Journal of Computational Physics **178,** 1–14 (2002)
doi:10.1006/jcph.2001.6943, available online at http://www.idealibrary.com on IDEAL®

A Performance Comparison of Tree Data Structures for *N*-Body Simulation

J. Waltz,[1] G. L. Page, S. D. Milder, J. Wallin, and A. Antunes

Institute for Computational Sciences and Informatics, George Mason University, Fairfax, Virginia 22030

Received June 1, 2000; revised September 14, 2001

We present a performance comparison of tree data structures for N-body simulation. The tree data structures examined are the balanced binary tree and the Barnes–Hut (BH) tree. Previous work has compared the performance of BH trees with that of nearest-neighbor trees and the fast multipole method, but the relative merits of BH and binary trees have not been compared systematically. In carrying out this work, a very general computational tool which permits controlled comparison of different tree algorithms was developed. The test problems of interest involve both long-range physics (e.g., gravity) and short-range physics (e.g., smoothed particle hydrodynamics). Our findings show that the Barnes–Hut tree outperforms the binary tree in both cases. However, we present a modified binary tree which is competitive with the Barnes–Hut tree for long-range physics and superior for short-range physics. Thus, if the local search time is a significant portion of the computational effort, a binary tree could offer performance advantages. This result is of particular interest since short-range searches are common in many areas of computational physics, as well as areas outside the scope of N-body simulation such as computational geometry. The possible reasons for this are outlined and suggestions for future algorithm evaluations are given.

Key Words: tree data structures; N-body problem; computational astrophysics.

1. INTRODUCTION

The modern impetus for the study of the N-body problem [1] has come from astrophysics, where N-body problems are seen in the guise of galaxy dynamics, star cluster evolution, dark matter distribution, etc. The use of tree-like data structures to solve the numerical N-body problem began in the mid-1980s. This avenue of attack implements a "divide and conquer" strategy to solve the problem [2–5] and effectively reduces the computational work of the N-body problem from $O(N^2)$ to $O(N \log N)$. Fast multipole methods [6, 7]

[1] To whom correspondence should be addressed, at Laboratory for Computational Physics, Code 6410, Naval Research Laboratory, Washington, DC 20375. E-mail: waltz@lcp.nrl.navy.mil.

0021-9991/02 $35.00

also have enjoyed a certain degree of popularity. Tree algorithms have been applied to plasma and hydrodynamic modeling.

The simulation of N-body problems typically involves two classes of physical phenomena: those governed by long-range forces and those governed by short-range forces. Perhaps the best examples of these two extremes are gravity and gas dynamics. In the gravitational problem, the range of influence of particles is over the entire computational domain: every particle affects every other particle. For simulations of gas dynamics (inviscid compressible flow), only particles which are spatially local influence a given particle. Each particle in a gas dynamics simulation has a limited domain of influence which is much smaller than the entire computational domain. The differences between these classes of simulations lead to different computational goals in the creation of N-body algorithms.

For gravitational forces, the computational goal is to reduce the work as much as possible with a controllable amount of error. This task is a *long-range* search. In gravitational tree codes, for example, the nodes of the tree are used to represent the properties of all particles within the spatial region enclosed by the node. If the relative size of a node is small compared to its distance from a given test particle, the nodal properties (mass, center of mass, and higher order moments), rather than the properties of each individual particle contained within the node, are used to compute the force on the test particle. This approximation can be made if the size-to-distance ratio of the node is small enough to limit the error to some specified amount. If the error criterion cannot be met, the tree must be descended further.

For meshless simulations of gas dynamics phenomena, the most common approach is smoothed particle hydrodynamics (SPH). In this case, the computational goal is to identify as rapidly as possible all particles within a specified distance from the test particle. This task is a *short-range* search. The complete set of particles which fall within the region of influence of a given test particle must be obtained before the fluid variables can be updated.

Despite the fact that the long- and short-range problems represent different types of physical phenomena, there are many circumstances in which both gas dynamics and gravity are needed in the same simulation. In astronomy, the use of the tree for gravity calculations preceded its use to find the neighbors of SPH particles. However, many simulations of galaxy dynamics, cosmology, and binary star dynamics now use both gravity and SPH trees ([8] represents one of the first such approaches; see [9] for an example SPH trees).

This paper focuses on two tree algorithms as applied to long- and short-range searches in the context of N-body simulation: balanced binary (hereafter binary) trees and Barnes–Hut (BH) trees. When Hernquist successfully vectorized the Barnes–Hut algorithm [3] and explored its properties, spatial oct-trees became the *de facto* method for gravitational calculation. Since Hernquist's seminal papers, most of the attention in tree codes has focused on vector and parallel implementations of the Barnes–Hut algorithm [10], as well as methods for error control [11]. Similar (but fewer) studies have been performed for binary trees [12].

Despite the attention given to the individual algorithms, a systematic comparison of the relative merits of BH and binary trees has not appeared in the relevant literature. Previous work has compared the performance of BH trees with that of nearest-neighbor trees [13] and the fast multipole method [14]. Anderson explored the asymptotic behavior of the binary

and BH trees but did not numerically examine the performance differences between the two approaches on typical problems [5].

Thus, the purpose of this paper is to present a performance comparison of the BH and binary trees for both long- and short-range forces. In carrying out this task, a very general computational vehicle which permits controlled comparison of the two tree algorithms was developed. Due to the extensive use of tree-independent procedures, the comparisons are not influenced by implementation; only a small portion of the tree construction algorithm must be separately coded for each tree. For gravitational forces, the search has been implemented so that the error can be specified *a priori*. Therefore both trees are guaranteed to have identical error bounds, as is required for a truly fair comparison.

