), Moscow, USSR, 1978.

[8]
Jerome H. Friedman, Jon Louis Bentley, and Raphael Ari Finkel, *An Algorithm for Finding Best Matches in Logarithmic Expected Time,* ACM Transactions on Mathematical Software, 3 (1977), pp. 209ff.

[9]
Jerome H. Friedman and Margaret H. Wright, *A Nested Partitioning Procedure for Numerical Multiple Integration,* ACM Transactions on Mathematical Software, 7 (1981), pp. 76ff.

[10]
Edward J. Groth, P. James E. Peebles, Michael Seldner, and Raymond M. Soneira, *The Clustering of Galaxies,* Scientific American, 237 (1977), pp. 76 ff.

[11]

Roger W. Hockney and James W. Eastwood, *Computer Simulation Using Particles,* McGraw-Hill, New York, NY, 1981.

[12]

M. Joeveer, J. Einasto, and E. Tago, *Yacheiskaya Struktura Vselennoi (The Cell Structure of the Universe),* Preprint A-1, AN Estonskoi SSR, Tartu, Estonian SSR, USSR, 1977.

[13]

R. H. Miller and K. H. Prendergast, *Stellar Dynamics in a Discrete Phase Space,* Astrophysical Journal, 151 (1968), pp. 699ff.

[14]

R. H. Miller, K. H. Prendergast, and William J. Quirk, *Numerical Experiments on Spiral Structure,* Astrophysical Journal, 161 (1970), pp. 903-916.

[15]

Charles W. Misner, Kip S. Thorne, and John Archibald Wheeler, *Gravitation,* W. H. Freeman & Co., San Francisco, CA, 1973.

[16]

P. J. E. Peebles, *The Large-Scale Structure of the Universe,* Princeton University Press, Princeton, NJ, 1980.

[17]

R. Reddy and Allen Newell, *Multiplicative Speedup of Systems,* Perspectives on Computer Science, A. K. Jones, ed., Academic Press, New York, NY, 1977, pp. 183-198.

[18]

Richard M. Russell, *The CRAY-1 Computer System,* Computer Structures: Principles and Examples, Siewiorek, Daniel P., Bell, C. Gordon, Newell, Allen, ed., McGraw-Hill, New York, NY, 1982, pp. 743-752.

[19]

Unix Programmer's Manual, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA. Sections **prof(1)**, **pc(1)**, and **profil(2)** contain information about the execution profiler.

[20]

M. Mitchell Waldrop, *The Large-Scale Structure of the Universe,* Science, 219 (1983), pp. 1050-1052.

[21]

Ya. B. Zeldovich, *The Theory of the Large Scale Structure of the Universe,* IAU Symposium #79, International Astronomical Union, Dordrecht, Holland, 1978, pp. 409 ff.