JOURNAL OF COMPUTATIONAL PHYSICS **87**, 161–170 (1990)

A Modified Tree Code: Don't Laugh; It Runs

JOSHUA E. BARNES*

Institute for Advanced Study, Princeton, New Jersey 08540

Received July 19, 1988; revised April 25, 1989

I describe a modification of the Barnes–Hut tree algorithm together with a series of numerical tests of this method. The basic idea is to improve the performance of the code on heavily vector-oriented machines such as the Cyber 205 by exploiting the fact that nearby particles tend to have very similar interaction lists. By building an interaction list good everywhere within a cell containing a modest number of particles and reusing this interaction list for each particle in the cell in turn, the balance of computation can be shifted from recursive descent to force summation. Instead of vectorizing tree descent, this scheme simply avoids it in favor of force summation, which is quite easy to vectorize. A welcome side-effect of this modification is that the force calculation, which now treats a larger fraction of the local interactions exactly, is significantly more accurate than the unmodified method. © 1990 Academic Press, Inc

I. INTRODUCTION

Hierarchical or "tree" methods of solving the gravitational N-body problem have attracted increasing attention in recent years [1–9]. Tree methods are interesting because they can handle arbitrarily complicated mass distributions with asymptotic computing times of $O(N \log N)$ or even $O(N)$ in terms of the particle number N, as opposed to $O(N^2)$ for a direct summation code. The main application of hierarchical methods is thus to problems involving very large N, say 10^4 or more, where the advantage over older methods is greatest. In practice, this focus on large-N problems means that tree codes must run well on vector-oriented supercomputers, since at present such machines offer the only large computing resource available to most experimenters.

These factors have motivated several workers to consider implementing tree algorithms on present-generation supercomputers. This problem is not trivial, because the recursive flow of control required to descend the tree structure is at odds with the linear organization of vector processors. Vectorization of the Barnes and Hut [4] tree algorithm has been approached in several different ways. To begin with, Hernquist [7] noted that the force calculation on a particle p can be divided into two phases: recursive tree descent, constructing an *interaction list* of all particles and cells which interact with p, and force summation, which involves adding up all the terms on the interaction list. Force summation is easily vectorized, delivering a

* Current address: Canadian Institute for Theoretical Astrophysics, 60 Saint George Street, Toronto, Ontario, Canada M5S 1A1.

significant speedup over an unvectorized code, but much scalar CPU time is still spent descending the tree. A completely vectorized approach is described by Makino [10]; vectorizing the tree search "across particles," the algorithm performs independent tree descents for many particles simultaneously. This elegant method can be viewed as simulating a tree algorithm for a "fine-grained" parallel processor such as the Connection Machine [11] on a conventional vector processor; in practice, however, it requires hardware features such as indirect addressing not uniformly available on all vector machines. Yet another approach has been developed by Hernquist [12], vectorizing tree descent "level by level," a procedure which works well on a Cray.

A major regularity present in all hierarchical algorithms is that the representation of the gravitational field used to calculate the force on particle p is very similar to the representation used to compute the force on nearby particle q. This regularity is exploited in, for example, the fast multiple method (FMM) of Greengard and Rokhlin [5], which constructs a multiple expansion of the field due to the mass external to some chosen cell, and then uses this expansion to evaluate the external force on each particle within the cell. The expense of constructing the field expansion is thus shared between the particles within the cell. Clearly, there is an optimum number of particles per cell—too few, and the cost of the multipole expansions dominates: too many, and the cost of evaluating the local field by direct summation dominates. This optimum choice is briefly discussed in connection with an adaptive version of the FMM [8].

The same sort of strategy can be used to get around the problem of vectorizing the Barnes–Hut algorithm. If n is a node on the interaction list L_p of p, and the distance between n and p is much greater than the distance between p and q, then most likely $n \in L_q$. This leads to the suggestion that Hernquist's two-phase [7] method be modified to construct an interaction list L_c guaranteed to satisfy the usual Barnes–Hut tolerance condition $l/d < \theta$ everywhere within a small cell c, containing p, q, and a few other particles. L_c may then be (re)used to evaluate the force on p, q, etc. in turn, effectively reducing the number of tree descents required by a factor equal to the number of particles in c. Instead of vectorizing tree descent, this "non-vectorization" amounts to avoiding, as much as possible, that part of the calculation which is hard to vectorize. Details are presented in Section II.