The remainder of this paper is organized as follows. Section 2 describes the methodology, design, structure, implementation, test, validation, and optimization of the two tree codes. Section 3 provides an overview of the test cases, and Section 4 presents the results of the computational experiments. Conclusions regarding the advantages and disadvantages of each approach are presented in Section 5, along with insights into the appropriate regimes of applicability of each method.

2. METHODS

2.1. Tree Construction

The original papers describing the binary and Barnes–Hut trees describe basic methods for their construction. In this work a tree construction algorithm which is independent of the tree type has been implemented. This algorithm is based on a simple recursive subdivision technique [15] and distinguishes between the two trees by the number of daughters into which each parent is divided. For the BH tree, the parent node is divided into eight regions (usually cubic) of equal volumes. The division point is located at the geometric center of the parent node. For the binary tree, the parent node must be divided along its largest dimension such that an equal number of particles are present on each side of the division.

For either tree, the information stored with each node consists of the following:

- mass and center-of-mass coordinates
- quadrupole and higher order moments
- geometrical center
- size of the node
- the daughters or particles contained in the node

The calculation of mass, center of mass, and higher order moments is implemented as separate, tree-independent procedures.

Previous studies of tree data structures have suggested that the BH tree is asymptotically superior to binary trees for long-range gravity calculations [5]. Through the use of regular spatial divisions, the higher order multipole moments are bounded. For the binary tree, a single distant outlier can cause pathological behavior because the higher order moments converge so slowly.

A modified binary tree also was examined. In the modified binary tree, two levels of nodes are skipped so that a typical node has eight daughters. Each of these nodes was created through orthogonal recursive bisection, but the branching ratio was changed to be similar to that of the BH trees. This feature allows one to examine how the branching ratio, rather than the underlying tree structure, impacts performance.

2.2. Tree Walking for Long-Range Forces

Originally, the standard "B-mac" θ criteria for the binary and BH trees were tested. However, since the nodes for binary trees can become very elongated, the higher order multipole moments can create large errors which are not well represented by these walks. Large differences in the force accuracies between binary and BH trees at the same θ value were found to exist.

In order to draw fair conclusions about the relative performance differences of binary and BH trees for long-range forces, we have adopted the methods developed by Salmon and Warren to place upper limits on the maximum force error in a tree [11]. This algorithm works by creating a priority queue for the force errors. For each node traversed, the force is calculated for a particle and the node. At the same time, the estimated error for the force calculation is found through the use of higher order moments to bound the truncation error of the multipole expansion. The forces and relative errors are added to the priority queue such that the node with the highest error is at the top of the queue. The total error is then estimated as a sum of the errors for each node in the queue. If the the total error is small enough, the search terminates. If the total error is too large, the node with the worst error is deleted from the priority queue and its daughter nodes are inserted into the queue.

The Warren–Salmon approach is a very general procedure which allows bounds to be placed on either absolute or relative errors and the total estimated error to be calculated in a number of ways. For these tests, the acceptance limit was set to be some fraction of the force calculated during the walk. For example, a tolerance of 0.01 allowed the total estimated error to be 1% of the total force from the walk. The total error was calculated as a sum of the absolute values of the errors in the priority queue. When the Warren–Salmon walk is implemented, the time required for the calculation is dominated by the intermediate force and error calculations. The final interaction list becomes the final set of nodes within the priority queue, but all of the parents of these nodes were at some point included.

The walk algorithm was implemented as a tree-independent subroutine. This approach eliminates the possibility of coding differences which might affect measured performance. In both cases, the multipole moments calculated during the build phase are used to estimate the truncation errors. Furthermore, the test results presented in this paper are, to at least the first order, independent of any parallelization or temporal domain splitting. At the core of all these codes, particles walk one or more trees. Although a multiple timestep routine can change the number of times an individual particle needs to walk a tree, and a parallel implementation can effectively change the numbers of trees walked, the underlying efficiency of the tree structure will play a key role in either of these optimizations.

In parallel tree codes, domain decomposition is usually accomplished with either an orthogonal recursive bisection [16] or a hashed oct-tree [17]. After the spatially local set of particles have been placed on each node of the computer, local trees are constructed and information is exchanged between computer nodes. In the end, the efficency of the algorithm will be directly affected by the number of local and external nodes needed to complete all the walks on each computer node. If the number of nodes needed to form a "locally essential tree" increases because of the tree structure, both the communications and the walk time will increase. Numerical experiments have confirmed that the results presented for the binary and BH performance are nearly the same in parallel implementations, since the walk time dominates the communication.

For multiple-timestep algorithms, the number of walks needed will be exactly the same for both binary and BH trees. For particles with rapidly changing forces and a shorter timestep, one might expect a somewhat longer interaction list than for particles with slow timesteps. However, no significant additional performance differences between binary and BH trees have been observed when a multiple-timestep algorithm is used.

2.3. Tree Walking for Short-Range Forces

Short-range searches using a tree code are used throughout computational physics. As mentioned before, smoothed particle hydrodynamics uses a local list of particles to update density, momentum, and energy. However, many problems in computational geometry and molecular dynamics use similar algorithms [18, 19].

Performance differences in short-range searches are substantially simpler to measure, since the underlying operation is essentially an integer count. The list of particles within a certain distance from a given test particle must be exactly the same regardless of which tree is used. The answer is uniquely specified by the search radius, and hence the issue of error limits is completely irrelevant. The difference in the performance of binary and BH trees will be characterized by the number of nodes processed in the formation of the list.

