Chapter 8

N-Body Models of Collisionless Systems

Joshua Edward Barnes

Informally, a self-gravitating system is *collisionless* if the granularity of its mass distribution does not influence its evolution. Galaxies, in particular, are expected to evolve collisionlessly even over timescales much longer than a Hubble time. This chapter describes an N-body algorithm for simulating the evolution of collisionless systems.

8.1 Collisionless Stellar Systems

Consider a system composed of $\mathcal{N}$ stars bound together by their mutual gravitation. Two distinct time-scales arise in discussing the evolution of such a system (*e.g.* Binney & Tremaine 1987, § 4.1):

$$t_{\rm c} \simeq \sqrt{\frac{1}{G\rho_{\rm h}}}\,, \qquad \text{and} \qquad t_{\rm r} \simeq \frac{\mathcal{N}}{8\ln\mathcal{N}} t_{\rm c}\,, \tag{8.1}$$

where G is the gravitational constant and $\rho_{\rm h}$ is the average density within the contour containing half the mass. $t_{\rm c}$ is comparable to the orbital period of a typical star, while $t_{\rm r}$ is the time over which that orbit is significantly deflected by the cumulative effects of interactions with all other stars in the system. Over intervals much smaller than $t_{\rm r}$ one may be able to neglect the effects of interactions between individual stars, and if $t_{\rm c} << t_{\rm r}$ we expect the system to evolve collisionlessly.

8.1.1 The Collisionless Boltzmann Equation

When $\mathcal{N}$ is large it makes sense to adopt a statistical description instead of tracking individual stars. Let $f(\mathbf{r}, \mathbf{v}, t) d^3\mathbf{r} d^3\mathbf{v}$ be the total mass of the stars in the phase-space volume $d^3\mathbf{r} d^3\mathbf{v}$ centered on $(\mathbf{r}, \mathbf{v})$ at time t; assuming that correlations between stars can be neglected, this *distribution function* completely describes the dynamical state of the system. The evolution of $f(\mathbf{r}, \mathbf{v}, t)$ is governed by the collisionless Boltzmann equation,

$$\frac{\partial f}{\partial t} + \mathbf{v} \cdot \frac{\partial f}{\partial \mathbf{r}} - \nabla \phi \cdot \frac{\partial f}{\partial \mathbf{v}} = 0 \, , \tag{8.2}$$

where the gravitational potential $\phi(\mathbf{r}, t)$ is given by Poisson's equation,

$$\nabla^2 \phi = 4\pi G \int f d^3\mathbf{v} \, . \tag{8.3}$$

8.1.2 Solution by Monte-Carlo Method

Finite-difference methods of solving eq. (8.2) are impractical in three dimensions. Instead, one adopts a Monte-Carlo approach (*e.g.* White 1983): at some initial time t_0, construct an *N-body realization* of the system by chosing phase-space coordinates $(\mathbf{r}_i, \mathbf{v}_i)$ for $i = 1, \ldots, N$ with probability proportional to $f(\mathbf{r}_i, \mathbf{v}_i, t_0)$, and assigning each body a mass $m_i = N^{-1} \int f d^3\mathbf{r} d^3\mathbf{v}$. Thus the expectation value for the mass within any finite region of the realization is equal to the mass within the same region of the original system (though the *actual* value will be different due to $\sqrt{N}$ fluctuations). This N-body realization is evolved according to the Newtonian equations of motion,

$$\frac{d\mathbf{r}_i}{dt} = \mathbf{v}_i, \qquad \frac{d\mathbf{v}_i}{dt} = -\nabla \phi|_{\mathbf{r}=\mathbf{r}_i} \, . \tag{8.4}$$

One may think of this N-body representation as a "mechanical model" of the actual system. In the limit $N \to \infty$ the behavior of the model approaches the behavior described by eq. (8.2).

A discrete solution of eq. (8.3) is needed to apply this Monte-Carlo approach to a self-consistent system. One solution which appears entirely suitable from a mathematical point of view is

$$\phi(\mathbf{r} = \mathbf{r}_i) = -\sum_{j \neq i} G \frac{m_j}{(|\mathbf{r}_j - \mathbf{r}_i|^2 + \varepsilon^2)^{1/2}} \, . \tag{8.5}$$

Here ε is a "softening parameter" with dimensions of length, originally introduced to simplify the treatment of close encounters between individual bodies. Softening also has the welcome effect of smoothing the gravitational field on small scales; this helps reduce the effects of discreteness.