By necessity, the interaction list L_c will generally contain more information than the list L_p for any particle p within c; if node $n \in L_p$, then either n or its descendants $\in L_c$. As a consequence, the modified method should deliver somewhat more accurate forces than the original Barnes–Hut algorithm. In fact, this improvement is large enough to be important when comparing the modified method to other versions of the Barnes–Hut algorithm. Section III presents tests of the modified algorithm, including a stringent series of experiments demonstrating that as the timestep Δt and force calculation tolerance θ are jointly refined, the trajectories of individual particles converge to a definite limit. Timing tests presented in Section IV show that the modified algorithm is significantly faster when run on a vector processor.

II. Descriptions of the Method

Where an implementation op the Barnes–Hut algorithm would normally loop over particles from 1 to N and compute the force on each in turn, the modified algorithm must loop over the members of a selected set of cells C, constructing an interaction list for each cell c in C and using it to compute forces on all particles within c. In the present implementation, a cell c is in C if it encloses a total of n_{crit} or fewer particles *and* its parent cell encloses more than n_{crit} particles. Thus the set C covers the entire simulation volume without overlap, partitioning it into cells containing at most n_{crit} particles each; moreover, C is the smallest set with these properties.

The **sequence-force-calculation** routine does a depth-first recursive descent of the tree structure, starting at the **tree-root**. When it finds a cell enclosing **n-critical** or fewer particles, it constructs an interaction list good everywhere within that cell and invokes **perform-force-calculation**. In the SCHEME dialect of Lisp [13], this algorithm may be stated as follows:

```
(define tree-root ...)
(define n-critical ...)
(define (sequence-force-calculation node)
  (cond ((body? node)
         (perform-force-calculation node (interaction-list node tree-root)))
        ((< (particle-count node) n-critical)
         (perform-force-calculation node (interaction-list node tree-root)))
        (else
         (do ((desc (descendents node) (cdr desc)))
             ((null? desc) ( ))
         (sequence-force-calculation (car desc))))))
```

The next routine, **perform-force-calculation**, continues the recursive descent, using **inter-list** to evaluate the force on each particle it comes to.

```
(define (perform-force-calculation node inter-list)
  (cond ((body? node)
         (set-force! node (sum-force node inter-list)))
        (else
         (do ((desc (descendents node) (cdr desc)))
             ((null? desc) ( ))
         (perform-force-calculation (car desc) inter-list)))))
```

Here **sum-force** is a vectorized force summation routine which hides details of the force calculation, including evaluation of the quadrupole contribution and a correction for self-interaction.

The **interaction-list** function walks the tree structure and constructs a list of particles and cells obeying the Barnes–Hut opening-angle criterion with respect to any point within **node**.

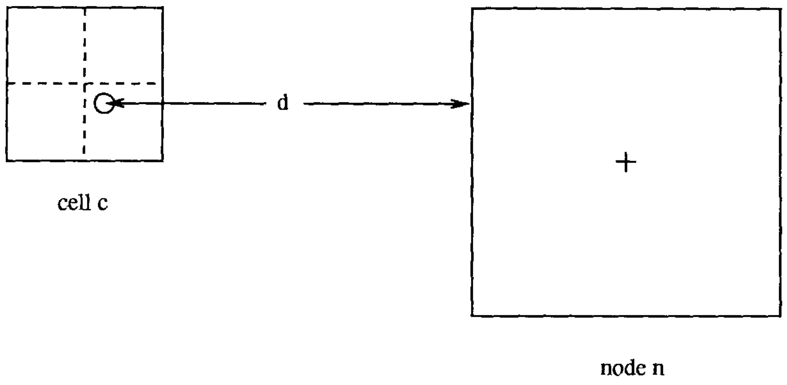

FIG. 1. The **cooked-distance** is the distance *d* from the center of mass (○) of cell *c* the nearest point in node *n*.

```
(define (interaction-list node tree)
  (cond ((must-subdivide? tree node)
         (concat
          (map (lambda (t) (interaction-list node t))
               (descendents tree))))
        (else
         (list tree))))
```

Here the **map** function constructs a list of interaction lists for the descendents of **tree**, which are then spliced together by the **concat** function.