As in the long-range search, a tree-independent procedure is used to perform short-range searches. The underlying approach is to filter out nodes with a fast check on the bounding dimensions of the node. If a node is found to intersect the search radius, the daughters of the node must be examined. Note that situations can arise where a node intersects the search radius, but the particles within that node lie outside the search radius. Therefore, at some level the interparticle distances must be examined directly.

Obviously, the overall speed of the procedure will rely heavily on the specific algorithmic details used to process nodes. This procedure, however, is independent of the type of tree used. The relative performance between the two trees can be captured simply by counting the number of nodes that are processed during the search.

3. ANALYSIS

3.1. Overview of Test Cases

Two test cases were used to examine the relative performance of the BH and binary trees. These cases were

1. Uniform average density (UAD) within a cubic region of space.
2. Lowered isothermal sphere (LIS), i.e., a centrally concentrated star cluster.

Note that the UAD test case is more representative of periodic systems, while the LIS test case is more representative of astrophysical applications.

The number of particles N_P ranged from 2^{13} to 2^{15}. The upper limit was chosen based on two observations. First, for calculations significantly larger than 10^5, parallel architectures are typically used with 10^4–10^5 particles per processor. Second, our tests were intended to be limited to algorithmic performance, which can be examined with moderately sized data sets.

Each test case was evaluated for a single time step. For algorithmic performance, a single time step is sufficient. In the actual code an individual timestepping scheme is used for each particle, but this feature is not relevant to the performance measures examined in this study.

3.2. Uniform Average Density

The first case in the study consists of a uniform average density of particles within a cube. Particles for this case were distributed with a pseudo random number generator with uniform iterates. For the BH algorithm, the average depth of the tree should be close to $\log_8 N_P$, since the spatial distribution of the particles will match closely the distribution of cells within the tree. The binary tree also should have regularly shaped nodes. After each three successive bisections, one would expect the shape of the cells to be almost cubic because of the uniform distribution. This makes this particular test case very similar for both binary and BH trees.

3.3. Lowered Isothermal Sphere

The second test case in this study consists of particles distributed according to a lowered isothermal sphere or a King model. King models are generated with a distribution function of the form

$$f_K(\varepsilon) = \begin{cases} \rho_1 (2\pi\sigma^2)^{3/2} (e^{\varepsilon/\sigma^2} - 1) & \varepsilon < 0 \\ 0 & \varepsilon \geq 0, \end{cases} \tag{1}$$

where ε is the relative energy of a star and σ is the velocity dispersion in this distribution. The density function for this distribution is given by

$$\rho_K(\Psi) = \rho_1 \left[e^{\Psi/\sigma^2} \operatorname{erf}(\sqrt{\Psi}/\sigma) - \sqrt{\frac{4\Psi}{\pi\sigma^2}} \left(1 + \frac{2\Psi}{3\sigma^2} \right) \right], \tag{2}$$

where Ψ is the relative potential energy in this distribution. This function fits the distribution of globular clusters and elliptical galaxies well and has many similarities to Plummer distributions. Details of King model distributions are discussed by Binney and Tremaine [20].

Particles in King models are distributed with random velocities and angles and possess a radially decreasing density distribution with a finite truncation radius. The density ratio between the inner core and the outer halo in King models is characterized by the value of Ψ/σ^2. In this test case, we set $\Psi/\sigma^2 = 3$. Since there is a radial density gradient in the distribution, this case is less ideal than the UAD case for both the BH and binary algorithms. This case, however, is not particularly pathological for either.

3.4. Performance Metrics

The tree walk itself is composed of two distinct parts. First, the tree must be searched to generate the interaction list. Since the search algorithm is identical for both trees, the number of node evaluations required to generate the interaction list is a measure of the speed with which the tree can be searched. Since this measure depends on the shape of the tree, some dependence on both particle distribution and N_P is expected. The second part of the walk is the loop over the interaction list to calculate and sum the forces. The time required to compute the force contribution from a single entry in the interaction list will be the same for both trees.

For long-range forces such as gravity, the length of the interaction list is not, in general, expected to be the same for both trees. For this reason one tree may require significantly