A number of methods for calculating the gravitational acceleration have been described in the literature (Sellwood 1987; Hockney & Eastwood 1981). Many of these methods, especially those tailored to systems with simple geometries, are computationally much faster than direct summation; however, this speed is often gained by sacrificing generality (in the case of expansion techniques) or spatial resolution (in the case of three-dimensional

Fourier methods). It is sometimes claimed that certain methods "suppress two-body relaxation", meaning that effects due to discreteness are less severe than they would be in a simulation with the same number of bodies and spatial resolution performed using eq. (8.5). Such claims are highly suspect. Regardless of the method used to compute the gravitational field, suppression of two-body relaxation for a given N *always* entails a cost in spatial resolution (*e.g.* Hernquist & Barnes 1990).

It is important to recognize the limitations of the Monte-Carlo approach; effects due to discreteness may become quite apparent over times of order $0.1t_{\mathrm{r}}$ or less. For example, van Albada (1986) found that the binding energies of individual bodies in an equilibrium model of a triaxial galaxy fluctuated by 10 or 20% over a few tens of dynamical times. This behavior was observed in a system of $N = 20,000$ bodies, for which eq. (8.1) gives $t_{\mathrm{r}} \simeq 250t_{\mathrm{c}}$. Simulations with $N \sim 10^3$ to 10^5 bodies can describe some aspects of violent relaxation, merging, and the development of gross dynamical instabilities, but they are probably too coarse to address questions of long-term evolution.

8.2 Hierarchical N-Body Methods

Relatively fast and completely general methods for evaluating the sum in eq. (8.5) are now available (Greengard 1990). These hierarchical "tree" codes are based on the idea of approximating the long-range force field of a localized region, containing many bodies, with some sort of multipole expansion. A tree-structure is used to partition the system into a hierarchy of such regions. The gravitational field at any point is approximated by making a partial scan of the tree to obtain a detailed description of the nearby mass distribution and an increasingly coarse-grained description of more distant parts of the system. If a better approximation is required, the algorithm may either examine a finer level of the hierarchy or include more moments of the mass distribution for each region.

The strategy adopted to refine the force approximation is closely related to the representation adopted for the gravitational field (Katzenelson 1989). "Action-at-a-distance" codes represent the field at a given point $\mathbf{r}$ by a list of masses which together generate an approximation to the potential and force at $\mathbf{r}$. This list is $O(\log N)$ elements long, so the time required to evaluate the force on all N bodies scales like $N \log N$ (Barnes & Hut 1986). "Field" codes represent the field at $\mathbf{r}$ by a multipole expansion carried out to some prespecified order. The cost of evaluating the field at a single point is then independent of N, so forces on all bodies can be computed in time proportional to N (Greengard & Rokhlyn 1987). Action-at-a-distance codes typically examine finer levels of the hierarchy to obtain more accurate forces, while field codes expand the field to a higher orders to accomplish the same end.

Despite the favorable asymptotic scaling of field methods, the relative advantages of these two approaches are not clear at present. From a practical standpoint, however, action-at-a-distance codes seem to be simpler to implement, and such codes are well-established in astrophysical research. Consequently, the code presented here uses an action-at-a-distance algorithm.

8.2.1 Tree Structure

The data structures needed to implement a tree code are widely used in computer science. Following Knuth (1973, § 2.3), a tree may be abstractly defined as a finite set T of nodes such that

1. There is a single node r which is the *root* of T, and
2. The other nodes, exclusive of r, are divided into $m \geq 0$ disjoint sets $T_1, \ldots, T_m$, each of which is also a tree; T_i is the i^{th} *subtree* of T.

This definition is recursive – as are many of the algorithms used to manipulate trees – but it is not circular, since complex trees are defined in terms of simpler ones. The simplest trees consist of just one node; such a node is sometimes called a terminal node. Conversely, if T contains more than one node then its root is an interior node. Since every node is the root of some tree, every node is either terminal or interior.

At a slightly less abstract level, a tree is often visualized as a graph, with vertexes representing nodes and edges representing the relationships between them. Specifically, an edge is drawn between nodes p and q if p is the root of some tree T and q is the root of one of T's subtrees. As a rule, trees are sketched "upside-down", with the root at the top and other nodes arranged below; thus to descend a tree is to move towards the terminal nodes. The set of all nodes which may be reached by descending from node p are p's *descendents*, while the nodes visited in tracing a path from p to the root of the entire tree are p's *ancestors*. Immediate descendents and ancestors are sometimes called offspring and parents, respectively.

Two different kinds of tree structure have been used in tree codes. In "lagrangian" codes (Appel 1981, 1985; Porter 1985; Jernigan 1985; Press 1986; Benz *et al* 1990) the tree is constructed bottom-up, for example by repeatedly replacing pairs of mutually nearest neighbors with pseudo-bodies. This generates a binary tree structure, identifying each body with a terminal node, and each pseudo-body with an interior node. In "eulerian" codes (Barnes 1986; Greengard & Rokhlyn 1987) the tree is constructed top-down by repeatedly subdividing a cubical cell enclosing the entire system, as shown in Fig. 8.1. In three dimensions, this generates an octonary tree structure, associating each cell with an interior node. A cell may be subdivided along the coordinate planes into eight smaller cells of equal volume; thus each cell has an ordered set of up to eight descending links, referred to as subcells. Fig. 8.2 presents a two dimensional analog of this construction, showing how such a tree is organized.

