Parallel Methods for Astrophysical Simulation

Brendan Mumey
brendan@cs.washington.edu
Department of Computer Science
University of Washington, Seattle, WA 98195

14 January 1994

Abstract

A central question in astrophysics is "Why is matter in the universe distributed the way it is and how did get that way?" In particular, when we look at images of space we see long filamentous strands of matter and regions which are essentially void. As analytic methods fail for solving for the dynamics of the universe (modeled as a finite set of mutually attracting particles), computer simulations are relied on to study the evolution of structure in the universe. There are two separate problems to solve. The first is performing the simulation efficiently, the second is collecting relevant information along the way. The information typically collected is a description of the "clustering" present at different time slices. As it seems necessary to use a large number of particles (perhaps 10^7 or more) in order to capture scale phenomena accurately, both problems are computationally daunting. We are attempting to improve the best known solutions for these two problems and will describe our progress to date as well as our future expectations.

1 Introduction

We are interested in improving two aspects of astrophysical simulation. The first aspect is that of performing large scale particle simulations, the second is analysing the data at a given time slice to extract information on how "clustered" the points are. Because of the large number of particles required to make such simulations realistic (10^7 or more), the only feasible approach is to use large parallel machines. We are developing new parallel algorithms for both of these problems and are interested in implementing them on large parallel machines. In the next two sections we discuss our work so far on these problems and in the final section we outline where it is going and our proposed implementations.

2 Particle Simulation

The n-body problem, as the particle simulation problem is commonly known, is: given a set of n points in space, calculate the force on each particle due to the combined gravitational fields of the other particles. Once each force is determined, the system can be "integrated" for a short time step to determine the new particle positions. This process is repeated many times to simulate the behavior of the system over time. The naive approach is to calculate the attraction of each pair of particles present, resulting in an $O(n^2)$ algorithm. In the mid-1980s, dramatic improvements were made over this simple approach, yielding $O(n \lg n)$ algorithms [1, 2] and even $O(n)$ algorithms [4], although the latter seem to be impractical due to large constants. Common to all of these faster methods are the fact they rely on approximating the force felt by a particle due to far-away clump of particles.

One class of $O(n \lg n)$ methods which have been particularly successful in practise are *treecodes*. As the name implies, the algorithms use a tree built on the point set as the central data structure. The most prevalent kind of tree has been the *oct tree* which is a tree built by finding a box which contains all the points and dividing it into equally sized octants. Each octant is recursively divided if it contains more than one point. To evaluate the force on a given particle, the tree is traversed from top to bottom. If the box associated with a given node meets an *accuracy criterion* then the force on the particle from all the points within the box is approximated by assuming all the particles are at their center of mass (higher order approximations are sometimes used). This involves only one force calculation. If the accuracy criterion is not met, then the box is *opened* and the algorithm continues recursively, descending to each of the node's children in turn. It can be shown for many distributions of points that the expected cost of finding the total force on one particle is $O(\lg n)$, thus yielding an $O(n \lg n)$ sequential method.

Another kind of tree which seems competitive with oct trees [5], is the *nearest neighbor tree*. The tree is constructed with the following rule: If two points are mutually nearest neighbors, they may be removed and combined to form a new point at their center of mass. Thus the tree constructed is binary. Nearest neighbor trees attempt to overcome a problem with oct trees which is that oct trees are somewhat oblivious to the data. Often a large clump of points will be broken up because it lies on a number of octant boundaries. This leads to the problem of "exploding galaxies" reported in [3]. Nearest neighbor trees use more of the underlying geometry in their construction and should be more immune to this condition. We are currently investigating some of the theoretical properties of nearest neighbor trees. In particular we think we can show an $O(\lg \frac{\Delta}{\delta} + \lg \frac{M}{m}$ height bound, where Δ (δ) is the largest (smallest) distance between a pair of points in the original point set and M (m) is the largest (smallest) mass present.

Particle simulation seems particularly apt for parallelization. Not only can all the force calculations be done completely in parallel, but there is a strong

degree of data locality as well. Most of the force a particle experiences is due to particles nearby and thus likely to exist on the same processor (assuming physical memory is distributed equally among the processors). At greater distances much more approximation is tolerated and thus the amount of information required is correspondingly much less. This will happen provided that initially the data is distributed in contiguous clumps to the processors and as the simulation progresses particles are redistributed as they migrate away from a given clump. There are lots of issues to explore about how the choice of tree interacts with data redistribution.

We are currently analysing some different parallel methods for constructing and maintaining nearest neighbor trees and hope to have a better understanding of their effectiveness soon.

3 Cluster Detection

As a simulation is run, matter coalesces into long filaments and clumps. In fact, this clustering may be happening at several different distance and time scales. Some means of extracting information about this clustering from the vast amount of data present in a given time slice is required. One approach taken is to use information contained in a *Euclidean Minimum Spanning Tree*, or EMST built on the Euclidean distance graph of the point set. The property of EMSTs which is applicable to cluster detection is the fact that they provide a convenient means of computing the *bottleneck distance* between pairs of points.