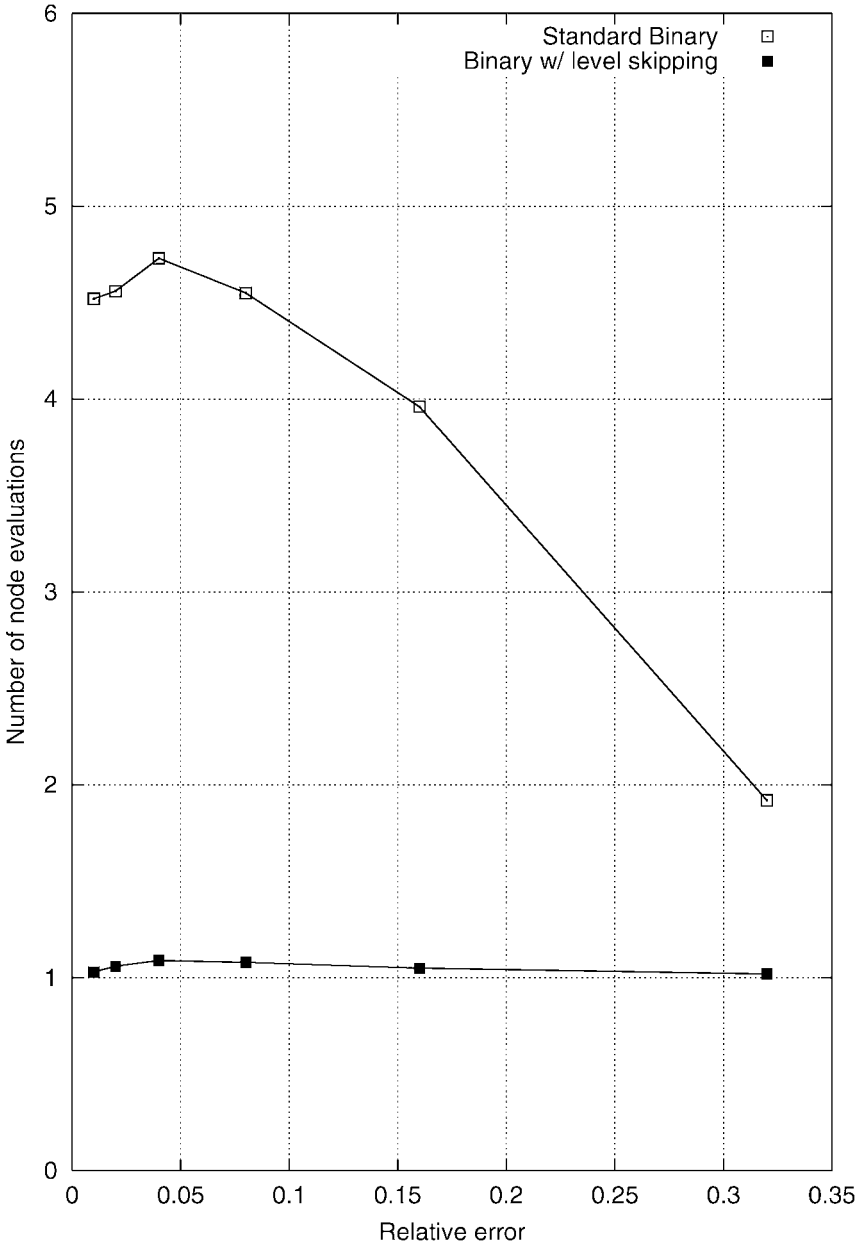

FIG. 1. N_N as a function of error, UAD test case.

more force calculations than the other. Differences in the length of the interaction list will contribute directly to measured differences in performance, as will the number of node evaluations required to generate the interaction list. For SPH, the interaction list is an integer count which must be the same for both trees. The length of the interaction list will have no effect on the relative performance of the two trees. Only the number of node evaluations required to generate the interaction list will differ between the two trees.

In view of the above considerations, the primary performance metric presented is N_N, defined to be the average number of nodes processed per particle during the search.

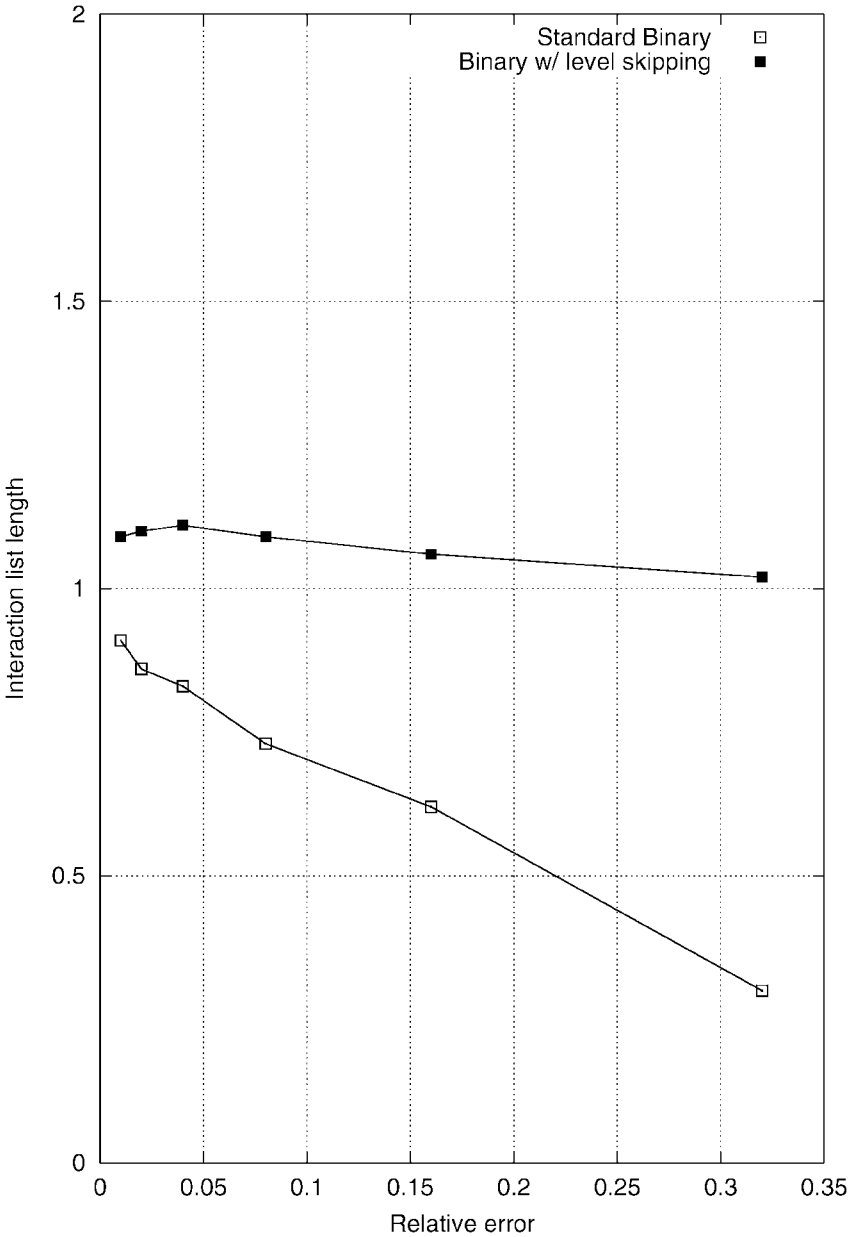

FIG. 2. N_I as a function of error, UAD test case.

