Parallel Matrix-Vector Product Using Approximate Hierarchical Methods *

Ananth Grama, Vipin Kumar, Ahmed Sameh
Department of Computer Science,
University of Minnesota
Minneapolis, MN 55455
{*ananth, kumar, sameh*}*@cs.umn.edu*

Abstract

Matrix-vector products (mat-vecs) form the core of iterative methods used for solving dense linear systems. Often, these systems arise in the solution of integral equations used in electromagnetics, heat transfer, and wave propagation. In this paper, we present a parallel approximate method for computing mat-vecs used in the solution of integral equations. We use this method to compute dense mat-vecs of hundreds of thousands of elements. The combined speedups obtained from the use of approximate methods and parallel processing represent an improvement of several orders of magnitude over exact mat-vecs on uniprocessors. We demonstrate that our parallel formulation incurs minimal parallel processing overhead and scales up to a large number of processors. We study the impact of varying the accuracy of the approximate mat-vec on overall time and on parallel efficiency. Experimental results are presented for a 256 processor CM5 parallel computer.

1 Introduction

The behavior of many physical systems is governed by Laplace's equation. A variety of techniques exist for solving this equation. One such technique that finds extensive application is the boundary element technique applied to the integral form of Laplace's equation derived from potential theory. Consider the Laplace equation with a Dirichlet boundary condition:

$$\nabla^2 f(u) = 0 \text{ over } \Omega$$

and

$$f(u) = h(u) \text{ on the boundary.}$$

*This work was supported by IST/BMDO through Army Research Office contract DA/DAAH04-93-G-0080, and by Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Access to computing facilities were provided by Minnesota Supercomputer Institute, Cray Research Inc. and by the Pittsburgh Supercomputing Center. Related papers are available via WWW at URL: http://www.cs.umn.edu/users/kumar/papers.html.

To solve this equation, first the boundary of the given domain is divided into a number of panels. Using Green's function of the Laplace equation, the potential at each panel can be represented as the sum of contributions of every other panel. Applying the Dirichlet boundary condition yields a large scale linear system of equations.

For an n panel boundary discretization, the $n \times n$ linear system arising from this approach is dense. Solving the system using direct factorization methods has a computational complexity of $\Theta(n^3)$. This limits the values of n to a few thousands. Therefore iterative methods are used to solve these systems. These methods reduce the operation count significantly. The core operation of these iterative solvers is a matrix-vector product (mat-vec). The complexity of this mat-vec is $\Theta(n^2)$. For large values of n, this becomes extremely expensive. The physical properties of the underlying domain allow us to use approximate hierarchical techniques that can compute this mat-vec in $O(n)$ or $O(n \log n)$ time.

Approximate hierarchical techniques have received a lot of attention in the context of particle simulations. Given n particles, if each particle influences every other particle in the system, a total of n^2 interactions must be computed. However, in most physical systems, the influence of a particle on another diminishes with the distance. In such systems, it is possible to aggregate into a single expression, the impact of several particles on another distant particle. Using this approach, the total number of interactions in the system can be reduced significantly. This forms the basis of hierarchical methods. These methods provide systematic ways of aggregating entities and computing interactions while controlling the overall error in modeling. Algorithms based on hierarchical techniques include Barnes-Hut [2], Fast Multipole [8], and Appel's [1] algorithms. Approximating long range interactions in this manner reduces the sequential complexity of typical simulations involving n particles from $O(n^2)$ to $O(n \log n)$ or $O(n)$.

Particle simulation techniques compute the combined influence on each particle as a sum of the influences of all the other particles in the system. This computation is identical to the computation of the potential at each boundary element as a sum of contributions of each other element. The analog of a particle in n-body simulations is the boundary element in dense mat-vecs.

