SIAM Conference on Parallel Processing, San Francisco, 1994

n-Body Simulations Using Message Passing Parallel Computers*

Ananth Y. Grama[†] Vipin Kumar[‡] Ahmed Sameh[§]

Abstract

In this paper, we present new parallel formulations of the Barnes-Hut method for n-body simulations on message passing computers. These parallel formulations partition the domain efficiently incurring minimal communication overhead. This is in contrast to existing schemes that are based on sorting a large number of keys or on the use of global data structures. The new formulations are augmented by alternate communication strategies which serve to minimize communication overhead. The impact of these communication strategies is experimentally studied. We report on experimental results obtained from an astrophysical simulation on an nCUBE2 parallel computer.

1 n-Body Simulation Techniques

The n-body problem simulates the behavior of n particles, each of which is influenced by every other particle during a time-step. An exact formulation of this problem therefore requires calculation of n^2 interactions between each pair of particles. Many approximate algorithms have been devised that reduce the complexity of this problem. Most of these algorithms are based on a hierarchical representation of the domain using a spatial tree data structure. The leaf nodes consist of aggregates of particles. Each node in the tree contains a series representation of the effect of the particles contained in the subtree rooted at the node. These representations are typically based on Taylor or Legendre polynomials. Interactions between nodes and particles are dictated by a multipole acceptance criteria (MAC). Different algorithms use different MAC. Selection of an appropriate MAC is critical to controlling the error in simulation. Methods in this class include those due to Appel, Barnes and Hut, and Greengard and Rokhlin (Fast Multipole Method). Even with the reduced complexity of these algorithms, the n-body problem takes a significant amount of time because of the large number of particles and time steps. Efficient parallel formulations therefore need to be developed.

The Barnes-Hut method is one of the most popular methods due to its simplicity. Although its computational complexity of $O(n \log n)$ is more than that of the Fast Multipole Method (FMM), which is $O(n)$[4], the associated constants are smaller for the Barnes-Hut method particularly for simulations in three dimensions. Furthermore, FMM uses a number of complicated data structures that make it difficult to program. However, FMM has proven error bounds unlike the Barnes-Hut method. Warren and Salmon [9] present a variant of the Barnes-Hut method that has a good worst-case error bound. In this paper, we describe parallel formulations of the Barnes-Hut method. The parallel formulations presented in this paper can also be applied to FMM with some modifications because the issues in parallelizing either of these algorithms are very similar.

2 Parallel Formulations of Barnes-Hut Method

Parallel formulations of the Barnes-Hut method involve partitioning the domain (or the tree) among various processors with the combined objectives of optimizing communication and balancing load. If particle densities are uniform across the domain, these objectives are easily met [1, 10, 5, 6, 3]. For irregular distributions, these objectives are hard to achieve because of the highly unstructured nature of both computation and communication. Singh et al. [7] and Warren and Salmon [9, 8] present schemes for irregular distributions that try to meet these objectives.

*This work was supported by IST/BMDO through Army Research Office contract DA/DAAH04-93-G-0080, and by the ARO under contract DAAL03-89-C-0038 with the University of Minnesota Army High Performance Computing Research Center; Access to the nCUBE2 at Sandia National Lab was provided via DeSRA consortium.

†Department of Computer Science, University of Minnesota, Minneapolis MN. (*ananth@cs.umn.edu*)

‡Department of Computer Science, University of Minnesota, Minneapolis MN. (*kumar@cs.umn.edu*)

§Department of Computer Science, University of Minnesota, Minneapolis MN. (*sameh@cs.umn.edu*)

The scheme of Warren and Salmon uses Morton ordering to hash the bodies into a table (Morton ordering is based on keys derived by interleaving the bits of the x and y coordinates of the particle). This table is sorted and assigned to processors in such a way that each processor gets equal computational load. The computational load associated with a particle in a particular iteration is used as a measure of the load in the next iteration. This is because the load is not likely to change significantly between subsequent iterations. Although it is possible to use techniques for sorting "almost sorted" lists, partitioning the particles among processors can become a source of overhead. The cost-zones scheme of Singh et al. uses the global Barnes-Hut tree data to partition the domain. Each node in the global tree keeps track of the load associated with the subtree rooted at the node. The partitioning technique starts at the root and traverses the tree in an in-order fashion assigning nodes to processors such that each processor has equal computational load. Both these partitioning schemes have excellent load balance and communication characteristics. However, the former has a potentially high sorting cost associated with it and the latter was believed to be suited only to shared address space machines because of its use of the global tree data structure. In [2] we show that these two schemes are very similar in terms of the resulting partitions and the communication overhead.