For long-range gravitational searches, an additional metric is presented: N_I, defined to be the average interaction list length per particle. For both of these metrics, a smaller measure indicates better performance. All measurements are normalized to those of the BH tree. Therefore, for each test case one set of metrics is presented which directly expresses the performance of the binary tree relative to the BH tree. The results obtained with the level-skipping binary tree are denoted by N'_N and N'_I. Since N_N and N_I are not timing measurements, they are independent of machine type or programming language.

4. RESULTS

The results for $N_P = 2^{14}$ are shown in Figs. 1–3 for the UAD test case and in Figs. 4–6 for the LIS test case. The results show N_N and N_I as a function of error for the long-range forces, and N_N as a function of search radius for the short-range forces. The results are normalized to those of the BH tree so that a value less than one indicates that the binary tree is faster, and vice versa. The results for the two different test cases do differ slightly, as one would expect, but the general trends are the same. No change in results in relative

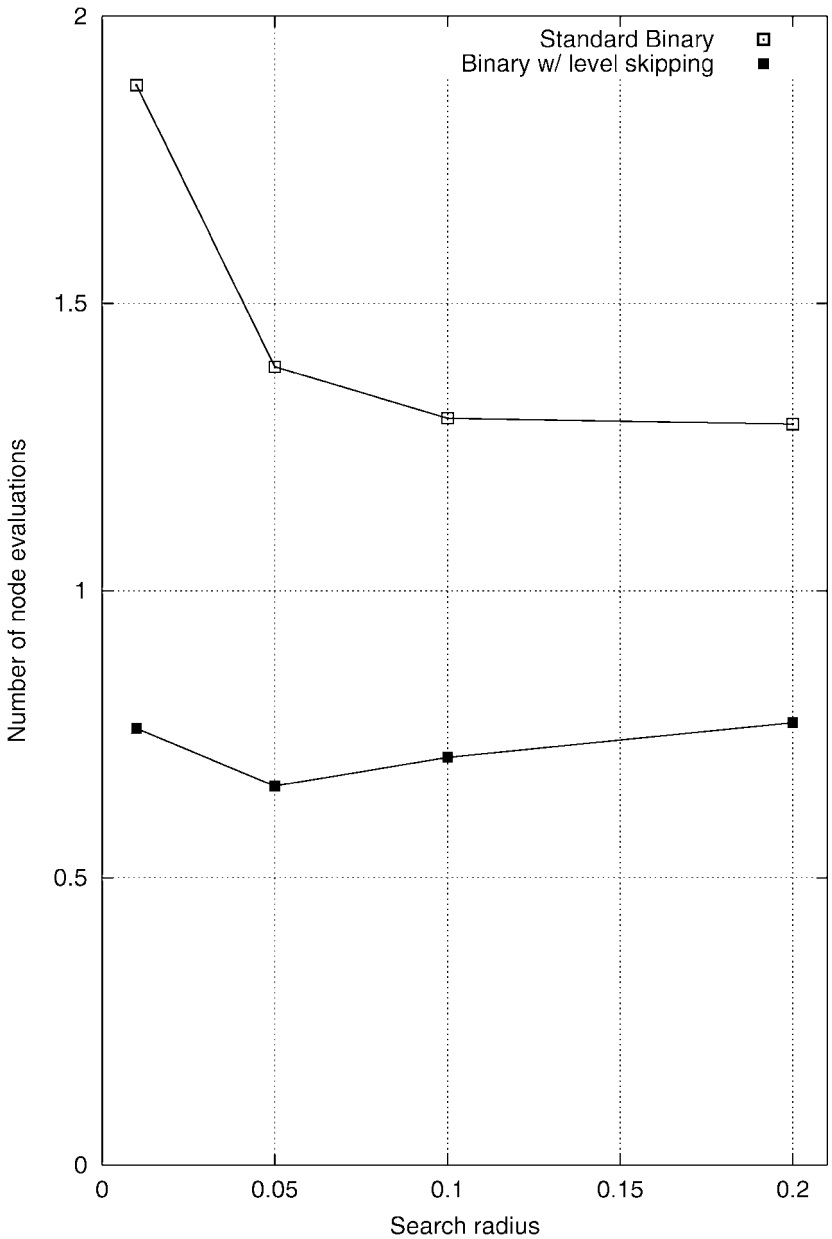

FIG. 3. N_N as a function of search radius, UAD test case.

