Scalable Parallel Formulations of the Barnes-Hut Method for n-Body Simulations*

Ananth Y. Grama, Vipin Kumar, and Ahmed Sameh

Department of Computer Science, University of Minnesota, Minneapolis, MN 55455

Abstract

In this paper, we present two new parallel formulations of the Barnes-Hut method. These parallel formulations are especially suited for simulations with irregular particle densities. We first present a parallel formulation that uses a static partitioning of the domain and assignment of subdomains to processors. We demonstrate that this scheme delivers acceptable load balance, and coupled with two collective communication operations, it yields good performance. We present a second parallel formulation which combines static decomposition of the domain with an assignment of subdomains to processors based on Morton ordering. This alleviates the load imbalance inherent in the first scheme. The second parallel formulation is inspired by two currently best known parallel algorithms for the Barnes-Hut method. We present an experimental evaluation of these schemes on a 256 processor nCUBE2 parallel computer for an astrophysical simulation.

1 Introduction

The n-body simulation problem, also referred to as the many-body problem finds extensive applications in various engineering and scientific domains. Important applications of this problem are in astrophysical simulations, fluid dynamics, design of composites, and protein synthesis.

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. Typical simulations comprise of millions of particles. Clearly, it is not feasible to compute n^2 interactions for such values of n. 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 boxes 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 [1, 4], Barnes and Hut [2], and Greengard and Rokhlin [8, 6, 7].

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 $O(n)$[8], 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 which make it difficult to program. However, FMM has proven error bounds unlike the Barnes-Hut method. Warren and Salmon [15] 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. The experimental study is performed in the context of a 2-dimensional astrophysical simulation but the results hold for the Barnes-Hut algorithm applied to any general n-body simulation problem in three dimensions too.

The Barnes-Hut method works in two phases: the tree construction phase and the force computation phase. In the tree construction phase, a spatial tree representation of the domain is derived. At each step in this phase, if the domain contains more than one particle, it is recursively divided into four equal parts (eight parts in three dimensions). This process continues until each part has a single element in it. The resulting tree is an unstructured quad-tree (oct-tree in three dimensions). This tree is now traversed in post-order. Each internal node in the tree computes and stores an approximate representation of the particles contained in that tree. For astrophysical simulations, this can be approximated by the center of mass of the particles contained in the tree. Once the tree has been computed, the force on each particle can be be computed as follows: the multipole acceptance criteria is applied to the root of the tree to determine if an interaction can be computed; if not, the node is expanded and the process is repeated for each of the four sons. The multipole acceptance criteria for the Barnes-Hut method computes the ratio of the dimension of the box

*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. The authors can be reached via E-mail at {*ananth, kumar, sameh*}*@cs.umn.edu*.

to the distance of the point from the center of mass of the box. If this ratio is less than some constant α, an interaction can be computed.

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 [3, 16, 9, 11, 7]. For irregular distributions, these objectives are hard to achieve because of the highly unstructured nature of both computation and communication. Singh et al. [12] and Warren and Salmon [15, 14] present schemes for irregular distributions that try to meet these objectives. Motivated by these schemes, in this paper, we propose two new parallel formulations. These formulations are characterized by their simplicity and performance and capture most of the communication in two collective communication operations. The first formulation uses a static decomposition of the domain and assignment of subdomains to processors. The load balancing is implicit in this scheme. Although, this formulation incurs some overhead due to load imbalances, it has excellent communication characteristics. In the rest of the paper, we refer to this formulation as the simple scheme. The second formulation uses a dynamic assignment of subdomains to processors. It tries to balance load by assigning clusters to processors so that the load at all processors is the same. This formulation also preserves many of the communication characteristics of the simple scheme. This formulation is hereafter referred to as the hybrid scheme.

This paper appears in an extended form in [5].

2 Related Research

In this section, we overview two existing parallel formulations of the Barnes-Hut method proposed by Singh et al. [12] (referred to as Costzones) and Warren and Salmon [15] (hereafter referred to as WS). These schemes are currently acknowledged to be the best known parallel formulations of the algorithm for irregular particle distributions. We also discuss close similarities between these two schemes.

2.1 Costzones

This scheme was designed and presented for shared address space computers such as the DASH and KSR. It uses the Barnes-Hut tree to partition the domain of simulation among various processors. [1]

The first step of the Costzones formulation constructs the Barnes-Hut tree. Initially, there exists some arbitrary distribution of particles among the processors. Starting with an empty domain, each processor injects its particles into the domain, one at a time. As soon as an injected particle leads to two particles in the same domain, the domain is divided recursively into four subdomains until each subdomain contains just one particle. Since the tree is shared by all processors, it might appear that this process would result in a large number of locking operations on the root node of the tree. However, while reading a node, it is not necessary to lock the node. A node needs to be locked only when its corresponding domain is divided into four parts or when a particle is inserted into it. Since the particles assigned to a processor are localized in physical space, different processors are likely to lock different nodes in the tree. Therefore, the locking overhead incurred in this phase is not significant. Singh et al. [12] also present an improved tree construction scheme. This scheme requires all processors to construct local trees assuming that they have the entire domain. These local trees are then merged to form a single tree.