In many of the modeling problems, it is desirable to have very fine discretizations. This corresponds to a large value of n. Clearly, the reduced computational complexity of hierarchical methods represents a significant reduction in the time for solving the system. However, simulations with millions of boundary elements still take an inordinately large amount of time on conventional serial computers. Parallel processing offers a tool for effectively speeding up this computation. It enables us to solve problems with a large number of elements and to increase the accuracy of simulation by incorporating a higher precision into the approximate hierarchical mat-vec. Parallel formulations of hierarchical methods involve partitioning the domain 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, 17, 10, 12, 7]. For irregular distributions, these objectives are hard to achieve because of the highly unstructured nature of both computation and communication. Singh et al. [13] and Warren and Salmon [16, 15] present schemes for irregular distributions that try to meet these objectives. In [4, 5] we present alternate schemes for irregular distributions that improve on the performance of the earlier schemes.

In this paper, we present a highly scalable parallel formulation of a mat-vec based on approximate hierarchical techniques that incorporates key desirable features of all of these formulations.

We show that our formulation incurs minimal communication and idling overheads even for extremely irregular discretizations. We demonstrate near-linear speedups for up to 256 processors of a CM5 parallel computer for moderate sized problems. We study the tradeoffs of accuracy and computation time and its impact on the parallel processing overheads. We use this formulation to compute mat-vecs for problems ranging from a few thousand to a few hundred thousand elements.

The rest of the paper is organized as follows: Section 2 formulates the Laplace equation in its integral form and demonstrates how approximate hierarchical methods can be used in their solution; Section 3 presents an overview of hierarchical methods and existing parallel formulations; Section 4 presents our parallel formulation; Section 5 presents experimental results; and Section 6 draws conclusions and discusses avenues for future research.

2 Integral Form of the Laplace Equation

In this section, we discuss in more detail the formulation of Laplace's equation as an integral equation. This is done in the context of evaluating the charge density in a given conductor [11, 6]. The charge distribution is related to the potential by the following integral equation:

$$\psi(x) = \int_{\partial\Omega} G(x, x')\sigma(x')da' \tag{1}$$

Here, Ω is the given domain, $\partial\Omega$ the boundary of the domain, $G(x, x')$ is the Green's function, $\psi(x)$ is the given potential at point x on the surface, $\sigma(x)$ is the unknown surface charge density at point x, and da' represents the surface element. The Green's function $G(x, x')$ in three dimensions is given by $1/r$, where r is the magnitude of the vector $x - x'$.

In this paper, we use the boundary element method for solving the integral equation (Equation 1). This method is also referred to as the panel method. In the context of computation of charge densities, a key observation simplifies the solution significantly: the charge in a conducting body resides only on the surface of the conductor. This implies that it is adequate to discretize the surface of the conductor and evaluate the charge at these points. Equation 1 relates the surface charge densities to applied potential using the Green's function. Consider the impact of panel i on the charge distribution in panel j. Let x_j be the center of panel j and $\sigma(x)$ denote the charge density at point x in panel i. Then using Equation 1 and Green's function in three dimensions, the total impact $p_{i,j}$ will be given by

$$p_{i,j} = \int_{\partial(\text{panel } i)} \frac{\sigma(x)}{||x - x_j||} dx$$

Now, let x_i denote the center of panel i. As the magnitude of the vector $x_i - x_j$ becomes much larger than the surface area of panel i, the above integral tends to:

$$p_{i,j} = \frac{q_i}{||x_i - x_j||}$$

Here q_i is the total surface charge on panel i. Each of the panels in the system exerts a similar influence on every other panel. Therefore the combined potential p_j at panel j is a summation of individual panel influences given by:

$$p_j = \sum_{i=1}^{n} \frac{q_i}{||x_i - x_j||} \quad i \neq j$$

The readers would note that this equation is not accurate since nearby panels must be integrated explicitly. We will demonstrate at a later stage how these near panel interactions are handled separately.

The above equation can be rewritten as

$$P = Cq,$$

where P is an n dimensional vector of the known potentials at n panels, C is an $n \times n$ coupling matrix of Green's functions, and q is the vector of unknown charges. Since it is desirable to have a large number of panels, iterative methods such as the Generalized Conjugate Residual Method are used to solve this system. The critical component of these methods is the product of the coupling matrix with a vector. This can be computed using approximate hierarchical methods by treating each panel as a particle. The coupling matrix is never computed explicitly. Instead panels are aggregated and their impact on other panels is expressed in terms of multipole expansions.

