An O(NlogN) Hypercube N-body Integrator

Mike Warren, John Salmon
California Institute of Technology, Pasadena, CA 91125

ABSTRACT

The gravitational N-body algorithm of Barnes and Hut [1] has been successfully implemented on a hypercube concurrent processor. The novel approach of their sequential algorithm has demonstrated itself to be well suited to hypercube architectures. The sequential code achieves $O(N \log N)$ speed by recursively dividing space into subcells, thereby creating a hierarchical grouping of particles. Computing interactions between these groups dramatically reduces the amount of communication between processors, as well as the number of force calculations. Parallelism is achieved through an irregular spatial grid decomposition. Since the decomposition topology is not simple, a general loosely synchronous communication routine has been developed. Operations are simplified if the conventional grey code decomposition is modified so that the bits are taken alternately from each cartesian dimension. A speedup of 180 has been achieved for a 500,000 particle two-dimensional calculation on 256 processors. A speedup of 65 has been obtained for a 64,000 particle three-dimensional calculation on 256 processors.

Background

A direct integration of the N-body problem requires $1/2N(N-1)$ force calculations between pairs of particles. This allows a precise description of the time evolution of the system, but in a computation time that grows as N^2. Another method in common use is to solve for the global potential field to the particles, and then propagate each particle in this field for a short time. The potential field method is faster, in that it requires only order $N \log N$ operations per timestep, but it entails a loss of short range accuracy.

Because of the widespread applicability of N-body calculations, the parallel formulations of these algorithms were among the first to be implemented on hypercube processors. [2, 3] The conventional hypercube decomposition for a direct integration is to form a 1-dimensional ring, and send a copy of each particle halfway around the ring, accumulating forces as it goes. From a purely parallel programming standpoint, this technique is highly efficient and capable of almost linear speedup as long as the number of objects assigned to each processor is large (greater than 50, say). Nevertheless, even parallel computers cannot cope with algorithms that require $O(N^2)$ time, and despite its high efficiency, it is still not a practical solution when N is in the range 10^4 to 10^6.

The potential method is also well suited to a hypercube topology. If a smoothed density function is calculated, the potential problem can be solved in Fourier space. The problem then becomes essentially one of computing a parallel Fourier transform. Efficient Fast Fourier Transforms, FFT's, have been implemented on hypercubes, and some astrophysical simulations have been run on Caltech's Mark II hypercubes employing this algorithm. [4, 5] The main disadvantage of this method is common to both the sequential and parallel implementations, namely, the loss of accuracy at length-scales approaching the size of the mesh used to model the potential.

The efficiency of the potential method and the accuracy of direct integration have been combined in the algorithm of Barnes and Hut[1]. By grouping together increasingly large groups of particles at increasingly large distances, direct summation of forces may be used while maintaining logarithmic growth. The method relies on the representation of the system as a hierarchical tree.

The root of the tree is a cubical cell large enough to contain the entire system. This root cell contains the mass and center-of-mass coordinates of the entire system, as well as pointers to eight daughter cells, which contain more detailed information. The daughter cells are subdivided similarly, until each cell contains at most one particle.

Once the tree is constructed, the force on a given particle may be determined by a recursive descent of the tree. Starting at the root, if the distance from the particle to the center-of-mass of the cell is less than the diameter of the cell divided by a fixed accuracy parameter θ, then the interaction is computed. Otherwise the current cell is opened, and each of its eight daughter cells is examined. The error introduced by treating all particles within a cell as a single super-particle at the center of mass depends on quadrupole and higher-order moments of the mass distribution within the cell, and may be rigorously determined. In practice, forces computed with an accuracy parameter as large as 1 are accurate to a few percent.

Decomposition

The parallel algorithm is an exact implementation of the Barnes and Hut code. No major changes were made to the sequential code; parallelism was achieved by the addition of spatial decomposition and communication routines. These routines utilize several of the functions already implemented in the sequential code to great advantage. A crystalline operating system (CrosIII) was used to manage the communication. The use of a fairly low level, loosely synchronous operating system is indicated because the algorithm requires a great deal of communication, which can only be handled efficiently at a low level[3].