Once the Barnes-Hut tree has been constructed, it needs to be partitioned among various processors in such a way that each processor gets equal load. In the Barnes-Hut method, the computation of total force acting on a particle may require traversing a number of nodes in the tree. The number of nodes traversed for computing the force in one iteration can be used as a measure of the load associated with the particle in the next iteration. For partitioning the tree among processors, each node inside the tree is labeled with the sum of loads at all nodes in the tree rooted at the particular node (including itself). The root node is labeled with the entire computational load W of the previous iteration. To balance this load among each of the p processors, each processor must be assigned W/p load. Conceptually, in the Costzones partitioning scheme this is done as follows: The tree is traversed in an in-order fashion. If a node is encountered in the tree whose acquisition does not cause the load at the processor to exceed W/p, this node is added to the partition and the subtree rooted at this node need not be traversed. This partition is assigned to the first processor. The next W/p load is assigned to the second processor and so on. (In reality, this partitioning is done concurrently.) If the sons of a node are ordered in an arbitrary manner, the resulting partitions assigned to a processor may not be contiguous in space. Singh et al. show that for maintaining spatial continuity in the physical domain, the sons of a node must be numbered in a particular sequence.

Once the particles have been assigned to processors, the processors can compute forces on each of its particles by starting at the root and traversing down the tree until all interactions have been computed. The top level nodes in the tree are repeatedly accessed by all the processors. Once these nodes are fetched into the cache, all subsequent accesses to these nodes are localized. This reduces communication overhead significantly.

2.2 WS parallel formulation

This scheme was originally presented by Warren and Salmon [15] using Morton ordering, for a message passing computer. In the WS scheme, a key is computed for each particle by interleaving the bits of the x and the y coordinates. (For instance, if the x coordi-

[1] This scheme was primarily presented in the context of the Fast Multipole Algorithm. But as the authors note, the same scheme works for the Barnes-Hut algorithm too. In this paper, we discuss it only in the context of the Barnes-Hut algorithm.

nate of a particle is 1010 and the y coordinate is 1100, the resulting key is 11100100. The least significant bit of the key is the least significant bit of x, the next bit is the least significant bit of y and so on.) The particles are then sorted on the basis of this key. The sorted sequence is a Morton ordering of the particles (based on their coordinates). The load associated with a particle is assumed to be the number of tree nodes traversed to compute the force on the particle in the previous iteration. The load associated with each particle is also stored along with its key. This list can now be partitioned by assigning particles whose cumulative load is equal to W/p from this sorted list to each processor. As Warren and Salmon note, this scheme can be easily modified to use Peano-Hilbert ordering instead of the Morton ordering, which results in better communication characteristics. In the rest of the paper, we will assume a Peano-Hilbert ordering for this scheme.

After assigning particles to processors, each processor constructs the local tree using its particles. The WS scheme defines a set of branch nodes in the tree which represent the entire processor domain at the coarsest level. These branch nodes are globally communicated to all processors. Each processor proceeds to fill the top part of the tree locally. In the final phase of the WS scheme, processors proceed to compute interactions for the particles assigned to them. The top level nodes which are repeatedly accessed are available to all processors locally since processors construct the top part of the tree independently. Each processor computes all the interactions it can compute with locally available data first. Requests for data required for interactions with non-local nodes are buffered. This is done to reduce the latency in communication. Furthermore, repeated requests to identical pieces of data are merged into a single request. Once this data is fetched, more interactions can be computed. This may result in additional requests for non-local data which is then fetched. This process is continued until all the interactions have been computed.

2.3 Comparison of Costzones and WS schemes

The Partitioning Schemes The partitioning scheme in Costzones orders the successors of a node in the Barnes-Hut tree is a certain way. If this Barnes-Hut tree is traversed in an in-order fashion, the leaf nodes are encountered in exactly the same sequence as they are numbered by a Peano-Hilbert ordering. This implies that the partitions arrived at by both the Costzones and the WS schemes are identical. This is illustrated in Figure 1.

The only difference between these two schemes in the tree partitioning phase is how the partitions are arrived at. The Costzones scheme takes a top down approach to deriving the partitions. The WS scheme, on the other hand, builds a partition bottom up by aggregating individual units of load together.