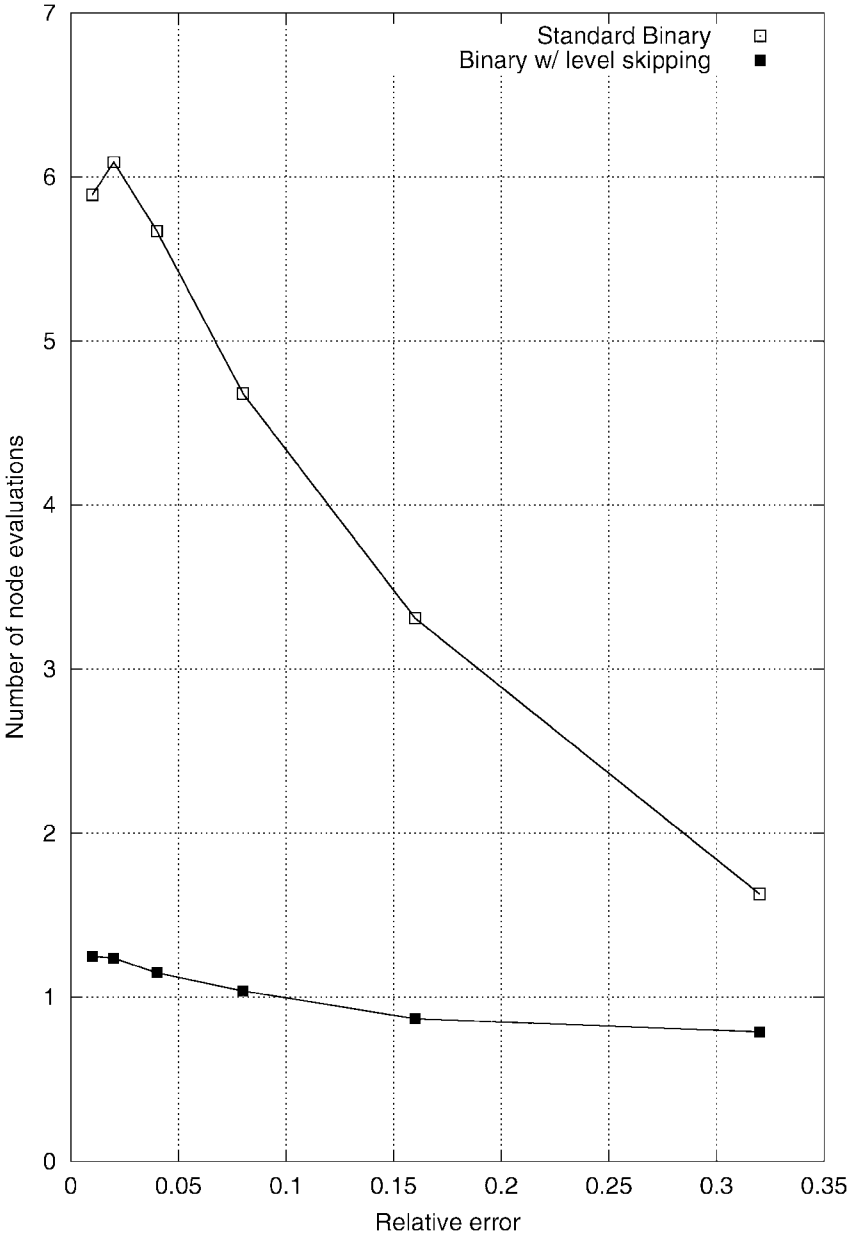

FIG. 4. N_N as a function of error, LIS test case.

performance was observed for either $N_P = 2^{13}$ or $N_P = 2^{15}$. Therefore these results are not presented.

The long-range results show that the BH tree is more efficient than the binary tree for gravitational calculations. With the level-skipping feature enabled in the binary tree, however, the performance differences drop to a few percent. Although the test geometries are somewhat typical for N-body problems, extreme outliers are not present in the particle distribution. If such particles had been included, the performance difference between BH and binary trees would widen.