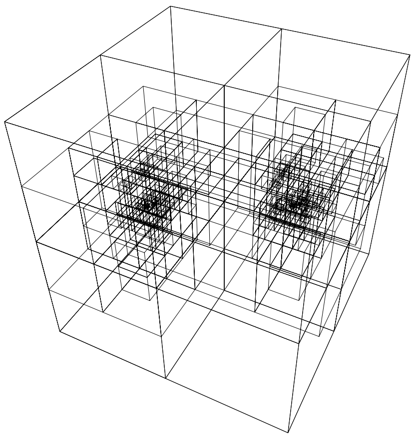

Fig. 8.1. Hierarchical box structure generated from a particle distribution with two density centers.

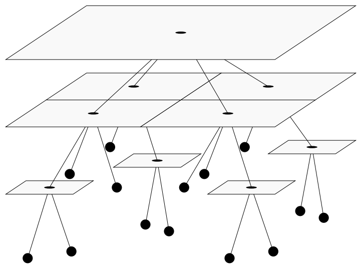

Fig. 8.2. Tree generated from a two-dimensional system. Filled circles are bodies; shaded squares are cells. The largest cell is the root.

One might suppose that the arbitrary divisions created with cubical cells would lead to inaccuracies in the force calculation. In general this is not the case (Makino 1990), although problems can arise in certain exceptional situations. On the other hand, eulerian tree codes are simple to characterize and easy to construct. The present code uses an eulerian tree, generated by subdividing the root cube until each body is isolated within a single cell. In practice, of course, it is redundant to allocate a cell to hold only one body, and such trivial cells are replaced with the bodies themselves. The number of non-trivial cells in the tree structure depends on the mass distribution, but is typically of order $0.5N$.

8.2.2 Force Calculation

Evaluating the force on a given body, p, is accomplished by a partial exploration of the tree, starting at the root. At each node q visited, a three-way decision is made. If q is a body, then the gravitational field due to q is added to the running total for p. If q is a cell, there are two further possibilities. If q is sufficiently well-separated from p, its gravitational field is again added to the running total. On the other hand, if q is too close to p to yield an accurate force, the descendents of q must be examined instead.

In the original version of the algorithm presented here, a cell q was deemed far enough away from p to be approximated as a single mass if $\ell_q/\theta < |\Delta\mathbf{r}|$, where ℓ_q is the linear size of q, $\Delta\mathbf{r} \equiv \mathbf{r}_q - \mathbf{r}_p$ is the vector extending from p to the center-of-mass of q, and θ is a user-specified parameter, used to control the accuracy (Barnes & Hut 1986). This criterion can give rise to large errors in certain circumstances when the center-of-mass of a given cell lies far from its geometric center (Salmon & Warren 1994). In the present code the criterion used is

$$\ell_q/\theta + \delta_q < |\Delta\mathbf{r}|\,, \tag{8.6}$$

where δ_q is the distance between q's center-of-mass and its geometric center. Since the critical radius $\ell_q/\theta + \delta_q$ depends only on the parameters of the cell q, it is conveniently evaluated during tree construction, and the result is stored in the data structure associated with each cell for quick reference during force calculation.

While similar to the original, eq. (8.6) has some advantages. For example, the original criterion cannot guarantee that q is not an ancestor of p unless $\theta < 1/\sqrt{3} \simeq 0.577$; noting this, Hernquist (1987) proposed testing to insure that $p \notin q$. Eq. (8.6) positively guarantees $p \notin q$ for any $\theta < 2/\sqrt{3} \simeq 1.155$; since this is larger than the θ values used in most simulations, Hernquist's test need not be explicitly included. This new criterion also effectively cures the "detonating galaxy" pathology described by Salmon & Warren at a somewhat smaller computational cost than the "Bmax" criterion they suggest.

The contribution of a given node q to the gravitational field at body p depends on the type of q. If q is a body, its contribution to the potential at p is simply

$$-\phi_q = G\frac{m_q}{(|\Delta\mathbf{r}|^2 + \varepsilon^2)^{1/2}}\,, \tag{8.7}$$

where m_q is the mass of q. On the other hand, if q is a cell, its contribution to the potential may also include a quadrupole correction, giving

$$-\phi_q = G\frac{m_q}{(|\Delta\mathbf{r}|^2 + \varepsilon^2)^{1/2}} + G\frac{\Delta\mathbf{r}\cdot\mathbf{Q}_q\cdot\Delta\mathbf{r}}{2(|\Delta\mathbf{r}|^2 + \varepsilon^2)^{5/2}}\,, \tag{8.8}$$

where $\mathbf{Q}_q$ is the traceless quadrupole moment of the cell q (note that it is not necessary to include a dipole correction since ϕ_q is calculated using the center-of-mass of the cell). In either case, q's contribution to the force on p is obtained by symbolically differentiating ϕ_q with respect to $\mathbf{r}_p$. As Hernquist (1987) has shown, inclusion of quadrupole moments can significantly increase the accuracy of force calculation.