3 Hierarchical Methods for Particle Dynamics

A number of hierarchical techniques have been proposed for particle dynamics. Of these, the methods of Barnes and Hut, and the Fast Multipole Method have gained widespread popularity. Both of these methods are based on a hierarchical quad-tree (oct-tree in three dimensions) representation of the domain of simulation. Each node in the tree contains a representation of the particles contained in the physical domain corresponding to the node. This representation is in the form of multipole expansions defined around the center of the box. Particle interactions are computed by traversing the tree systematically.

In the Barnes-Hut method, each particle starts at the root of the tree and applies the multipole acceptance criteria (MAC) to it. The MAC determines whether a direct interaction of the particle with the node leads to an accurate estimation of an interaction of all the particles contained in the corresponding domain. If the MAC is satisfied, a direct interaction is computed, else the node is expanded and the MAC is applied to each of its off-spring. In this way the tree is traversed until all interactions are accounted for. The fast multipole method uses a slightly different traversal technique. Here, in addition to allowing particles to interact with nodes, nodes are also allowed to interact with other nodes. If two nodes i and j in the tree are well separated, then the the impact of all particles in i on all particles in j can be directly computed by translating the multipole expansion of i to the center of box j.

In this paper, we use an augmented version of the Barnes-Hut method for computing the mat-vecs. The Barnes-Hut method was originally presented for astrophysical simulations. Two astrophysical bodies can be adequately represented by their center-of-mass. This corresponds to a monopole expansion. This is not adequate for our application. Therefore, we construct the multipole expansion of each particle around the center of the leaf nodes. The multipole expansion of a non-leaf node is constructed by translating the multipole expansions of the off-springs to the center of the node and adding them. The tree thus constructed is identical to that constructed by FMM. The multipole expansion of a node converges at all points that are well separated from the box. Consider the scenario depicted in Figure 1. The MAC for the Barnes-Hut method computes the ratio of the length of a side of the box l to the distance of the point from the center of the

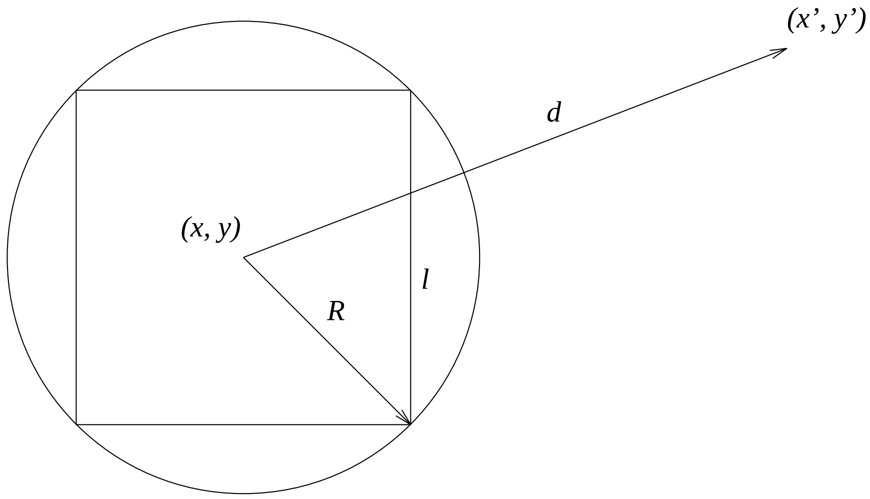


Figure 1: Relation between the convergence of the multipole expansion and the α criterion for Barnes-Hut method.

mass of the box. A direct interaction of the point with the box is allowed if the ratio is less than a preset constant α. We modify the distance d to be the distance between the particle and the center of the box. From Figure 1, it is evident that the multipole expansion converges for all particles at distance $d > R$ from the center of the box. Furthermore, if we assume the domain of simulation to be square, $R = l/\sqrt{2}$. Therefore $d > l/\sqrt{2}$, or the ratio $l/d < \sqrt{2}$. This implies that for the series to converge, the value of α selected for the Barnes-Hut method must be less than $\sqrt{2}$. This is usually the case and we use significantly smaller values of α ensuring greater accuracy.