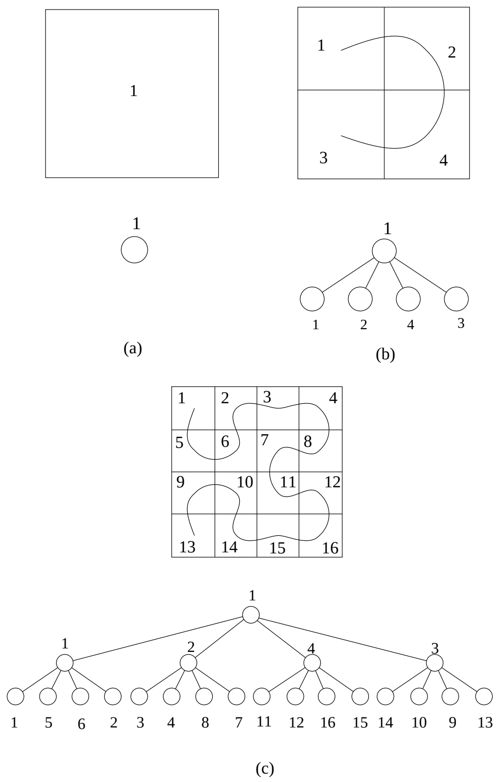

Figure 1: Illustration of the equivalence of Peano-Hilbert ordering and the numbering scheme used by Singh et al.

Tree Construction A careful examination of the tree construction in WS and Costzones reveals that the top part of the tree containing the branch nodes in the WS scheme is identical to the global nodes in the Costzones scheme which participate in the tree merge operation. Therefore, tree construction in both schemes involves sharing of identical pieces of data.

Performance In the implementation of Costzones on shared address space computers, caching of data plays an important role in enhancing the performance of this scheme. The top level nodes in the tree participate in the tree merging process and are cached at each processor. These nodes are also made available to all processors in the WS scheme by a broadcast. The overhead incurred for accessing these nodes in either scheme is therefore identical since in both cases they are available locally. In the force computation phase, local tree traversal is identical in both schemes since the partitions assigned to a processor are the same. For non-local tree traversals, data is fetched when required and traversal is performed at the processor which holds the particle. In Costzones, this data is cached, and therefore repeated access to this data can be localized. In the WS scheme, the latency hiding mechanism implemented by Warren and Salmon ensures that repeated access to non-local data can be clubbed together into a single request.

We can conclude from this discussion that the two schemes are very similar in their partitioning, tree construction and force computation. The primary difference is that whereas the WS scheme explicitly manages data movement, the Costzones scheme relies on efficient cache management to handle data movement.

2.4 Issues and Potential Enhancements

We now try to identify ways in which the performance of these two schemes can be improved.

- The WS scheme is a message passing implementation of the partitioning technique since it is bottom-up and requires minimal global information. However, it requires sorting a large number of keys for deriving the Peano-Hilbert ordering. Costzones uses a top-down approach but needs global tree data for partitioning. The question is whether it is possible to develop a scheme that yields similar partitions without paying the cost of sorting N keys or using global tree data.
- The top levels of the tree are accessed repeatedly for each particle in the tree. It is therefore desirable that these nodes be made locally accessible to all the processors. Rather than trying to minimize the number of globally shared nodes, it is actually possible to use these to enhance performance. The top level nodes can be made available to all processors by a single k-to-all broadcast operation. Here, k is the subset of processors which contain nodes in the top part of the tree. Since the k-to-all broadcast of a relatively small number of nodes is not expensive, is it possible to control the depth of k-to-all broadcast to enhance the performance of these schemes?
- Both of these schemes fetch data from remote processors and compute non-local traversals at the processor holding the particle. An alternate approach is to send particle coordinates to the processor which holds the subtree for which interactions need to be computed. Is the latter approach, used in conjunction with a controlled depth k-to-all broadcast capable of yielding improved performance?
- One of the primary deterrents to porting n-body codes to message passing computers identified by Singh et al. [13] is the fact that receiver initiated data transfer of small messages over long distances is very expensive. Is it possible to formulate the n-body problem in such a way that no receiver initiated fine grain long distance communication is required?

In this paper, we present parallel formulations which address these issues.

3 Parallel Formulation of the Barnes-Hut Method

In this section, we present a parallel formulation of the Barnes-Hut method that divides the domain into a number of subdomains (or clusters) and assigns them statically to processors. The parallel formulation comprises of the following steps:

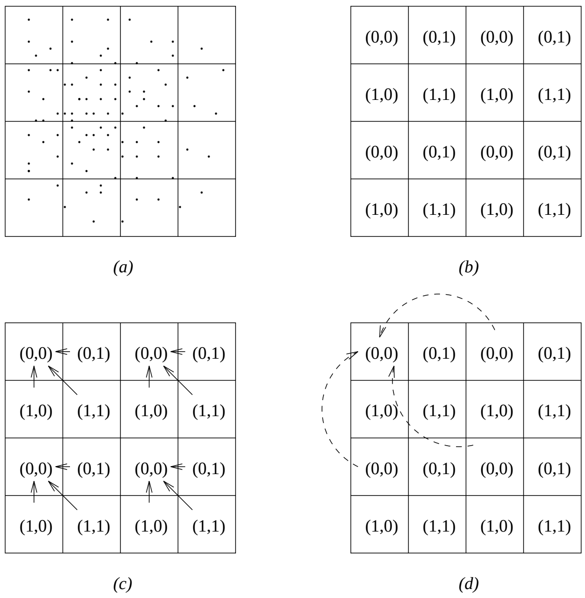

Figure 2: (a) Partitioning the domain into $r = 16$ parts; (b) Mapping the sub-domains to processors; (c) and (d) Communication for merging subtrees to form global tree. Note that no communication is required in step (d).