We now present two new partitioning techniques and alternate communication mechanisms and demonstrate good performance for formulations based on these.

2.1 Partitioning Techniques

2.1.1 Modular Scattered Decomposition (MSD)

In this technique, the domain of simulation is partitioned into sub-domains. The number of sub-domains (r) is greater than the number of processors (p). Each processor is assigned $k = r/p$ sub-domains. The value of k depends on the nature of the particle distribution. If the distribution is very irregular, k must be large so that the uneven parts of the domain can be partitioned and assigned to different processors; otherwise the resulting load at processors would be unbalanced. On the other hand, if the domain is not highly irregular, a smaller value of k would suffice. By selecting k appropriately, the dense parts of the domain (parts that have high concentration of work) are distributed among various processors and the resulting partition and mapping is adequately load balanced. For a two-dimensional simulation running on a d dimensional hypercube, sub-domain (i,j) is assigned to processor $(\text{gray}(i, d/2), \text{gray}(j, d/2))$. Here, $\text{gray}(p,q)$ represents the p^{th} entry in the gray-code table formed from q bits. A similar mapping that assigns neighboring subdomains to neighboring processors can be devised for a mesh.

2.1.2 Morton Ordering Based Scattered Decomposition (MOBSD)

Modular scattered decomposition relies on randomization for load balance. It is possible to combine the static partitioning of the domain into clusters with other techniques for balancing load in such a way that the resulting technique is better load balanced and maintains the communication characteristics of this scheme. One way of achieving this is to use Morton ordering (or Peano-Hilbert ordering) for assigning clusters to processors.

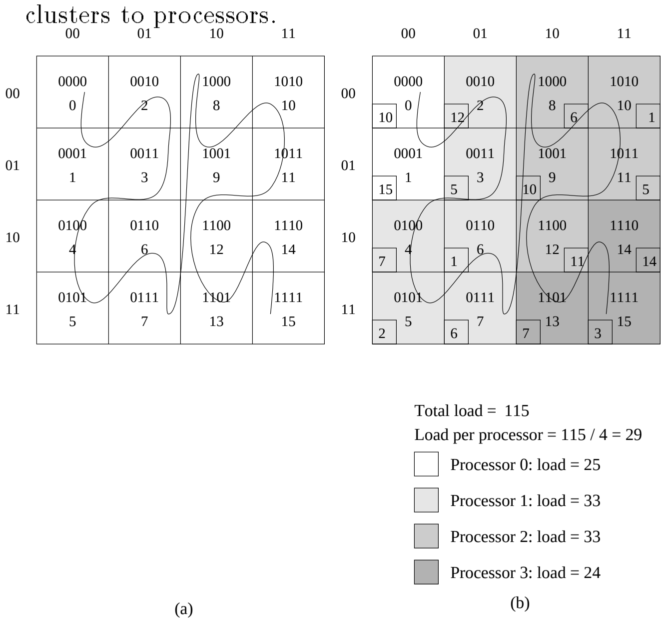

Figure 1: (a) Morton ordering of a domain decomposed into 16 clusters; (b) An illustration of cluster mapping using the improved scheme. The numbers in the inset represent loads in each cluster. The total load is determined by adding these loads. Each processor is assigned approximately equal load in accordance with its Morton ordering.

In this new formulation, a Morton ordering is constructed by using the cluster coordinates (instead of particle coordinates as was the case in the WS scheme). This is illustrated in Figure 1(a). The bits of the row and column are interleaved and the boxes are labeled by the Morton number. This ordering can be computed in advance and stored in a sorted list. After an iteration, a processor computes the load in each of its clusters. This load is entered into the sorted list. Note that it is not necessary to sort the list after each iteration. Once sorted, the ordering of the clusters does not change.

After each iteration, the load of each cluster at a processor is added and a global sum is computed. The global sum is divided by the number of processors to derive the desired load at each processor during the next iteration. Each processor compares the load in the current iteration with the desired load in the next iteration. If current load is less than the desired load then clusters are imported from the next processor in the Morton ordering. If this load is more, excess load is exported from the end of the list of clusters to the next processor. This data movement will not be significant since cluster loads are not expected to change drastically after each iteration.