8.2.3 Time Integration

In principle, integration of orbital trajectories is independent of the details of force calculation. Practical issues, however, require some consideration of the force calculation algorithm when chosing an integrator. Schemes in which different bodies have different – and generally variable – time-steps are needed for simulations of star clusters (*e.g.*, McMillan & Aarseth 1993; also see Chapter 10); they are also useful in simulations of astrophysical gas dynamics (*e.g.*, Hernquist & Katz 1989; Benz *et al* 1990; also see Chapter 5). But in collisionless N-body models the choice of time-step is tied to issues of global stability, and it is not entirely clear how to go about selecting individual time-steps. In the interests of simplicity, the present code therefore uses a time-centered leap-frog, advancing all bodies with the same time-step parameter Δt chosen by the user. This approach is simpler, more robust, and more economical of memory than available high-order schemes.

As normally formulated, the leap-frog requires that velocities be offset by half a time-step with respect to positions. However, the method can easily be recast as a mapping from time $t^{[n]}$ to time $t^{[n+1]} = t^{[n]} + \Delta t$ (*e.g.*, Barnes & Hut 1989). Let $\mathbf{r}^{[n]}$ and $\mathbf{v}^{[n]}$ be position and velocity of a body at time-step n; the body is advanced as follows:

$$
\begin{aligned}
\mathbf{v}^{[n+1/2]} &= \mathbf{v}^{[n]} + \frac{1}{2}\,\Delta t\,\mathbf{a}(\mathbf{r}^{[n]}) \\
\mathbf{r}^{[n+1]} &= \mathbf{r}^{[n]} + \Delta t\,\mathbf{v}^{[n+1/2]} \\
\mathbf{v}^{[n+1]} &= \mathbf{v}^{[n+1/2]} + \frac{1}{2}\,\Delta t\,\mathbf{a}(\mathbf{r}^{[n+1]})\,,
\end{aligned}
\tag{8.9}
$$

where $\mathbf{a}(\mathbf{r})$, the acceleration obtained from the force calculation, depends on the positions of all bodies. This reformulation does not compromise the desirable properties of the leap-frog integrator.

8.3 The TREE Code

Apart from the routines associated with tree construction (§ 8.3.2) and force calculation (§ 8.3.3-4), the TREE code follows a fairly conventional outline. The main program calls a series of routines to initialize the system; of these, GETPAR, which reads parameters from the parameter file, and GETBDS, which reads the initial masses, positions, and velocities of the bodies, are probably of greatest interest to the user. With the data read in, the code performs an initial force calculation, and outputs system diagnostics. The code then loops NSTEPS times, advancing velocities and positions according to a simple time-centered leap-frog and recomputing forces from the new positions. The main loop is structured so that velocities and positions are synchronized at the beginning of each cycle. At regular intervals, and again at the end of the run, PUTBDS is called to output body coordinates; the final output includes masses so that it may be used as initial conditions for a further run.

Initial conditions and parameter values may be specified in any set of units with $G \equiv 1$.

8.3.1 Representation of Tree Structure

Nodes are naturally implemented as C structures or Pascal records, blocks of heterogeneous data which may be addressed by pointers. Fortran 77 does not support such objects; instead, the information for a given node is distributed across a number of arrays (*e.g.* Hernquist 1987). Let MXBODY and MXCELL be the maximum number of bodies and cells, respectively. Quantities defined for both bodies and cells, such as mass and position, are stored in arrays with indices running from 1 to MXNODE $\equiv$ MXBODY + MXCELL. Quantities defined only for bodies are stored in arrays with indices running from 1 to MXBODY, while those defined only for cells are stored in arrays with indices running from INCELL $\equiv$ MXBODY + 1 to MXNODE. The special value NULL $\equiv$ 0 is used to denote nonexistent nodes. Indices may thus refer to either bodies or cells without conflict. The type is deduced by comparing the index to the value of INCELL; if the index is less than INCELL it refers to a body, otherwise, it refers to a cell.