The first step in the parallel program is a spatial decomposition. The particles are initially divided equally among the processors, without regard to spatial position. A coordinate to divide the particles into two parts is then determined. If a given particle is on the wrong side of this split, it is sent to the processor on the highest communication channel. Now each half of the particles may be divided again and traded on the next highest channel. This process is continued until a split over each channel has been made. At the end of this process each processor has its own spatially grouped set of particles. Depending on how the split coordinate is determined, several types of domain decomposition may be obtained. If the splittiing coordinate simply divides the cubical volume in half each time, a regular grid decomposition results. Although simple, the regular decomposition results in a highly unequal number of particles in each processor. An irregular decomposition with more nearly equal numbers of particles in each processor results if the particles are split along their mean positions. A third method of decomposition is to split along the median particle position, which results in the best balance of particles among the processors. The median, however, is difficult to compute exactly, and is usually not much different from the mean, so the benefits of using it as a splitting criterion are questionable. Nevertheless, a parallel median finding algorithm using a successive approximation technique has been developed. Figure 1 shows the processor boundaries obtained by using the center-of-mass splitting criterion on a system with two well separated concentrations of particles (galaxies). Also shown is the full hierarchical subdivision of space constructed by the sequential algorithm.

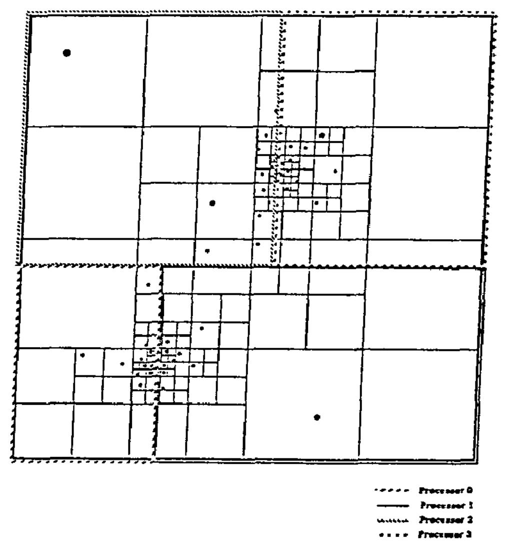

Figure 1. Decomposition of a system of particles on four processors.

Once the particles have been spatially subdivided they must be placed in a hierarchical tree. One's initial inclination would be to make the tree identical in all processors, necessitating a complete exchange of information throughout the machine, much as in the $O(N^2)$ algorithm. Fortunately, this is not necessary due to the way the forces are calculated from the tree. The heart of the algorithm lies in the fact that a given particle will only traverse part of the tree. In fact, a set of particles from a limited spatial domain will all traverse approximately the same parts of the tree, so it is only necessary to represent a portion of the whole tree in each processor. This allows for a dramatic reduction in communication, as well as storage requirements. The question of which data must be communicated may be elegantly solved by using precisely the tree-traversal routines in the original sequential algorithm and applying a different action to each node of the tree. Once each processor has a tree from the particles it controls, the tree is then traversed with a function almost identical to the force calculation routine. Instead of calculating a force when the desired level of the tree has been reached, the function sends the data to other processors. When the communication cycle is complete, each processor has a picture of the entire system, made up of its own particles and virtual particles of various sizes sent from other processors. A new hierarchical tree then made in each processor, containing both the real and virtual particles in the processor. The algorithm then proceeds exactly as in the sequential case. Figure 2 shows the same system as Figure 1, as represented in processor 0 after the communication cycle. Note that the galaxy in the upper right is represented by only six virtual particles.