The predicate **must-subdivide?** decides if **interaction-list** must examine the descendents of **tree** to construct an interaction list valid everywhere within **node**.

```
(define theta ...)
(define (must-subdivide? tree node)
  (and (cell? tree)
       (> (diameter tree)
          (* theta
             (cooked-distance (cm-position tree)
                              (centroid node)
                              (diameter-node))))))
```

This test is very similar to the standard Barnes–Hut algorithm, except that the *minimum* distance **cooked-distance** between the center of mass of **tree** and any point within **node** is computed; see Fig. 1.

III. NUMERICAL TESTS

To establish the reliability of the modified tree algorithm, I ran an extensive series of test calculations. It seemed crucial to check the code on a reasonably realistic problem, such as the head-on collision and merger of two spherical "galaxies" used here. There are, however, no known analytic solutions with which to compare such calculations, so the validity of the numerical models had to be tested by showing that, as sources of error were reduced, the results converged to a well-defined limit. This in itself does not prove the limit is the right one, but by appealing to the existence and uniqueness of solutions of ODEs, a good case can be made for the correctness of the N-body calculations.

The tests presented here employed a head-on, parabolic encounter between two $W_c = 5$ King [14] models. Units were chosen so that $G = 1$ and each galaxy had a total mass and three-dimensional *rms* velocity dispersion of unity. A total of $N = 4096$ particles were used, gravitational potentials were softened by the usual $(r^2 + \varepsilon^2)^{-1/2}$ form with $\varepsilon = 0.025$, and the equations of motion were integrated with a time-centered leap-frog. The galaxies were released 2 length units apart and given velocities consistent with falling together from ∞. The evolution of the system is shown in Fig. 2. The galaxies passed through each other at $t \simeq 1$ time unit and separated by ~ 1.5 length units before falling back together and finally merging at $t \simeq 4$. On the whole, the results of these simulations are quite consistent with earlier head-on merger calculations [15].

(a) *Convergence in the Δt, θ Plane*

For the first test, a series of models were run varying the force accuracy and integration timestep but fixing all other parameters. After some preliminarry experimentation, the maximum number of particles sharing an interaction list was set to $n_{\text{crit}} = 64$. This gave a speedup of ~ 3 over an unmodified Barnes–Hut code when compared on a Cyber 205; a similar improvement was obtained on a Cray XMP. The parameter grid is shown in Fig. 3. Since all runs started from exactly the same initial data, their evolution could be compared on a particle-by-particle basis. Let $\mathbf{r}_i(t; a)$ and $\mathbf{r}_i(t; b)$ be the position vectors of particle i at time t in simulations a and b, respectively. At first, $\mathbf{r}_i(0; a) = \mathbf{r}_i(0; b)$, but the trajectories will subsequently diverge due to differences in force calculation and/or integration. The numbers tabulated between the runs are medians of $\Delta r_i \equiv |\mathbf{r}_i(t; a) - \mathbf{r}_i(t; b)|$, evaluated at $t = 4$. Since the distribution of Δr_i does not possess, for example, an unusually long tail, the median is a good indication of the overall difference between simulations. Further discussion of the Δr_i distribution is postponed pending a more complete investigation.

Examining Fig. 3, it is clear that as the force calculation and time integration are jointly refined, the calculated trajectories do indeed converge to a definite limit. This is a reassuring if not unexpected result; one would scarcely believe a simulation of a *collisionless* system which did not pass this kind of test. It is, however, gratifying to note that the errors due to these two parameters appear to be largely independent of each other; the overall improvement on refining Δt does not depend on the value of θ, and vice versa.