With these conventions adopted, the declarations needed to represent the tree structure itself, exclusive of other considerations, are

```
REAL MASS(MXNODE), POS(MXNODE,NDIM)
INTEGER SUBP(INCELL:MXNODE,NSUBC), ROOT
```

where NDIM $\equiv$ 3 is the number of dimensions, and NSUBC $\equiv$ 2**NDIM is the number of subcells within a cell. The MASS and POS arrays store the mass and position of each node. The SUBP array stores the indices of the nodes in the subcells of each cell, and the ROOT is the index of the cell enclosing the entire system. Cells have two additional arrays,

```
REAL RCRIT2(INCELL:MXNODE), QUAD(INCELL:MXNODE,NQUAD)
```

where RCRIT2 is the square of the critical distance within which each cell will fail to satisfy eq. (8.6), and QUAD is the traceless quadrupole moment, which has NQUAD $\equiv$ 5 independent components in three dimensions. Finally, in connection with N-body integration, bodies have arrays,

```
REAL VEL(MXBODY,NDIM), ACC(MXBODY,NDIM), PHI(MXBODY)
```

which store the velocity, gravitational acceleration, and the local value of the gravitational potential.

8.3.2 Tree Construction

The tree structure required for hierarchical force calculation is dynamic; it changes to reflect the mass distribution as the latter evolves. Rather than update the tree, this code simply regenerates it directly at each time-step. This is reasonable since tree construction takes only a few percent of the computational time in an equal-time-step code.

Tree construction involves two phases. The first sets up the indices which link the tree together; the second computes total masses, center-of-mass coordinates, critical radii, and quadrupole moments for all cells. Besides initializing the indices in the SUBP array, the first phase generates some information of use to the second phase. In particular, it stores the linear size and geometric midpoint of each cell in the CLSIZE and MID arrays, respectively. To save memory, these arrays have been equivalenced to the common arrays RCRIT2 and POS. The second phase of tree construction, after making use of the data in these arrays, reuses them to store the squared critical radius and center-of-mass of each cell.

MKTREE. This routine governs tree construction; besides timing, it does no more than invoke the routines EXPBOX, LDTREE, and HACKCM.

EXPBOX. The purpose of this routine is to define a cube enclosing the entire particle configuration, with linear dimensions which are an integer power of two. This guarantees that cell midpoints will be accurately represented in binary floating-point arithmetic.

LDTREE. This routine creates an empty tree consisting of one cell, indexed by the ROOT, which represents the entire volume of space defined by EXPBOX. With this cell initialized, the routine loops over all bodies, calling LDBODY to add each body in turn to the tree structure.

LDBODY. This routine installs the body indexed by P into the tree structure attached to the ROOT cell. In doing so it also adds new cells as needed to insure that each body remain isolated in a single subcell. To find the correct place to insert P, this routine follows a trail from the root cell toward smaller cells, zeroing in on the single correct subcell. This is accomplished by the following code:

```
      Q = ROOT
      QIND = SBINDX(P, Q)
   10 IF (SUBP(Q,QIND) .NE. NULL) THEN
        IF (SUBP(Q,QIND) is a body) THEN
          extend the tree with a new cell
        ENDIF
        Q = SUBP(Q,QIND)
        QIND = SBINDX(P, Q)
        GOTO 10
      ENDIF
      SUBP(Q,QIND) = P
```

If another body is already occupying the targeted subcell, a new cell must be added to the tree. The new cell is one-eighth the volume of the cell holding the original occupant; that is, it is exactly the volume of the contested subcell, and its midpoint is offset from its parent cell in the appropriate direction. Once the new cell is initialized, the old occupant is installed within it, as shown in Fig. 8.3. This case is handled *inside* the main loop of the routine, since it may come up several times in succession before separate subcells are available for the two contesting bodies.

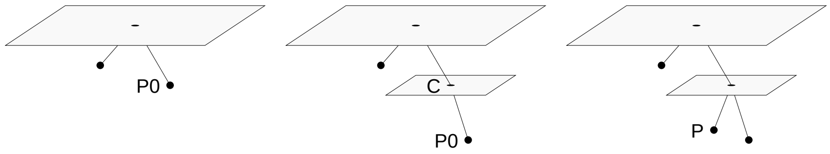

Fig. 8.3. Adding a cell to the tree structure. On the left is the situation before the tree is extended, with a body P0 already occupying the subcell. In the middle the new cell C has been linked into the tree, and P0 has been inserted within it. On the right is the situation one cycle later with the new body P in place.

SBINDX. An integer-valued function which accepts the indices of a body P and a cell Q. It checks that P is somewhere within the volume represented by Q and returns the index of the subcell of Q which encloses P.

MKCELL. An integer-valued procedure which initializes an empty cell and returns its index.

HACKCM. This routine performs the second phase of tree construction, computing total masses, center-of-mass positions, critical radii, and quadrupole moments for all cells. This phase typically involves the flow of information from the leaves toward the root of the tree. It is therefore implemented with loops which scan the cells in order of increasing size; this insures that the subcells of any cell are processed before the cell itself is. Before the first such loop, cells are listed in order of decreasing size by the BFLIST routine. The first loop computes total masses, center-of-mass positions, and critical radii; if required, a second loop computes quadrupole moments.