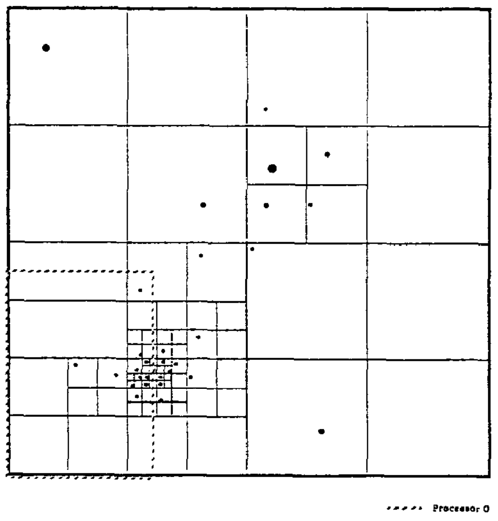

Figure 2. The system of Figure 1 as represented in processor 0.

Communication is accomplished by trading data on each channel in turn. Since the data is grouped spatially, the code is simplified if the conventional grey code is modified so the bits are taken alternately from each cartesian dimension. When this is done a loop over the channels will communicate alternately over each cartesian dimension. Because data dependence decreases with cartesian distance, data need only be addressed to neighboring processors in the hypercube topology, as long as the decomposition is regular. After each trade the data is assimilated into the tree, and may be sent again on the next channel. For irregular decompositions the procedure is not so simple. A processor may need to send data to another cartesian neighbor which is several channels away in the communication space. Since the data need not be entered into the trees of the intermediate processors, a general communication routine is needed to send data to any given processor. This type of loosely synchronous, long-distance communication system is known as a *crystal router* [3]. The conventional crystal router is unsuitable for this particular problem due to its inefficient use of memory, and a need for very large buffers. A new crystal router was developed which is especially suited to this application. By arranging messages in the proper order the communication may be done with a minimum of additional buffer space.

Performance

The performance of the parallel algorithm is quite good. The domain decomposition stage typically takes less than 3% of the computation time per timestep, even in the worst case of the first timestep where the data is spread at random. The time spent communicating the tree data for an accuracy parameter, θ, of 1 varies from 2% for small cube dimensions up to 50% for 256 processors. Two-dimensional calculations perform somewhat better than three-dimensional ones. The major cause of inefficiency remains load imbalance, even after an irregular domain decomposition based on dividing the particles about their mean positions. Currently under investigation is a decomposition about the median of the particles, which results in an equal number of particles in each processor.

The execution time per timestep for the sequential algorithm on a single Ncube node is very nearly $N \log_2 N \times 3$msec in two dimensions, and $N \log_2 N \times 6$msec in three dimensions. The 2D parallel algorithm on a hypercube of dimension eight (256 processors) can integrate 500,000 particles in 160 seconds per timestep. The time is attributed to various sources in Figure 3. The 3D program can integrate 64,000 particles in 100 seconds on 256 processors. Figures 4 and 5 show the efficiency the parallel implementation for several values of N and several sizes of hypercube.

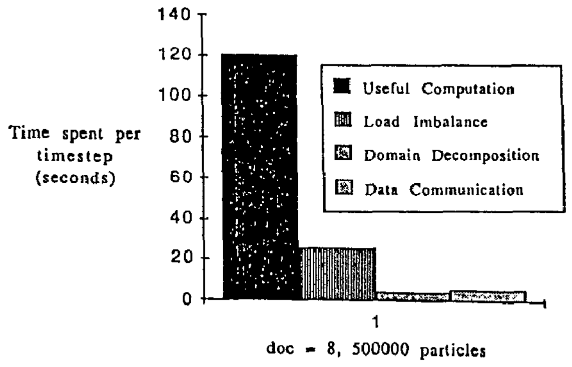

Figure 3. Breakdown of computation time (2 dim.)