Domain decomposition and mapping 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 (gray$(i, d/2)$, gray$(j, d/2)$). Here, 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. The partitioning and mapping are illustrated in Figures 2(a) and (b). We refer to this assignment as *modular assignment.* This is similar to modular scatter decomposition discussed by Nicol and Saltz [10]. The corresponding tree partitioning imposed by this domain decomposition is illustrated in Figure 3.

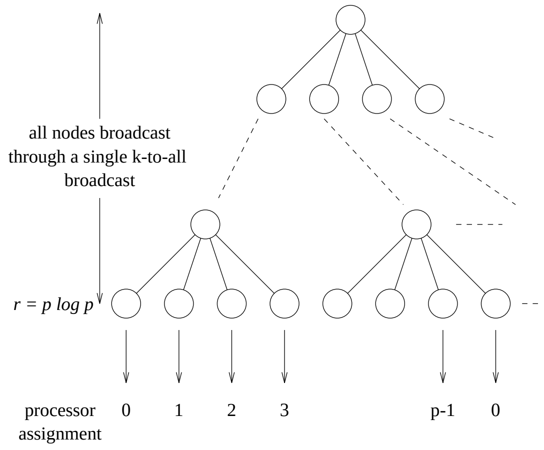

Figure 3: Tree partitioning imposed by the domain decomposition into $r = p \log p$ clusters.

Tree construction Tree construction is simplified by two characteristics: all particles in a particular sub-domain 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. This is illustrated in Figure 3. 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 it, a processor constructs the corresponding tree. The dimensions of the root node in the tree are those of the sub-domain and not the entire domain.

In the second step, these trees are merged. The only information that needs to be communicated for merging trees is the number of particles contained in a particular local tree and the center of mass information. Figures 2(c) and (d) illustrate the communication patterns for the merge operation. The merge operation is performed in $\log_4 r$ steps. At each step, four nodes are combined into a single node. Of these, in the first $\log_4 p$ steps, one node is available locally, and the other three are communicated from other processors. An efficient way of merging trees is as follows: each processor determines the position of every cluster assigned to it in a 2-dimensional grid. For a particular cluster, let this position be (i, j). In step l, the processor computes $(i \bmod 2^l, j \bmod 2^l)$. This tuple represents the offset of the destination cluster that would receive the particular node from the processor during that step. If this offset is $(0, 0)$ then the processor inserts its node from the previous level as one of the sons and waits for three sons to arrive from other processors. If this offset is (x, y) where at least one of x or y is non-zero, then the node from the current level is sent to the processor holding cluster $(i - x, j - y)$. After sending the node corresponding to a cluster to another processor, no further processing needs to be performed for it in the tree-merge phase. The processor receiving and combining nodes simply adds the number of particles contained in each sub-tree. If this number is one or less, it just ignores the existence of all local sub-trees; else it marks the four subtrees in appropriate data structures. In this way, sub-trees are merged until the complete tree is built. The center-of-mass computations can also be performed locally and combined during the tree construction phase in an identical manner.

k-to-all broadcast In the Barnes-Hut algorithm, force computation for each particle starts at the root of the tree. Therefore, this node is repeatedly accessed by all processors and becomes a major source of communication and contention overheads. This overhead is also incurred for nodes at the top few levels of the tree (that are also highly likely to be repeatedly accessed). To alleviate these overheads, the top $\log_4 r$ levels of the tree are broadcast to all processors using a single k-to-all broadcast. In general the number of levels broadcast is a function of the particle distribution, the number of processors and the parameter α of the simulation. As we shall discuss in Section 5.1, this k-to-all broadcast helps us to localize communication during the force-evaluation phase significantly.

Force evaluation Since the top $\log_4 r$ levels of the tree are available locally to all processors, interactions with these nodes are computed first. For each particle, there will be other nodes in the tree whose interactions cannot be accounted for in the top $\log_4 r$ levels. Typically, a particle would need to interact deeper with the nodes in the subtree to which it belongs, and also with nodes in trees corresponding to neighboring clusters. Thus, each particle has a list of clusters whose force contributions have not yet been computed. Each processor maintains p bins. Bin i holds all particles that require force contribution from a cluster at processor i. Since a processor contains more than one cluster, each particle in a bin must also store the cluster at the processor it needs to interact with. An alternate formulation could just send the particle data to the destination processor and the recipient processor can determine the clusters for which force contributions have to be computed. It is easy to see that each bin corresponds to a certain amount of work (force computations). The bin label tells us the processor holding the data required for the computation in the bin.