3.1 Overview of Existing Parallel Formulations of Hierarchical Methods

A number of researchers have explored parallel formulations of hierarchical techniques. Whereas most of the research addresses uniform particle distributions [3, 17, 10, 12, 7], work has also been done for highly non-uniform distributions [16, 15, 13, 4, 5].

The scheme proposed by Singh et al. [13] (referred to as Costzones) 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] For partitioning the tree among processors, each node inside the tree is labeled with the sum of loads from the previous iteration at all nodes in the tree rooted at the particular node (including itself). The root node is labeled with the entire computational load W. To balance this load among each of the p processors, each processor must be assigned W/p load. 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 a 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.

[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.

The scheme proposed by Warren and Salmon [16] (henceforth referred to as WS) was originally designed 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. This key is referred to as the Morton index. 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 both of the schemes discussed above, all force computations for a particle are performed by the processor to which the particle is assigned. This implies that if a tree node is not available locally, it must be fetched. This paradigm of is called *data shipping*. In [4, 5], we noted several drawbacks of data shipping schemes. We propose two other schemes that are based on the *function shipping* paradigm. In these schemes, if a particle requires interaction with non-local data, instead of the data being fetched, the particle is sent to the processor where the data resides. We demonstrated that function-shipping incurs minimal communication overhead. Combined with effective load balancing schemes, our schemes yielded near linear speedups for up to 256 processors of a CM5 multicomputer.

4 Parallel Barnes-Hut Method for Mat-Vecs

In this section, we present a new parallel algorithm for computing approximate mat-vecs using the augmented Barnes-Hut method described in Section 3. We identify overheads in current algorithms and demonstrate how our algorithm overcomes them.

4.1 Drawbacks of Existing Parallel Treecodes

Parallel treecode formulations can be viewed as belonging to one of two broad classes: function shipping and data shipping. The schemes of Singh et. al. and Warren and Salmon are data shipping schemes. We had demonstrated the equivalence of these schemes in [4]. Data shipping schemes incur the following overheads:

- Since a point may interact with any node in a tree, it must be possible to address a node inside a tree quickly. This is done using a hash function in the WS scheme. Collisions are handled using chaining. If the memory allocated to the hash table is significantly more than that required for the tree representation, nodes can be accessed efficiently. However, when this is not the case, randomly accessing nodes in a tree can be a significant overhead. This problem is handled by the cache coherence hardware available in the DASH for the costzones scheme.

- The WS scheme is a message passing implementation of the costzones 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. In this paper, we present an efficient implementation of the partitioning technique for message passing machines that avoids the sorting overhead.

- The communication overhead increases as the degree of multipole expansion is increased.
- One of the primary deterrents to porting n-body codes to message passing computers identified by Singh et al. [14] is the fact that receiver initiated data transfer of small messages over long distances is very expensive.

The function shipping schemes presented in [4] are based on a static decomposition of the domain into a large number of clusters. These clusters are then assigned to processors using a Morton ordering of the clusters. This formulation has been shown to have good performance for moderately irregular particle distributions. However, when the particle distributions become highly irregular, it becomes necessary to use a very large number of clusters. This poses several problems for the implementation of the formulation. Furthermore, it is impossible to estimate the number of clusters *a-priori*. Since the particle densities may change significantly during simulation, it becomes necessary to change the number of clusters accordingly.

4.2 Parallel Formulation of the Barnes-Hut Method

The Barnes-Hut method requires interactions of particles with nodes in the tree. It is possible to view a particle as a computational entity. Since each particle requires traversing a different number of nodes in the tree, the computational load associated with each particle is not the same. However, as noted in [13, 16], since the particles are not expected to move significantly between two time-steps, it is possible to use the number of interactions during the current time-step as an estimate of its load in the next time-step. First, assume that the whole Barnes-Hut tree is available to all the processors. In this case, the only aspect that needs to be addressed by the parallel algorithm is the load balancing problem. This problem can be solved by enumerating all the particles and their corresponding loads and assigning particles with equal aggregate load to each processor. However, in real simulations, the number of particles may be in the range of millions. It is not possible to store the tree corresponding to such domains at every processor. Furthermore, the tree generation phase itself may take a significant amount of time. Therefore, in addition to partitioning the particles, the corresponding tree and its generation must also be partitioned among the processors. This introduces communication overhead.