It is also interesting if not surprising that for a given value of θ, the modified algorithm appears to be significantly more accurate than the original Barnes–Hut algorithm [9]. This increased accuracy comes at a price, of course; the more par-

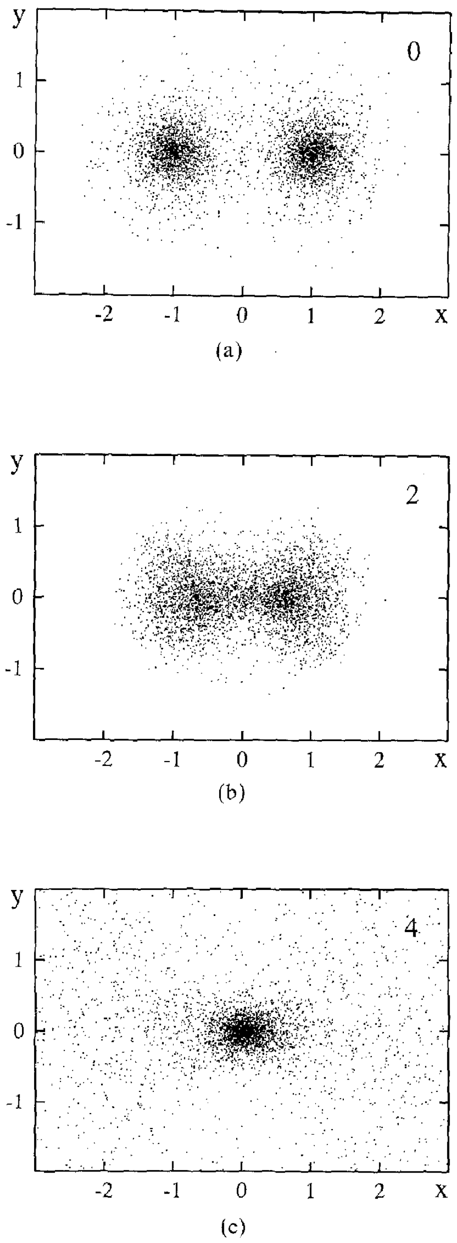

FIG. 2. Head-on collision and merger of two spherical galaxies; elapsed time in model units is shown at the upper right of each frame.

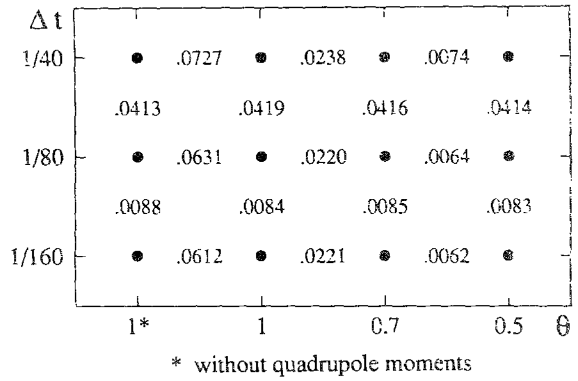

FIG. 3. Grid of models in timestep vs force calculation accuracy plane, all started from the same initial conditions; each dot represents a calculation. The numbers plotted between models are median distances of corresponding particles after $t = 4$ time units. The left-most column of runs was made without quadrupole corrections; all others included them.

ticles sharing the interaction list, the longer the list, and hence the *more* time spent in vectorized force summations. To attain a given level of force calculation accuracy, is it better to reduce θ or increase $n_{\rm crit}$? This question motivated the next set of tests.

(b) *Convergence in the θ, $n_{\rm crit}$ Plane*

In the second test θ and $n_{\rm crit}$ were varied, while all other parameters were fixed, with a timestep $\Delta t = 1/80$. The resulting grid is shown in Fig. 4, including data from