At this stage, it is possible to compute interactions of particles with all clusters contained at the processor locally. The computation associated with the remaining $p - 1$ bins is performed in the next step.

All-to-all personalized communication At this point, there are two ways of computing the rest of the interactions. The first, and the commonly used approach, is to get data pertaining to the non-local nodes and perform interactions locally [12, 14, 15]. This may require a significant amount of communication because it is difficult to estimate which non-local nodes a given particle interacts with. We take

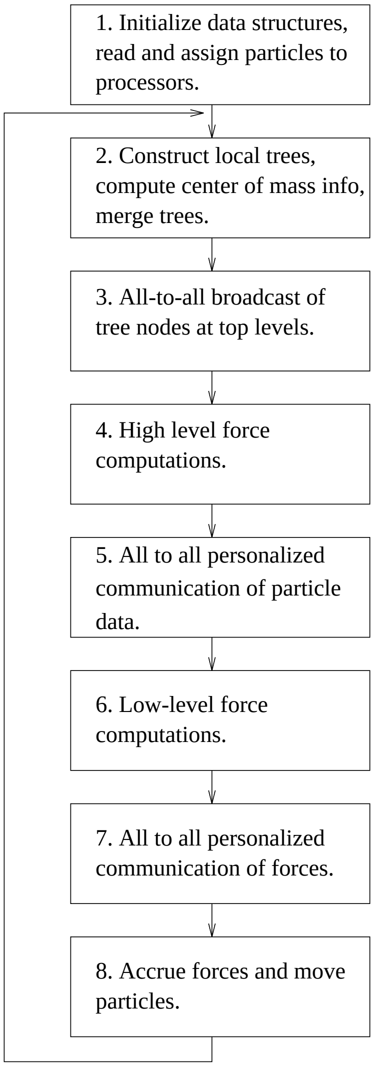

Figure 4: Block diagram of the parallel Barnes-Hut algorithm.

an alternate approach to computing these interactions. Instead of fetching non-local nodes, we communicate the particle data to the processors holding the non-local nodes. The interactions are computed at the remote processors and the force contributions are returned. This requires an all-to-all personalized communication for communicating particle data, followed by a force evaluation phase, followed by another all-to-all personalized communication for returning force contributions.

Accrue forces and move particles In this phase, all forces on a particle are added up and the particle is moved to its new position. If the particle crosses the boundary of a sub-domain, it is communicated to the appropriate processor. This phase requires predominantly nearest neighbor communication since a particle is not expected to skip entire subdomains in a single time-step.

4 Improved Cluster Mapping

The simple scheme 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.

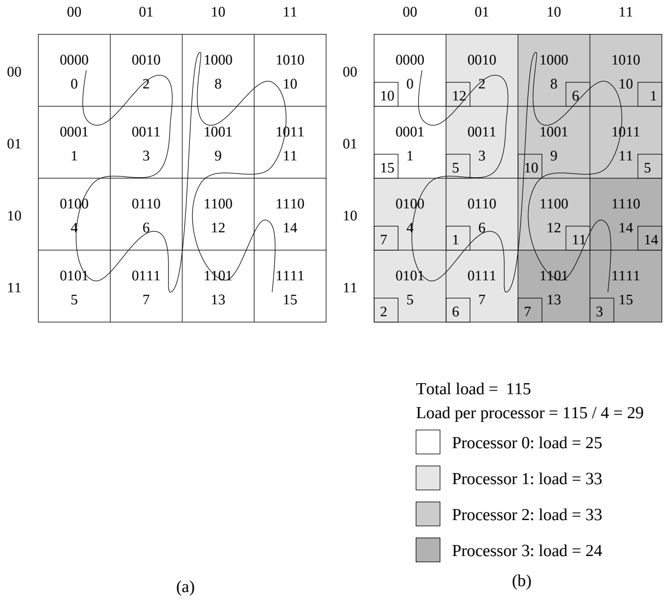

Figure 5: (a) Morton ordering of a domain decomposed into 16 clusters; (b) An illustration of cluster mapping using the improved scheme. The numbers in the inset represent loads in each cluster. The total load is determined by adding these loads. Each processor is assigned approximately equal load in 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 5(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. After the new cluster assignments have been computed, each processor broadcasts the starting point of its clusters in the Morton ordering. Cluster data can now be moved between processors as required by the reassignment. This data movement will not be significant since cluster loads are not expected to change drastically after each iteration.