Assume that there is an initial assignment of particles to processors. The tree construction and force computation phases are implemented as follows:

4.2.1 Tree Construction

In this phase, each processor constructs the Barnes-Hut tree using its own particles. We assume that the domain assigned to each processor corresponds to some node in the Barnes-Hut tree. All particles in this domain are assigned to one processor. The minimal set of nodes in the tree that specify the domain of a processor completely are referred to as its branch nodes. For instance, if the domain of simulation is divided into four parts and each of these is assigned to a different processor, then the four off-springs of the root node are the branch nodes. It is easy to see that the subtree rooted at the branch node is an accurate representation of the Barnes-Hut tree. After constructing

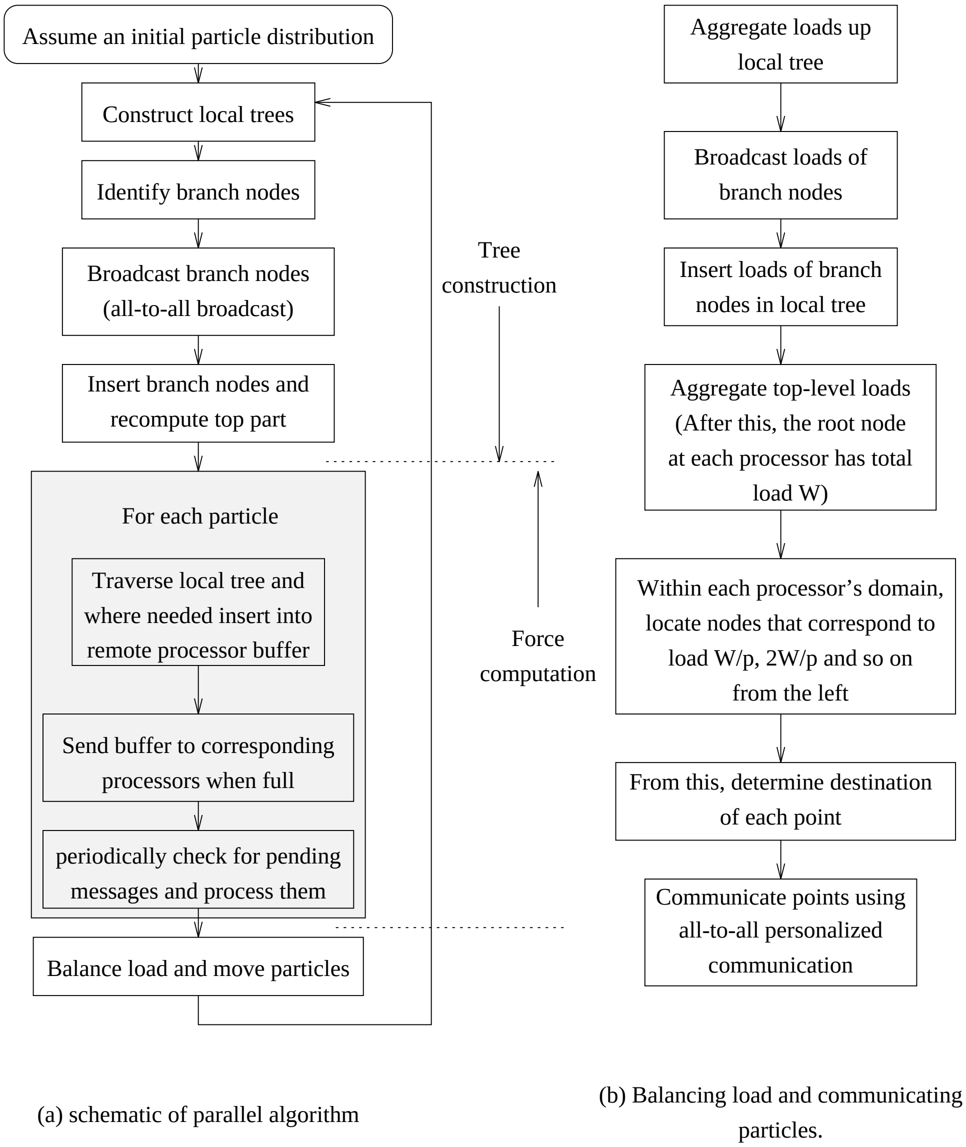


Figure 2: Schematic of the parallel treecode formulation and load balancing technique.