2.2 Tree construction

Tree construction is simplified by two characteristics: all particles in a particular subdomain are assigned to a single processor; and, by selecting the number of partitions along any dimension of the domain to be a power of 2, each subdomain in fact corresponds to a potential node in the Barnes-Hut tree. Whether this node exists is determined by the number of particles in the sub-domain. Tree construction is performed in the following two steps: (a) local tree construction; (b) merging of local trees to form global tree. Local tree construction does not require any communication and can be performed by all processors independently. For each sub-domain assigned to a processor, it constructs the corresponding tree. The dimensions of the root node in the tree are those of the sub-domain and not the entire domain. Only the root nodes of local subtrees need to participate in the merge phase. In this phase, the four nodes forming the offsprings of a particular node are all sent to a designated processor. This processor combines the nodes. This process continues until the entire tree is constructed.

2.3 Communication Mechanisms

The top level nodes in the Barnes-Hut tree are accessed repeatedly by all processors. This indicates that these nodes should be made available globally to all the processors (and not on an 'on-demand' basis). This can be done by a single all-to-all broadcast. All nodes above a certain cut-off depth are broadcast to all processors. Forces acting on a particle are computed by accruing contributions of forces resulting from interactions between particles and nodes in the tree. Traditionally, the interactions are computed by the processor that holds the particle. The data corresponding to the nodes is communicated to the processor holding the particle on an 'on-demand' basis. We call these the owner-computes schemes. Such schemes have a disadvantage on message-passing computers because receiver-initiated communication is typically expensive on these machines. Furthermore, since the multipole acceptance criteria is evaluated on the fly, it is difficult to predict what nodes a particle interacts with a-priori. This often results in fine grain communication. An alternate approach is to communicate the particle data to the processor holding the tree data and computing the interaction there. We refer to this scheme as a receiver-computes scheme. The latter approach has advantages in that the communication becomes sender-initiated and coarse grained. Furthermore, if the entire subtree rooted at a particular node resides at a single processor, then all interaction resulting from nodes in the subtree can be computed at the cost of a single communication.

We use the receiver-computes communication strategy in conjunction with the two partitioning techniques. Conventional methods of evaluating load are based on computing particle loads in terms of number of tree nodes traversed by the particle, or the number of force computations. In our parallel formulations, this metric for characterizing load does not work. This is because if a particle interacts with a tree node, the computation is performed at the processor that contains the tree node (as against the processor that contains the particle, which is the case with other parallel formulations). Therefore, the load in our formulations is associated with a cluster and not with a particle. Specifically, if particle i requires r force computations to account for cluster k, then the corresponding load of cluster k is increased by r units.

3 Experimental Results

We implemented the two new partitioning strategies in conjunction with the receiver-computes communication scheme in the context of an astrophysical simulation for an nCUBE2 parallel computer. The performance of these schemes is evaluated for a number of simulation instances containing approximately 160K, 326K, 657K, and 1.2 million particles. The problem instances are named g_n where n is the number of particles in the simulation. For instance, g_1192768 contains 1192768 or approximately 1.2 million particles. Problem g_1192768 contains two gaussian distributions centered randomly in a 100×100 domain. All other problems contain a single gaussian distribution. The largest problem requires approximately 1.3×10^8 force computations. The size of the

No. of Processors → Problem	Force computations	Scheme	16	64	256
g_160535 ($\alpha = 0.67$)	2.2×10^7	MSD	179.74	65.53	
		MOBSD	132.37	51.02	
g_326214 ($\alpha = 1.0$)	2.1×10^7	MSD	167.449	62.70	22.57
		MOBSD	133.75	45.42	15.63
g_657499 ($\alpha = 1.0$)	4.6×10^7	MSD		114.75	31.06
		MOBSD		91.02	24.27
g_1192768 ($\alpha = 1.0$)	1.2×10^8	MSD		197.51	54.86
		MOBSD		163.96	45.17

Table 1: Runtimes (in seconds) of the receiver-computes scheme for modular scattered decomposition and Morton ordering based scattered decomposition.

problems attempted is limited by the fact that the available nCUBE2 computer only had 4 Megabytes of memory at each processor. In fact, for most of the problems, the limited amount of memory makes it impossible to run these problems on a single processor, and therefore speedup results cannot be reported.