BFLIST. This routine scans the tree in breadth-first order, visiting all cells on each level before going on to the next. It stores indices of the cells visited in the IND array; on return, this array lists cells in order of decreasing size. The main loop terminates when it reaches a level of the tree without any cells; further levels are empty since bodies have no descendents.

8.3.3 Force Calculation

To compute the force on a body, the code makes a partial traverse of the tree structure. The goal is not to visit every node; if a sufficiently accurate approximation to the force can be obtained from a given cell, there is no need to visit its descendents.

ACCEL. This routine repeatedly calls TRWALK to compute the gravitational acceleration and potential of each body.

TRWALK. The heart of the algorithm is contained in this routine, which calculates the gravitational force on a single body P. It uses a stack to hold nodes yet to be examined; when this stack becomes empty, the tree has been traversed. Initially, the ROOT is placed on the stack as the first node to examine. Within the main loop, the node on the top of the stack is removed and examined to decide what to do. Bodies and sufficiently distant cells contribute directly to the gravitational force on P. Cells which do not satisfy eq. (8.6) are too close to P to yield accurate forces; instead, their subcells are pushed onto the stack, and control transfers back to the top of the main loop. In outline, the code to compute the force runs as follows:

```
      SPTR = 1
      STACK(SPTR) = ROOT
 10   IF (SPTR .GT. 0) THEN
        Q = STACK(SPTR)
        SPTR = SPTR - 1
        IF (Q is a body) THEN
          process body-body interaction
        ELSE IF (Q is far enough from P) THEN
          process body-cell interaction
        ELSE
          DO 20 K = 1, NSUBC
            IF (SUBP(Q,K) .NE. NULL) THEN
              SPTR = SPTR + 1
              STACK(SPTR) = SUBP(Q,K)
            ENDIF
 20       CONTINUE
        ENDIF
        GOTO 10
      ENDIF
```

8.3.4 Vectorized Force Calculation

A fully vectorized version of the TREE code would require substantial revision of a number of routines, but almost all of the run time is spent in the TRWALK routine, and a large speed-up can be obtained by vectorizing this routine alone. The code here is based on Hernquist's (1990) level-by-level vectorization of the tree search.

ACCEL. This routine now calls GRVSUM to sum the forces exerted by the bodies and cells listed by TRWALK.

TRWALK. Level-by-level vectorization involves maintaining a list of NACT nodes to be examined, stored in the ACTIVE array. On each cycle through the main loop, bodies and cells which satisfy eq. (8.6) are moved to holding arrays BODYTM and CELLTM for later processing. The offspring of the remaining cells are stored in the ACTSUB array, and the non-null elements of this array become the ACTIVE array for the next cycle. The code listed here is schematic; for details, see the actual sources:

```
      NACT = 1
      ACTIVE(NACT) = ROOT
   10 IF (NACT .GT. 0) THEN
        DO 20 I = 1, NACT
          IF (ACTIVE(I) is a body) THEN
            list body-body interaction
          ELSE
            retain cell in ACTIVE array
          ENDIF
   20   CONTINUE
        DO 30 I = 1, NACT
          IF (ACTIVE(I) is far enough from P) THEN
            list body-cell interaction
          ELSE
            retain cell in ACTIVE array
          ENDIF
   30   CONTINUE
        DO 40 I = 1, NACT
          store subcells of ACTIVE(I) in ACTSUB array
   40   CONTINUE
        DO 50 I = 1, NSUB
          if ACTSUB(I) is not null, copy to ACTIVE array
   50   CONTINUE
        GO TO 10
      ENDIF
```

The inner loops in this routine can all vectorized by the **CFT77** compiler.

GRVSUM. This routine performs vectorized summations over the BODYTM and CELLTM arrays to compute the total force on P.

8.4 A Tree Code Users Guide

This section describes how to compile the code, the input and output files it needs and produces, and a suite of tests which may be run to verify that it is working as intended.

8.4.1 Compiling the Code

As distributed with this volume, the TREE code consists of the following files:

treedefs.f – parameters and common blocks.
treecode.f – main program and integration.
treeload.f – tree construction.
treegrav.f – recursive force calculation.
treevect.f – vectorized force calculation.
treeio.f – I/O routines.

The only nonstandard **F77** feature used in these source files is the INCLUDE statement, which inserts the contents of treedefs.f at the beginning of each subroutine. The functionality of INCLUDE is so useful that on systems where this statement is not available, another mechanism for accomplishing the same objective generally exists. At a last resort, treedefs.f could be manually inserted wherever an INCLUDE statement appears.