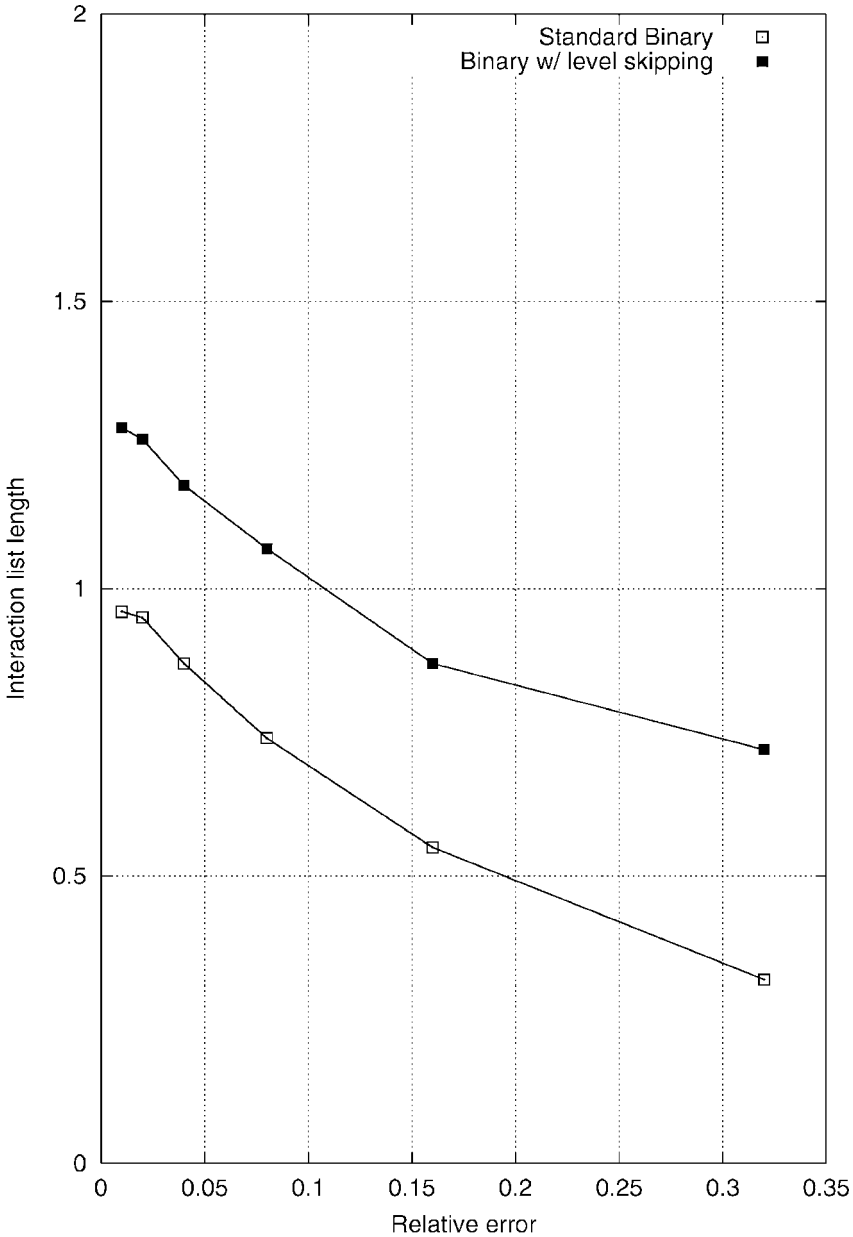

FIG. 5. N_I as a function of error, LIS test case.

One interesting observation is that the final interaction list of the binary tree is actually shorter than the final list for the BH tree. Because the binary tree branches to only two daughters, the final list size does not increase as rapidly as for the BH tree. However, the number of intermediate gravitational force and error calculations for the BH tree is substantially smaller for the binary tree. Therefore the BH tree is still more efficient. With level-skipping enabled, the interaction list of the binary tree exceeds that of either the standard binary tree or the BH tree. However, because N'_N is very close to 1, the overall efficiency of the binary tree with level-skipping is similar to that of the BH tree in the absence of pathological cases.

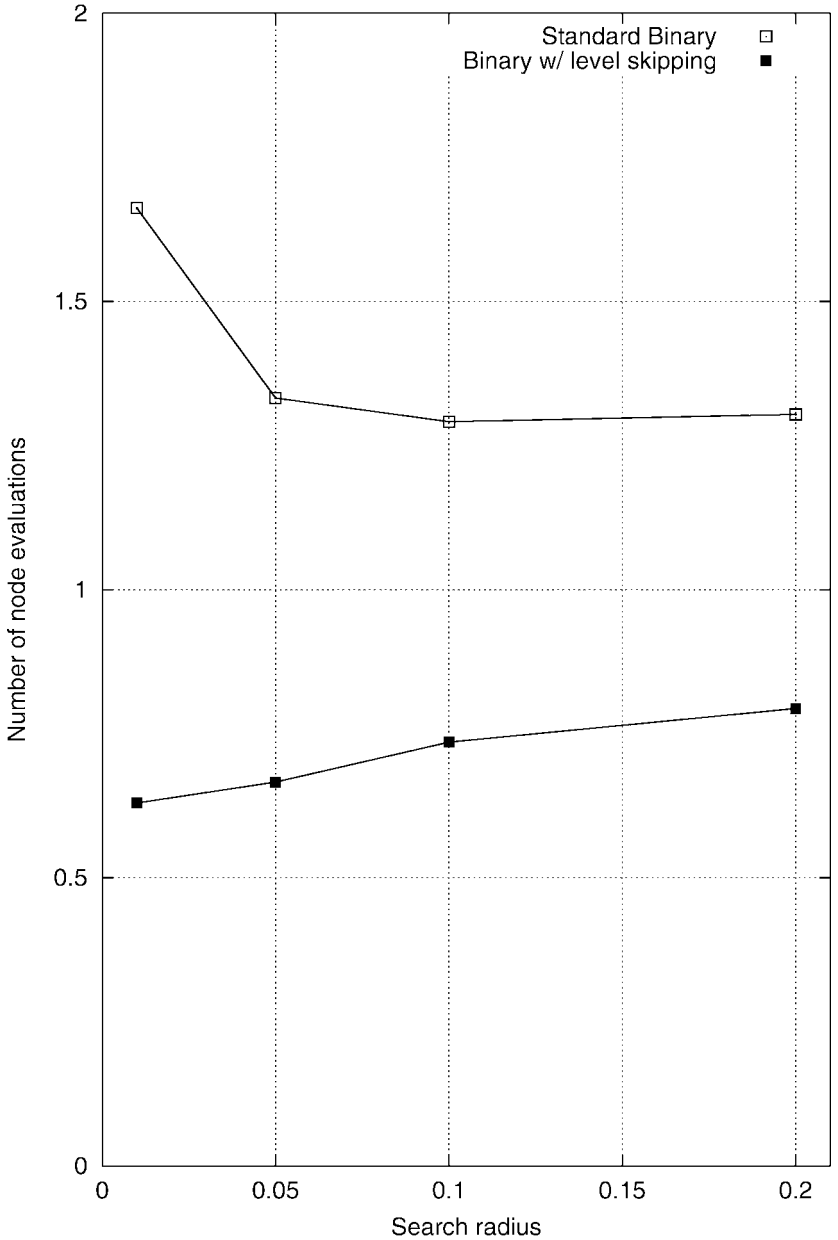

FIG. 6. N_N as a function of search radius, LIS test case.

The short-range results indicate that the relative performance differences between the binary and BH tree are much smaller in this scenario. When level-skipping is enabled, the binary tree becomes approximately 25–30% faster than the BH tree. The final interaction list has no meaning in these tests since, for all three algorithms, the list is exactly the same. The values for N_N are very similar for the two test cases: although the global particle distributions are different in the two cases, the local distributions seen by any particular particle will not vary greatly. Note that neither extreme outliers nor the length of the interaction list has any bearing on these results.