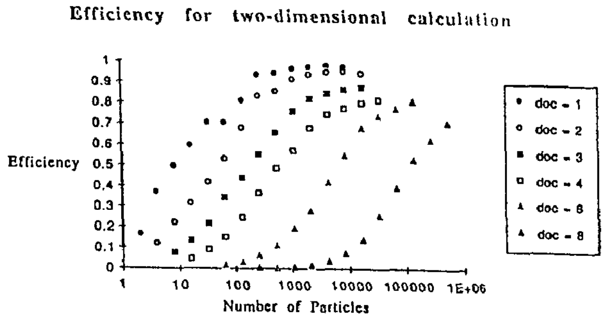

Figure 4. Efficiency of the 2D algorithm

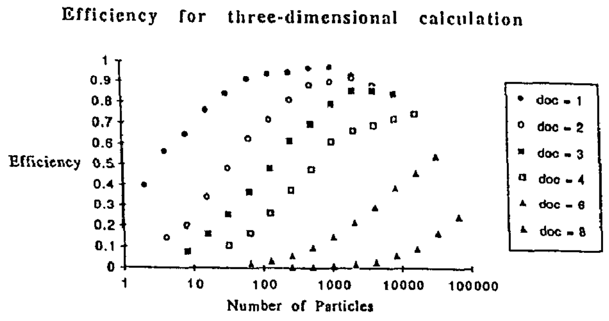

Figure 5. Efficiency of the 3D algorithm

Although the Barnes and Hut algorithm was not conceived with parallel computers in mind, it is remarkable that it still performs extremely well in parallel. The underlying structure of the sequential algorithm fits very well in the context of parallel programming. The crucial aspect of the algorithm is to reduce computation time by grouping particles in a hierarchical tree. A consequence is that the hierarchical structure is also very useful in reducing communication between processors and hence maintaining high efficiency in parallel. It is interesting that hierarchical structures have been "discovered" in numerous aspects of scientific computing in the last two decades. Hierarchical structures appear in multi-grid methods[6], the fast fourier transform[7], ray-tracing[8], renormalization group techniques[9] and other N-body algorithms [10] and in such non-scientific algorithms as quick-sort[11] and branch-and-bound techniques[12]. The same hierarchical structures that make these algorithms so attractive in sequential computation may prove to be the key to successful parallel computation as well.

Acknowledgments

This work was supported in part by Department of Energy Grant No. DE-FG03-85ER25009, the Program Manager of the Joint Tactical Fusion Office and the ESD division of the USAF as well as grants from IBM and SANDIA and a Shell Foundation Fellowship. We are indebted to Josh Barnes and Piet Hut for generously providing their C implementation of the sequential version of the algorithm.

References

1. J. Barnes & P. Hut, *A Hierarchical O(NlogN) Force-Calculation Algorithm*, Nature, 324, 446-449, 1986.
2. J. Salmon, *An Astrophysical N-body Simulation for Hypercubes*, Caltech Concurrent Computation report no. 78, 1984.
3. G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon & D. Walker, *Solving Problems on Concurrent Processors*, Prentice-Hall, Englewood Cliffs N.J., 1988.
4. J. Salmon & C. Hogan, *Correlation of QSO Absorption Lines in Universes Dominated by Cold Dark Matter*, Monthly Notices of the Royal Astronomical Society, 221, p. 93, 1986.
5. P. Quinn, J. Salmon & W. Zurek, *On the Structure of Galactic Haloes*, Nature, 322, p. 329, 1986.
6. W. Briggs, *A Multi-grid Tutorial*, SIAM, 1987.
7. J. Cooley & J. Tukey, *An Algorithm for the Machine Calculation of Complex Fourier Series*, Math. Comput., 19, p. 297, 1965.

8. S. Rubin & T. Whitted, *A 3-Dimensional Representation for Fast Rendering of Complex Scenes*, Computer Graphics, (Proc. SIGGRAPH 1980), 1980.

9. K. Wilson, & J. Kogut, *The Renormalization Group and the* ε Expansion, Physics Reports, **12**, p. 75, 1974.

10. L. Greengard & V. Rokhlin, *A Fast Algorithm for Particle Simulatiosn,* Yale University Computer Science Research Report YALEU/DCS/RR-459, 1986.

11. D. Knuth, *The Art of Computer Programming: vol. 3 Sorting and Searching*, Addison Wesley, Reading Mass., 1973.

12. C.H. Papadimitriou and K. Stieglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, New Jersey, 1982.