It is important to note the structure of computation at this point. The block diagram of the parallel formulation is shown in Figure 4. Steps 4 and 6 represent the dominant computation in the parallel formulation. A load balancing algorithm may try to balance the computation in one of these steps (by only counting the number of node traversals in either the top level interactions or in lower level interactions) or the sum of computational load at each processor during the two steps combined. We have noticed that balancing computation only in one of these steps leads to significant imbalances in the second phase. Therefore, it is crucial to balance the sum of loads at each processor during the two iterations combined. In our parallel formulation, this is achieved by counting the number of node traversals in both the top level and the lower level interactions. With such a load balancing scheme, it is possible that a processor may have a very high load in Step 4 and a low load in Step 6, and another processor may have the opposite. Conventional all-to-all personalized communication algorithms enforce an implicit synchronization point. This is not desirable since a synchronization between Steps 4 and 6 would lead to significant overall load imbalance. Therefore, we implement a non-synchronizing protocol for all-to-all personalized communication in Step 5. Note that it is possible to use a conventional all-to-all personalized communication in Step 7 since the load balancing algorithm ensures that all processors will reach this step at approximately the same time.

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.

5 Performance of the New Formulations

The simple scheme (based on modular assignment of clusters to processors) provides a nearest neighbor mapping of clusters to processors even for a mesh connected architecture. Consequently, most of the communication during the force computation phase is between neighboring processors. This is not the case with the other parallel formulations. When the particle distributions are not extremely uneven, modular assignment is capable of yielding good load balance. On architectures such as a mesh that have limited cross section bandwidth, the nearest neighbor characteristic of modular assignment becomes important.

The hybrid parallel formulation has an advantage over the WS scheme because after an initial sort of r cluster-morton-numbers, this scheme does not require any sorting. The sorting cost is a non-trivial cost in the WS scheme. This improvement comes at the expense of slight load imbalances.

5.1 Role of Collective Communication Operations

The k-to-all and all-to-all personalized communications play a very important role. Depending on the α parameter of the simulation, nodes from the top few levels in the tree will be repeatedly accessed by all processors. It is therefore necessary that these be made available locally to all the processors. The depth should not be so high that the broadcast operation itself becomes expensive. The optimal depth is determined by the accuracy of the simulation, the particle distribution and the number of processor. Typically, for a given particle distribution and a given number of processors, the optimal depth of broadcast increases as α is decreased.

Singh et al. [13] identify several drawbacks of programming n-body codes on message passing computers. One of the main drawbacks is the the fact that on-demand or receiver initiated data transfer of small data items over large distances is expensive in message passing computers. The top level k-to-all broadcast takes care of most of the long range data transfers. The all-to-all personalized communication provides a sender initiated mode of data communication with messages of coarse granularity. Another drawback identified is the fact that in message passing computers, the programmer explicitly needs to eliminate duplicate requests for the same data item [12]. This duplication check is implicit in the force-computation phase of our parallel formulations. A particle can interact with a particular cluster only once and so there is no possibility of sending multiple copies of particle data to the same processor. Therefore, the two communication operations together address the major drawbacks of implementing n-body codes on parallel computers.

6 Experimental Results

In this Section, we present an experimental evaluation of the two parallel formulations in the context of an astrophysical simulation. The simulation code is written for an nCUBE2 parallel computer. We have run a number of simulations ranging from a few hundred thousand particles to over a million particles. The objectives of this experimental evaluation are to show that:

- A simple partitioning scheme that modularly assigns clusters to processors can offer acceptable load balance even when particles are not uniformly distributed. Coupled with the two collective communication primitives, the overall performance of this scheme is good.

- The second parallel formulation based on Morton ordering coupled with clustering yields better load balance and performance compared to the simple scheme.

The simulation runs presented here are single iteration runs. Particle densities of test cases are assumed to be single and double Gaussian distributions. Particle densities in the densest regions are a few thousand times those in the sparse regions. The hybrid parallel formulation runs two iterations. It collects load data from the first iteration and moves clusters between processors. The first iteration is not timed. However, the time for the second iteration also includes the time for data collection (corresponding to cluster loads) and partitioning.

We now present runtimes of a number of simulation instances on an nCUBE2 parallel computer for up to 256 processors. The problems contain approximately 160K, 326K, 657K, and 1.2 million particles. The problem instances are named as 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. Unless otherwise mentioned, all other problems contain a single gaussian distribution. The largest problem requires approximately 1.3×10^8 force computations. A problem of this size is considered relatively small for n-body simulations. By demonstrating good performance and even for such small problems, we effectively demonstrate the quality of these parallel formulations. On bigger problems, the computation per processor will be an even bigger fraction of the total time taken, and thus better efficiencies will be obtained. The size of the problems attempted is limited by the fact that the available nCUBE2 computer only had 4 Megabytes of memory at each processor. Infact, 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 following conclusions can be drawn from this table: the parallel runtime of both formulations reduces 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 hybrid scheme and by a factor of 3.6 for the simple scheme. 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. The simple scheme has higher runtimes because of load imbalances.