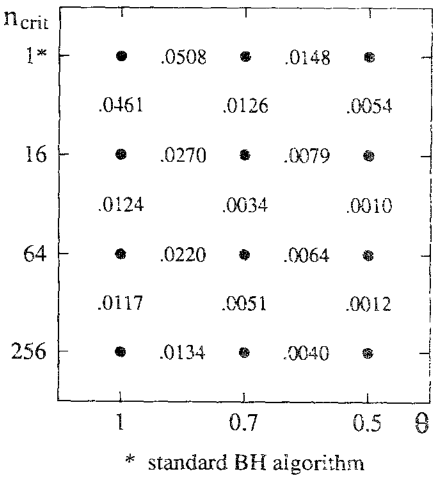

FIG. 4. Grid of models in $n_{\rm crit}$ vs θ plane, all started from the same initial conditions, each dot represents a calculation. The numbers plotted between models are median distances of corresponding particles after $t = 4$ time units. The top row of runs was made with the standard Barnes–Hut algorithm; all other runs were made using the modified algorithm.

the standard Barnes–Hut algorithm [9] along the top. The pattern of median Δr_i values shows that as either $\theta \to 0$ or $n_{\text{crit}} \to N$, the modified algorithm converges on a definite answer. This is expected since, in either limit, the modified algorithm reverts to an exact, $O(N^2)$ scheme.

Before comparing the accuracy of the old and new methods, the character of remaining errors must be considered. Much of the error removed by the modified algorithm was presumably contributed by nearby cells, which typically contain only a few particles each, and therefore tend to have large statistical octopole and higher moments. On the other hand, the effect of reducing θ is to sample the mass distribution more finely on all scales. Thus, although $\theta \to 0$ and $n_{\text{crit}} \to N$ both approach the same limit, they nevertheless do so from somewhat different "directions." In other words, the distributions of remaining errors are different; the modified algorithm is more faithful locally. With this point noted, the modified algorithm working at $\theta = \theta_1$ and $n_{\text{crit}} = 64$ appears comparable in accuracy to the Barnes–Hut algorithm working at $\theta = 0.7\theta_1$, as inferred from the median Δr_i values.

IV. Timing Results

How much faster is the modified algorithm? Unlike the questions of accuracy considered above, this one cannot be answered without reference to the machine running the code. To obtain any real advantage from this scheme, the vector performance of the hardware must significantly exceed the scalar performance; only then will the speedup from the force summation phase significantly outweigh the cost of the more complicated tree-search and the generally longer interaction lists. In general, this is just the balance of performance offered by conventional array processor and supercomputer systems.

Table I shows force-calculation timings on a Sun 3/60 and Cyber 205 as a function of θ and n_{crit}. For comparison, timings of the Barnes–Hut algorithm are also listed—these are for a version of the code including Hernquist's partial vec-

TABLE I

CPU Timings in Seconds as a Function of θ (Over) and n_{crit} (Down) for a Sun 3/60 and Cyber 205, Measured Using a Plummer Model with $N = 8192$ Particles

	Sun			Cyber		
	1.0	0.7	0.5	1.0	0.7	0.5
1[a]	1155	2089	3698	18.9	33.9	60.3
16	1562	2635	4483	11.1	16.5	26.7
64	1885	2832	4805	8.4	11.1	17.1
256	2212	3096	5175	8.1	10.2	15.3

[a] Standard BH algorithm.

torization [7, 9]. These times are measured in seconds using a spherical Plummer model with $N=8192$ particles. Tests with $2048 \leqslant N \leqslant 16384$ particles are consistent with $O(N \log N)$ scaling. On a scalar processor such as the Sun 3/60, the modified algorithm is slower than the original method, because the time saved in the tree-descent is less than the time lost evaluating the longer interaction lists. On the Cyber 205, on the other hand, the modified algorithm is between 2 and 3 times faster than Barnes–Hut algorithm for a given value of θ, with the greatest speedup found for small θ. This speedup is not very sensitive to n_{crit} in the range studied, although for much larger n_{crit} the speedup will be lost as the $O(N^2)$ limit is reached. When the increased accuracy of the modified method is taken into account, the effective speedup is a factor of 3 to 5.