In addition, there are several files useful when compiling the TREE code on a **UNIX** operating system:

makefile – used with the **UNIX** *make* facility to organize compilation.
second.c – a **C** procedure to measure CPU time used.

On a **UNIX** system it is sufficient to issue the commands

```
% make treecode
% make treevect
```

to compile versions of the code for scalar and vector processors, respectively.

On other operating systems the *make* facility may not exist; it is suggested that the user create a command file to organize compilation of the various source files. The files treecode.f, treeload.f, treegrav.f (or treevect.f), and treeio.f may be compiled individually. It may prove impossible to compile second.c. In this case, substitute the file

second.f – dummy **F77** subroutine to measure CPU time used.

This dummy routine should be replaced by a functional version.

The parameter statements in treedefs.f set MXBODY = 8192 and MXCELL = 6144. These are enough for modest tests of the code, such as described below. For larger calculations these parameters may be increased proportionately.

8.4.2 Running the Code

Once the code has been correctly compiled, two input files are needed to start a calculation. These are

treebodi – masses, initial positions and velocities.
treepars – parameters and control options.

The first file specifies the initial conditions for all bodies; the format of this file may be easily deduced by examining the GETBDS routine in treeio.f. The second file contains the parameters which control the course of the calculation. In the order given, these are

HDLINE – a message of up to 32 letters, used to label output.
NSTEPS – number of time-steps to take during the run.
NOUT – number of time-steps between body data outputs.
DTIME – time-step Δt in eq. (8.9).
THETA – accuracy parameter θ in eq. (8.6).

EPS – softening length ε in eqs. (8.7) and (8.8).
USQUAD – include quadrupole corrections for cells.

After the code has been run, several output files will be generated. These are

treebodo – positions and velocities of all bodies, output every nout time-steps.
treebodf – masses, final positions, and final velocities of all bodies. This file may be used as input for another run.
treelogs – a log of the run, listing the total energy ETOT and other information at each time-step.

8.4.3 Testing the Code

Included with the other files is a stand-alone program, testdata.f, which may be used to generate initial conditions and parameter files for various tests of the tree code. testdata.f is an interactive program which prompts the user for three input parameters:

TEST – an integer selecting the test to perform.
NBODY – N, the number of bodies to use.
SEED – an integer used to start the random number generator.

When run, testdata.f produces files treepars and treebodi which may be used as inputs to the tree code. The tests which may be selected are as follows:

1. Equilibrium Plummer model (Aarseth *et al* 1974).
2. Head-on parabolic collision of two Plummer models.
3. Off-axis parabolic collision of two Plummer models.
4. Generalized polytrope (Hénon 1973) with indices $n = 1$, $m = -1$.
5. Simplified "detonating galaxy" (Salmon & Warren 1994).
6. Binary hierarchy of $\lfloor \log_2 N \rfloor$ levels (Soneira & Peebles 1978).

All of these tests should be run with at least 1024 bodies. Note that test 6 will round N down to an integer power of 2.

Test 1 checks the stability of a spherical equilibrium configuration; no significant evolution of this system should occur. Tests 2 and 3, on the other hand, exercise the code in a non-equilibrium situation: the encounter of a pair of spherical "galaxy" models. Both of these encounters eventually result in the merger of the original systems.

Test 4 illustrates a dynamical instability observed in spherical systems with radial orbits (*e.g.*, Merritt 1987). The initial conditions describe a highly anisotropic equilibrium system in which all orbits pass through the origin. Over ~ 10 time units this system evolves from a sphere into an elongated bar.

Test 5 exhibits a potential pathology of the original algorithm, described in detail by Salmon & Warren (1994). A single massive body and an extended "galaxy" six times less massive are placed in orbit about each other. When run with the code supplied here this calculation is performed correctly. To detonate the galaxy, change the line assigning RCRIT2(P) in subroutine HACKCM to

```
RCRIT2(P) = (CLSIZE(P) / THETA)**2
```

and delete the test for "self-interaction" at the end of subroutine TRWALK. With these modifications the code exhibits pathological behavior. For $\theta = 1$ the pathology is dramatic: energy and momentum conservation are violated, and the victim galaxy sheds a good deal of mass as it enters the cell with $\ell = 8$ centered on the point $(x, y, z) = (4, 4, 4)$. This cell's center of mass is dominated by the massive body, which lies at the opposite corner from the galaxy; thus the part of the victim which falls within this cell is effectively lost to the self-gravity of the galaxy. The revised criterion (eq. 8.6) used in the present code cures this problem by insuring that the offending cell is not used to calculate the galaxy's self-gravitational attraction.

Test 6 exercises the code on a highly irregular mass distribution. Over time the fractal structure initially present will be erased, eventually leaving a relaxed, "monolithic" system (White & Negroponte 1982).