The bottleneck distance between two points is defined to be the minimum possible weight of the maximum weight edge in a path (in the distance graph) between them:

$$\delta_b(x,y) = \min_{\text{paths } P \text{ connecting } x \text{ and } y} \max_{\text{edges } e \in P} \text{length}(e)$$

We will say that two points x and y are τ-*clustered* if and only if $\delta_b(x,y) \leq \tau$. This defines an equivalence relation on the point set and so implicitly partitions it into sets of related points; clusters. A property of EMSTs is that if one simply deletes all the edges above the threshold value, the remaining forest of connected components will correspond exactly to the set of clusters according to the bottleneck definition.

Unfortunately the best known methods for constructing EMSTs in two, three, or more than three dimensions take respectively $O(n \lg n)$, $O((n \lg n)^{1.5})$, or $O(n^2)$ time [8, 9, 6] Because n is 10^7 or greater, the cost of finding the EMST fully and exactly is prohibitive. In light of this, it is natural to ask if a faster approximate method can be found. We say an approximate cluster finding algorithm is ϵ-*close* if:

1. $\delta_b(x,y) > (1+\epsilon)\tau \Rightarrow x$ and y are identified to be in different clusters

2. $\delta_b(x, y) \leq \tau \Rightarrow x$ and y are identified to be in the same cluster

There are methods for constructing approximations to the EMST in $O(n \lg n + \epsilon^{-d/2} \lg(\frac{1}{\epsilon})n)$ time [6], from which an ϵ-close cluster partition may be extracted (d is the dimensionality of the point set). However we do not need to fully construct an (approximate) EMST as we only use part of the information that it provides. We only need to know what the connected components are when the edges above the threshold are thrown away. We have developed a new linear time algorithm which finds an ϵ-close cluster structure for three dimensional point sets in $O([\lg(\frac{\Delta}{\epsilon\tau}) + \sqrt{\frac{1}{\epsilon}}(\lg(\frac{1}{\epsilon}))^{1.5}]n)$ time[1]. We report this algorithm in [12]. The algorithm is straightforward to parallelize and should yield near linear speed-up. We omit the details. Preliminary results on a uniprocessor have been very promising, and thus we expect parallel implementations to perform well too.

4 Proposed Work

As mentioned above, we are in the process of developing and implementing new algorithms for the problems of particle simulation and cluster detection. The machine used for the initial implementation is a Kendall Square KSR-1 multiprocessor which has a shared virtual memory architecture. We are beginning with cluster detection, as the algorithm development is furthest along, but expect to be implementing some trial algorithms for particle simulation shortly. The work on cluster detection is mine, but the particle simulation work is a collaboration with my advisor Richard Anderson.

References

[1] A. Appel. *An Efficient Program for Many-Body Simulation.* SIAM Journal of Scientific and Statistical Computing. January, 1985.

[2] J. Barnes and P. Hut. *A hierarchical $O(n \log n)$ force-calculation algorithm.* Nature, 324, 446, 1986.

[3] J. Salmon and M. Warren. *Skeletons from the Treecode Closet.* Submitted to Journal of Computational Physics, July 31, 1992.

[4] F. Zhao. *An $O(n)$ Algorithm for Three-dimensional N-body Simulations.* MIT Artificial Intelligence Laboratory. Technical Report 995.

[5] J. Makino. *Comparison of Two Different Tree Algorithms.* Journal of Computational Physics. 88: 393-408. 1990.

[1] The prefactor in front of n is somewhat messy; it may be possible to improve.

[6] P. Callahan and S. Kosaraju. *Faster Algorithms for Some Geometric Graph Problems in Higher Dimensions.* Proceedings of the 1993 ACM/SIAM Symposium on Discrete Algorithms. January, 1993. Austin, Texas.

[7] N. Katoh and K. Iwano. *Finding k Farthest Pairs and k Closest/Farthest Bichromatic Pairs for Points in the Plane.* Proceedings of the 8th ACM Symposium on Computational Geometry. 1992.

[8] A. Yao. *On constructing minimum spanning trees in k-dimensional spaces and related problems.* SIAM Journal on Computing. 11 : 721-736. 1982.

[9] P. Agarwal, H. Edelsbrunner, O. Schwarzkopf and E. Welzl. *Euclidean Minimum Spanning Trees and Bichromatic Closest Pairs.* Discrete and Computational Geometry 6:407-422. 1991.

[10] T. Cormen, C. Leiserson and R. Rivest. *Introduction to Algorithms.* McGraw-Hill. 1991.

[11] F. Preparata, M. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag. 1985.

[12] B. Mumey. *Fast Cluster Detection in Point Sets.* Submitted to the 1994 ACM Symposium on Computational Geometry.