Analysis of the modified algorithm at a level of detail sufficient to predict these timing results is extremely difficult, since the statistical properties of the mass distribution [9] and the performance of the computer hardware are both involved. A simple argument, however, can give the overall behavior. The CPU time required to evaluate the force on all N particles using the original Barnes–Hut algorithm is

$$T = N(T_L + T_F),$$

where T_L is the average time required to construct an interaction list for one particle, and T_F is the average time to sum the total force on one particle, given the interaction list. Both T_L and T_F are (roughly) proportional to the length of a typical interaction list, which is $O(\log N)$, giving $T \propto N \log N$ asymptotically [4]. The CPU time for the modified algorithm is

$$T' = N_L T'_L + N T'_F,$$

where N_L is the number of interaction lists constructed, T'_L is the average time to construct an interaction list for a cell, and T'_F is the average time to sum the force on a particle. Introducing the average number of particles handled per interaction list, $\bar{n} \equiv N/N_L \leqslant n_{\text{crit}}$, gives

$$T' = N(T'_L/\bar{n} + T'_F)$$

for the modified algorithm. Hence if $T'_L \gg T'_F$, as is the case on the Cyber 205, the modified algorithm can be faster than the original. As n_{crit} is increased, the number of interaction lists N_L is reduced, but the length of each interaction list increases, and hence so do T'_L and T'_F. The optimum value of n_{crit} depends on the ratio of scalar to vector performance but is, in the large N limit, independent of N itself.

V. Conclusions

Empirical tests show that the modified tree code performs well on a vector-oriented machine such as a Cyber 205. For a given θ, the typical gain over

Hernquist's two-phase formulation is somewhat smaller than the gain offered by the vectorization methods discussed in [10, 12]. However, for a given level of error, quantified by measuring particle displacements at the end of the calculation, the modified method permits a somewhat larger value of θ. In terms of overall throughput, therefore, the modified method is not at a disadvantage compared to other vectorization schemes. The primary advantages of the modified method are *portability* and *simplicity*. On the software level, the code is written entirely in standard F77, with no "vector" extensions or special library calls. On the hardware level, the code demands only efficient processing of longish vectors; hardware gather/scatter and conditional operations are not needed. The modified method is only slightly more complicated than the original Barnes–Hut algorithm; it took less than three hours and one hundred additional lines of code to implement.

ACKNOWLEDGMENTS

I thank colleagues at the IAS and at MIT for many helpful discussions on vectorization of treecodes and empirical measurement of errors in N-body simulations. Numerical calculations were run at the John von Neumann Supercomputer Center under Grant LAC-17016. I acknowledge support from New Jersey High Technology Grant 88-240090-02.

REFERENCES

1. A. APPEL, *SIAM J. Sci. Statist. Comput.* **6**, 85 (1985).
2. J. G. JERNIGAN, in *Dynamics of Star Clusters, Princeton, New Jersey, 1984*, edited by J. Goodman and P. Hut (Reidel, Dordrecht, 1985), p. 275.
3. D. PORTER, Ph.D. thesis, University of California, Berkeley, 1985 (unpublished).
4. J. E. BARNES AND P. HUT, *Nature* **324**, 446 (1986).
5. L. GREENGARD AND V. ROKHLIN, *J. Comput. Phys.* **73**, 325 (1986).
6. W. PRESS, in *The Use of Supercomputers in Stellar Dynamics, Princeton, New Jersey, 1986*, edited by P. Hut and S. McMillan (Springer-Verlag, Berlin, 1986), p. 184.
7. L. HERNQUIST, *Astrophys. Suppl.* **64**, 715 (1987).
8. J. CARRIER, L. GREENGARD, AND V. ROKHLIN, *Siam J. Sci. Statist. Comput.* **9**, 669 (1988).
9. J. E. BARNES AND P. HUT, *Astrophys. J. Suppl.* **70**, 389 (1989).
10. J. MAKINO, *J. Comput. Phys.* **87**, 148 (1990).
11. D. HILLIS, *The Connection Machine* (MIT Press, Cambridge, MA, 1985).
12. L. HERNQUIST, *J. Comput. Phys.* **87**, 137 (1990).
13. J. REES AND W. CLINGER, *Revised^3 Report on the Algorithmic Language Scheme*, MIT AI Memo 848a, 1986 (unpublished).
14. I. KING, *Astronom. J.* **76**, 64 (1966).
15. T. VAN ALBADA AND J. VON GORKOM, *Astronom. Astrophys.* **54**, 121 (1977).