Table 1 presents runtimes of various problems on different numbers of processors. The table demonstrates that the runtime decreases consistently with increasing number of processors. For example, on increasing the number of processors from 64 to 256 (a factor of 4) for problem g_1192768, the time reduces by a factor of 3.63 for the MOBSD scheme and by a factor of 3.6 for MSD. This shows that for larger problems, both schemes are scalable at least up to 256 processors. For smaller problems, the time reduces by a somewhat smaller factor. MSD has higher runtimes because of load imbalances.

Table 2 gives a breakup of the time taken by various phases of the two schemes for problems g_1192768 and g_326214 for $p = 256$. The time for local tree construction is very small for both schemes. The time taken by MOBSD is slightly more than MSD. This can be explained by the fact that in trying to balance load, MOBSD may have vastly different number of particles at individual processors. The tree merging cost is higher for MOBSD because unlike MSD in which all processors contained equal number of clusters, the number of clusters here may be very different. Furthermore, the systematic communication structure of MSD does not exist for MOBSD in the tree merge phase. The time for all-to-all broadcast is comparable for the two schemes. MOBSD spends lesser time in the force computation and tree traversal phase because of better load balance. This time also includes the time taken by the two all-to-all personalized communication operations embedded in the force computation routine. Finally, MSD spends no time in balancing load since load balance is implicit.

Problem → Phase	g_1192768 MSD	g_1192768 MOBSD	g_326214 MSD	g_326214 MOBSD
Local tree constr.	0.004	0.0065	0.0018	0.0023
Tree merging	0.061	0.79	0.022	0.24
All-to-all broadcast	0.40	0.39	0.30	0.28
Force computation & tree traversal	53.62	42.46	21.94	14.30
Load balancing	0	0.86**	0	0.61**
Total*	54.86	45.17	22.57	15.63

Table 2: Time taken (in seconds) by various phases of the parallel formulations for the MSD and MOBSD schemes for problems g_1192768 and g_326214 for $p = 256$. * The total times are not exactly equal to sum of individual phases. This is because of overheads incurred in other small segments of the code that set up required data structures. ** The load balancing cost does not include the time for sorting the Morton numbers corresponding to the cluster numbers. This time is not included since the cost is amortized over a large number of iterations.

4 Conclusions

In this paper, we presented two new parallel formulations of the Barnes-Hut method. These methods combine fast partitioning with efficient communication mechanisms to obtain good performance. We demonstrate the scalability of these formulations in the context of an astrophysical simulation on a 256 processor nCUBE2 parallel computer.

References

[1] J. A. Board, J. W. Causey, J. F. Leathrum, A. Windemuth, and K. Schulten. Accelerated molecular dynamics with the fast multipole algorithm. *Chem. Phys. Let.*, 198:89, 1992.

[2] Ananth Grama, Vipin Kumar, and Ahmad Sameh. Scalable parallel formulations of the barnes-hut method for *n*-body simulations. In *Supercomputing '94 Proceedings*, 1994.

[3] L. Greengard and W. Gropp. A parallel version of the fast multipole method. *Parallel Processing for Scientific Computing*, pages 213–222, 1987.

[4] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comp. Physics*, 73:325–348, 1987.

[5] J. F. Leathrum and J. A. Board. Mapping the adaptive fast multipole algorithm into mimd systems. In P. Mehrotra and J. Saltz, editors,

Unstructured Scientific Computation on Scalable Multiprocessors. MIT Press, Cambridge, MA, 1992.

[6] K. E. Schmidt and M. A. Lee. Implementing the fast multipole method in three dimensions. *J. Stat. Phys.*, 63:1120, 1991.

[7] J. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. Load balancing and data locality in hierarchical n-body methods. *Journal of Parallel and Distributed Computing*, 1994 (to appear).

[8] M. Warren and J. Salmon. Astrophysical n-body simulations using hierarchical tree data structures. In *Proceedings of Supercomputing Conference*, 1992.

[9] M. Warren and J. Salmon. A parallel hashed oct tree n-body algorithm. In *Proceedings of Supercomputing Conference*, 1993.

[10] F. Zhao and S. L. Johnsson. The parallel multipole method on the connection machine. *SIAM J. of Sci. Stat. Comp.*, 12:1420–1437, 1991.