Table 2 demonstrates the effect of increasing the number of clusters on the performance of the two schemes. As we increase the number of clusters, the load balance should improve for both schemes. However, for the simple scheme, when the number of clusters is increased beyond a certain point, the gains in load balance can be offset by the increased communication overheads incurred during the tree construction and remote force computation phases. This is because

Table 1: Runtimes (in seconds) of the simple and hybrid schemes for various problems (F denotes the number of force computations).

No. of Processors → Problem	Scheme	16	64	256
g_160535	Simple	179.74	65.53	25.08
($\alpha = 0.67$)	Hybrid	132.37	51.02	17.13
$F = 2.2 \times 10^7$				
g_326214	Simple	167.449	62.79	22.57
($\alpha = 1.0$)	Hybrid	133.75	45.42	15.63
$F = 2.1 \times 10^7$				
g_657499	Simple		114.75	31.06
($\alpha = 1.0$)	Hybrid		91.02	24.27
$F = 4.6 \times 10^7$				
g_1192768	Simple		197.51	54.86
($\alpha = 1.0$)	Hybrid		163.96	45.17
$F = 1.2 \times 10^8$				

as the clusters become small, more and more particles would require communication with neighboring clusters (that are assigned to neighboring processors). In the hybrid formulation, communication overhead does not increase with the number of clusters. However, the cost of repartitioning the clusters goes up. From Table 2, we can see that in most cases the runtime decreases with increasing number of clusters (we could not increase the number of clusters for these problems beyond 64×64 due to memory limitations). However, for 16 processors, for the simple scheme, the performance degrades as we increase the number of clusters from 32×32 to 64×64.

Let us now take a closer look at the time taken by various phases of the algorithm. A breakup of the time taken by various phases of the two schemes for problems g_1192768 and g_326214 for $p = 256$ is given in Table 3. The time for local tree construction is very small for both schemes. The time taken by the hybrid scheme is slightly more than the simple scheme. This can be explained by the fact that in trying to balance load, the hybrid scheme may have vastly different number of particles at individual processors. The tree merging cost is higher for the hybrid scheme because unlike the simple scheme in which all processors contained equal number of clusters, the number of clusters here may be very different. Furthermore, the systematic communication structure of the simple scheme does not exist for the hybrid scheme in the tree merge phase. The time for all-to-all broadcast is comparable for the two schemes. The hybrid formulation 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, the simple scheme spends no time in balancing load since load balance is implicit. We can see that the hybrid scheme incurs a minimal overhead in balancing load.

Now we study the impact of particle distribution on the performance of the hybrid scheme. We report results of four simulations on datasets s_1g_a, s_1g_b,

Table 2: Runtimes (in seconds) for different number of clusters for the two parallel formulations.

p	Problem	Scheme	Number of clusters		
			16×16	32×32	64×64
16	g_28131	Simple	27.16	28.12	28.22
	($\alpha = 0.67$)	Hybrid	23.97	21.38	20.75
	g_160535	Simple		179.74	180.6
	($\alpha = 0.67$)	Hybrid		139.27	132.37
	g_326214	Simple		167.45	173.615
	($\alpha = 1.0$)	Hybrid		133.75	134.403
64	g_160535	Simple	69.56	70.35	65.53
	($\alpha = 0.67$)	Hybrid	69.83	69.52	51.02
	g_326214	Simple	67.43	65.88	62.79
	($\alpha = 1.0$)	Hybrid	61.48	50.49	45.42
	g_657499	Simple		127.61	114.75
	($\alpha = 1.0$)	Hybrid		109.39	91.02
256	g_326214	Simple		28.26	22.57
	($\alpha = 1.0$)	Hybrid		18.19	15.63
	g_657499	Simple		40.61	31.06
	($\alpha = 1.0$)	Hybrid		31.48	24.27