It is instructive to repeat these tests with different θ, ε, and Δt values, and to vary N so as to contrast uncertainties due to finite particle number with numerical errors. After running a test, examine the treelogs file to see if total energy and momentum have been adequately conserved. Energy should be conserved to $1/\sqrt{N}$ or better in most cases; poor energy conservation is generally due to too long a time-step, although force calculation errors also degrade energy conservation. The system center of mass will drift as a result of force calculation errors; this probably has little consequence if the distance drifted is smaller than the softening length ε. The initial data generated for tests 3, 5, and 6 have significant angular momentum, which should be conserved to $1/\sqrt{N}$. In the other tests, the total angular momentum is only of order $1/\sqrt{N}$; relatively large fluctuations in this residual angular momentum are to be expected, but unlikely to compromise the results.

Conservation of global quantities, while reassuring, does *not* prove the correctness of a calculation. A stronger case may be made by showing that the results converge on a unique answer as force calculation and time integration errors are jointly refined. This may be done by running a series of models, all starting from the same initial conditions, while systematically varying θ and Δt. As these parameters are reduced, trajectories of individual particles converge towards definite limits (*e.g.*, Barnes & Hut 1989; Barnes 1990). Any systematic program of numerical experimentation should include convergence testing to assess the quality of the results.

I thank Lars Hernquist for helpful discussions and Pavel Curtis for a careful reading of this manuscript. The vectorized code was tested at the Pittsburgh Supercomputing Center.

References

Aarseth, S. *et al* 1974, *Astron. Astrophys.,* **37**, 183.
Appel, A. 1981, *B.A. thesis*, Princeton University.
Appel, A. 1985, *SIAM J. Stat Sci. Comput.,* **6**, 85.
Barnes, J. 1986, in *The Use of Supercomputers in Stellar Dynamics*, eds. P. Hut & S. McMillan (Berlin: Springer-Verlag), p. 175.
Barnes, J. 1990, *J. Comp. Phys.,* **87**, 161.
Barnes, J. & Hut, P. 1986, *Nature* **324**, 446.
Barnes, J. & Hut, P. 1989, *Ap. J. Supp.* **70**, 389.
Benz, W. *et al* 1990, *Ap. J.* **348**, 647.
Binney, J. & Tremaine, S. 1987, *Galactic Dynamics* (Princeton: Princeton University Press).
Greengard, L. 1990, *Computers in Physics,* **4**, 142.
Greengard, L. & Rokhlyn, V. 1987, *J. Comput. Phys.,* **73**, 325.
Hénon, M. 1973, *Astron. Astrophys.,* **24**, 229.
Hernquist, L. 1987, *Ap. J. Sup.,* **64**, 715.
Hernquist, L. 1990, *J. Comp. Phys.,* **87**, 137.
Hernquist, L. & Barnes, J. 1990, *Ap. J.,* **349**, 562.
Hernquist, L. & Katz, N. 1989, *Ap. J. Sup.,* **70**, 419.
Hockney, R. & Eastwood, J. 1981, *Computer Simulation Using Particles* (New York: McGraw-Hill).
Jernigan, J. 1985, in *Dynamics of Star Clusters*, eds. J. Goodman & P. Hut (Dordrecht: D. Reidel), p. 275.
Katzenelson, J. 1989, *SIAM J. Stat Sci. Comput.,* **10**, 787.
Knuth, D. 1973, *Fundamental Algorithms* (Reading: Addison-Wesley).
Makino, J. 1990, *J. Comp. Phys.,* **88**, 393.
McMillan, S. & Aarseth, S. 1993, *Ap. J.,* **414**, 200.
Merritt, D. 1987, in *Structure and Dynamics of Elliptical Galaxies*, ed. T. de Zeeuw (Dordrecht: D. Reidel), p. 315.
Porter, D. 1985, *Ph.D. thesis*, University of California, Berkeley.
Press, W. 1986, in *The Use of Supercomputers in Stellar Dynamics*, eds. P. Hut & S. McMillan (Berlin: Springer-Verlag), p. 184.
Salmon, J. & Warren, M. 1994, *J. Comp. Phys.,* **111**, 136.
Sellwood, J. 1987, *Ann. Rev. Astron. Astrophys.,* **25**, 151.
Soneira, R. & Peebles, J. 1978, *Astr. J.,* **83**, 845.
van Albada, T. 1986, in *The Use of Supercomputers in Stellar Dynamics*, eds. P. Hut & S. McMillan (Berlin: Springer-Verlag), p. 23.
White, S. 1983, in *The Origin and Evolution of Galaxies*, eds. B. Jones & J. Jones (Dordrecht: D. Reidel), p. 227.
White, S. & Negroponte, J. 1982, *Mon. Not. R. astr. Soc.,* **201**, 401.

This article was processed by the author using the TEX macro package from Springer-Verlag.