5. CONCLUSIONS

A performance comparison of Barnes–Hut and balanced binary trees was performed. This comparison was performed for both long- and short-range forces and with a tree-independent implementation. The performance metrics used to compare the performance were the number of node evaluations required to build the interaction list and, for long-range forces, the length of the interaction list itself. These metrics are well correlated to execution time for the tree walk and force summations, respectively. Due to the use of integer metrics rather than timings, the results are independent of computing platform or language. The comparison also is, to the first order, independent of any parallelization or multiple-timestep schemes.

For long-range forces, an explicit error control was used. In contrast to simpler θ-criterion methods, the error control ensures that both trees compute the "correct" answer and that the comparison is truly fair. The results of this test indicate that the BH tree outperforms the standard binary tree. The speed advantage of the BH tree varies with relative error but, generally speaking, the BH tree is several times faster than the standard binary tree. This behavior is observed with different particle numbers and distributions. However, a modified binary tree which makes use of a level-skipping procedure is competitive with the BH tree. The relative performance differences are on the order of a few percent. Note that the performance measurements for the long-range forces do not necessarily hold in the absence of a strict method for error control.

The trade-off associated with the modified binary tree is that although the number of node evaluations decreases to about that of the BH tree, the length of the interaction list increases (as one would expect). For the specific type of walk used in this study, the interaction list length is somewhat irrelevant because the forces (and the errors) are computed during the walk procedure itself rather than as a separate procedure. However, this may not always be the case. Note that this trade-off only exists for long-range forces.

For short-range searches, the desired result is a list of particles which lie within a certain distance from a given test particle. Since the list results from a counting operation, error control is irrelevant and the comparison is by default a fair comparison. The results of this test indicate that the relative performance differences between the binary and BH trees are much smaller than in the long-range case. The binary tree generally requires one to two times as many node evaluations as the BH tree. When the modified binary tree is used, however, the situation is reversed, and the BH tree becomes approximately 25–30% slower.

This last result has potential applications beyond astrophysical N-body work. For example, molecular dynamics problems which utilize Lennard–Jones (and similar) potentials stand to benefit from the use of the modified binary tree. Many problems in computational geometry also require rapid determination of nearest-neighbor lists. Note that in some simulations, the search time might constitute a small percentage of the overall computing expense. In these cases the impact of the modified binary tree may be less dramatic.

One possibility for future research is to implement a variable level-skipping procedure. An optimal value which provides for further performance improvements may exist. Investigation of average force error and how it compares with the specified error tolerance also may be worthwhile. In practice, one might be able to achieve low levels of force error with a relatively high error tolerance.

REFERENCES

1. H. Goldstein, *Classical Mechanics* (Addison–Wesley, Reading, MA, 1980).
2. A. W. Appel, An efficient program for many-body simulation, *SIAM J. Sci. Stat. Comput.* **6**, 1 (1985).
3. J. Barnes and P. Hut, An hierarchical $O(N \log N)$ force-calculation algorithm, *Nature* **324**, 446 (1986).
4. W. H. Press, Techniques and tricks for N-body simulation, in *The Use of Supercomputers in Stellar Dynamics*, edited by P. Hut and S. McMillan (Springer-Verlag, New York, 1986), p. 184.
5. R. J. Anderson, Tree data structures for N-body simulation, *SIAM J. Comput.* **28**, 6 (1999).
6. L. Greengard and V. Rokhlin, A fast algorithm for particle simulations, *J. Comput. Phys.* **73**, 325 (1987).
7. L. Greengard, The numerical solution of the N-body problem, *Comput. Phys.* **4**, 142 (1990).
8. L. Hernquist and N. Katz, TREESPH—A unification of SPH with the hierarchical tree method, *Ap. J. Suppl.* **70**, 419 (1989).
9. W. Benz, Applications of smooth particle hydrodynamics (SPH) to astrophysical problems, *Comput. Phys. Commun.* **48**, 97 (1988).
10. E. Athanssoula, A. Bosma, J.-C. Lambert, and J. Makino, Performance and accuracy of a GRAPE-3 system for collisionless N-body simulations, *Mon. Not. R. Astron. Soc.* **293**, 369 (1998).
11. J. K. Salmon and M. S. Warren, Skeletons from the treecode closet, *J. Comput. Phys.* **111**, 136 (1994).
12. K. Olson and J. Dorband, An implementation of a tree code on a SIMD parallel computer, *Ap. J. Suppl.* **94**, 117 (1994).
13. J. Makino, A comparison of two different tree algorithms, *J. Comput. Phys.* **88**, 393 (1990).
14. R. Capuzzo-Dolcetta and P. Miocchi, A comparison between the fast multipole algorithm and the tree-code to evaluate gravitational forces in 3-D, *J. Comput. Phys.* **143**, 29 (1998).
15. T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, MA, 1990).
16. J. Dubinski, A parallel tree code, *New Astron.* **1**, 2 (1996).
17. J. K. Salmon and M. S. Warren , A parallel hashed oct-tree N-body algorithm in, *Proc. Supercomputing 1993* (1993).
18. P. Gibbon and S. Pfalzner, *Many-Body Tree Methods in Physics* (Cambridge Univ. Press, New York, 1996).
19. M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids* (Oxford Univ. Press, Oxford, 1990).
20. J. Binney and S. Tremaine, *Galactic Dynamics* (Princeton Univ. Press, Princeton, NJ, 1987).