s_10g_a, and s_10g_b. Each of these simulations contains 25,130 particles. Dataset s_1g_a has a single gaussian distribution centered randomly in a 100×100 simulation domain. The variance of the distribution is such that all particles lie within a 2×2 subdomain. Dataset s_1g_b is a single gaussian distribution of 25,130 particles with a lower variance. All particles are distributed in a 4×4 subdomain. Datasets s_10g_a and s_10g_b have 10 gaussians (each with 2513 particles) centered randomly in the domain. Dataset s_10g_a has a high variance with all particles belonging to a certain gaussian distributed in a 2×2 subdomain. Dataset s_10g_b has a lower variance with all particles belonging to a certain gaussian distributed in a 4×4 subdomain. Table 4 presents the speedups obtained for these datasets on 4, 16, and 64 processors with different numbers of clusters. The speedup results can be explained on the basis of available concurrency and load balance. Partitioning the dataset s_1g_a into 128×128 clusters yields very few clusters that have any particles in them. Therefore, the available concurrency is limited. Consequently, the speedup saturates at a small number of processors ($p = 4$). Increasing the number of clusters increases the degree of concurrency and therefore the saturation point is pushed back ($p = 16$). Decreasing the variance has the effect of providing better concurrency for the same number of clusters. Therefore, simulations on dataset s_1g_b yield better speedup. Partitioning dataset s_10g_a into 128×128 clusters yields a larger number of clusters with particles compared to the datasets with a single gaussian. Increasing the number of areas of particle concentrations, in general, yields higher concurrency and better load balance. Reducing the variance of distributions in dataset s_10g_a, as is done in dataset s_10g_b yields a more regular distribution and consequently better performance. For problems with very irregular particle concentrations, it becomes necessary to increase the number of clusters to very large values before adequate load balance can be achieved. In current implementations, we tie the top-level broadcast to the number of clusters. When the number of clusters become large, this broadcast becomes too expensive. We are currently working on de-coupling the top-level broadcast from the number of clusters. This would enable us to increase the number of clusters arbitrarily and thus balance load while incurring minimal communication overhead.

Table 3: Time taken (in seconds) by various phases of the parallel formulations for the simple and hybrid 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.

Problem→	g_1192768		g_326214	
Phase	Simple	Hybrid	Simple	Hybrid
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 4: Speedup results for four problems with varying degrees of irregularities. F represents the total number of force computations. The α parameter for these simulations is set to 0.67.

Problem	Number of Clusters	Speedup		
		$p = 4$	$p = 16$	$p = 64$
s_1g_a	128×128	3.1	3.07	2.98
($F = 6.8 \times 10^6$)	256×256	3.5	8.2	7.9
s_1g_b	128×128	3.68	11.46	11.23
($F = 4.9 \times 10^6$)	256×256	3.79	12.38	20.10
s_10g_a	128×128	3.73	12.51	28.16
($F = 5.1 \times 10^6$)	256×256	3.78	13.81	39.40
s_10g_b	128×128	3.81	13.81	38.46
(4.1×10^6)	256×256	3.80	13.83	44.18

7 Conclusions and Future Research

In this paper, we have presented two parallel formulations of the Barnes-Hut algorithm. We have discussed the relationship of these schemes with the existing schemes of Singh et al. [12] and Warren and Salmon [15]. We have shown the similarity between the Costzones and WS schemes and show how our parallel formulations enhance the performance of these schemes by using sender initiated coarse grain localized communication. We have also analyzed various parameters of our formulations and experimentally demonstrated the performance of these formulations.

This work opens a number of avenues for future research. It is necessary to analytically study the impact of the two communication operations on the performance of the parallel formulation. One possible enhancement to these new parallel formulations is to define broadcast windows. With broadcast windows, the top few levels of the tree can be broadcast globally and the next few levels can be selectively broadcast to subsets of processors (in the windows of processors that are likely to use these nodes). The purpose of this broadcast is to minimize receiver-initiated communication of short messages without paying a significant overhead involved in the broadcast of large amounts of data over the entire set of processors. Another important aspect that needs investigation is the fact that as the simulation proceeds, the particle densities may become more and more irregular. The number of clusters will therefore have to be dynamically changed to accommodate this increased irregularity. It would be interesting to explore the impact of number of clusters on the performance of these schemes analytically.

8 Acknowledgements

The authors sincerely acknowledge S. Sahni and R. Fischer for generous access and help with the 64 processor nCUBE2 at the University of Florida. The authors also acknowledge Sandia National Labs for providing access to their 1024 processor nCUBE2.

References

[1] A. W. Appel. An efficient program for many-body simulation. *SIAM Journal of Computing*, 6, 1985.

[2] J. Barnes and P. Hut. A hierarchical o(n log n) force calculation algorithm. *Nature*, 324, 1986.

[3] 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.

[4] K. Esselink. The order of appel's algorithm. *Information Processing Letters*, 41:141–147, 1992.

[5] Ananth Y. Grama, V. Kumar, and A. Sameh. Scalable parallel formulations of the barnes-hut method for n-body simulations. Technical report, University of Minnesota, Minneapolis, 55455, 1994. Also available through anon. FTP from ftp.cs.umn.edu:users/kumar/nbody.ps.

[6] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems.* ACM Press, 1987.

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

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

[9] 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.

[10] D. M. Nicol and J. H. Saltz. An analysis of scatter decomposition. *IEEE Transactions on Computers*, 39:1337–1345, November 1990.

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

[12] 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).

[13] J. P. Singh, J. L. Hennessy, and A. Gupta. Implications of hierarchical n-body techniques for multicomputer architectures. Technical Report CSL-TR-92-506, Stanford University, 1992.

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

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

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