the local trees, the branch nodes are collected and broadcast to all the processors using a single all-to-all broadcast operation [9]. The non-local branch nodes thus received by each processor are inserted into their respective trees and the top levels of the tree are reconstructed by all processors independently. After this phase, each processor has the top few levels of the tree enveloped by the branch nodes, and the complete subtrees rooted at its branch nodes. This phase of the algorithm is identical to the tree construction phase of Warren and Salmon [16].

4.2.2 Force Computation

In this phase, each particle traverses the tree and computes required interactions. For each particle assigned to a processor, it starts at the root node of the tree (which is locally available). It traverses the tree until it reaches a node that is not locally available. Each node in the tree keeps track of the processors that contain its sons. Using this information, the point is placed in the bin corresponding to the processor that hosts the required non-local node. In addition, it also places in the bin a unique key corresponding to the node with which it needs to interact. It can be seen that the only non-local nodes that need interaction are the sons of branch nodes. When the number of points in a bin reaches a preset limit, this bin is sent to the corresponding processor. The processor traverses the rest of the tree inserting non-local nodes into bins and dispatching them when required.

Each processor must also periodically process incoming messages. Messages may be of two types: points that need to be processed, and force contributions corresponding to processed points. For each point that needs to be processed, the processor must locate the branch node and its son that needs to be interacted with. These are located using the unique key corresponding to the branch node. The keys index into a hash table that directly returns a pointer to the corresponding branch node. Since only branch nodes have entries in the hash table, the size of the hash table is small and collisions are rare. Once the required node has been located, the point then interacts with the subtree rooted at that node. The force contribution of the subtree is packaged into a return message. Having done this for each point in the message, the message containing the force contributions is sent back to the originating processor.

After computing local interactions for all particles, each processor is left with a number of half filled buckets. These are dispatched to the corresponding destinations where they are processed. Termination can be detected as follows: each processor keeps track of the number of buckets it sends out. For each response it receives, it decrements the count. At any point of time, this count therefore gives the number of pending messages. When this count becomes zero at all the processors and the local interactions have been computed, the algorithm can terminate.

The tree construction and force computation phases of the algorithm are illustrated in Figure 2.

4.2.3 Load Balancing Technique

The above parallel treecode formulation is combined with a load balancing technique. This technique is an efficient implementation of the costzones scheme on message-passing computers. Each node in the tree contains a variable that stores the number of particles it interacted with. After the force computation phase, this variable is summed up along the tree. The value of load at each node now stores the number of interactions with all nodes rooted at the subtree. The loads at branch nodes are broadcast to all the processors using a single all-to-all broadcast. The non-local branch node loads are inserted into the local tree and the top level loads are recomputed. The root node at each processor now contains the total number of interactions W in the system. This load must be partitioned equally among the processors, i.e., each processor must receive W/p load after load balancing. The corresponding load boundaries are $0, W/p, 2W/p, ..., (p-1)W/p$. The load balancing problem now becomes one of locating these points in the tree. Each processor traverses its locally available tree in an in-order fashion and locates all load boundaries in its own domain. All particles lying in the tree between load boundaries iW/p and $(i+1)W/p$ are collected in a bin for processor i. After each processor goes through its local tree, the points are communicated to the designated processors using a single all-to-all personalized communication [9].

In this load balancing technique, the tree is traversed in a left to right manner. If the sons of a node are ordered arbitrarily, the resulting partitions are not guaranteed to be physically contiguous. Singh et. al. [13] show that by ordering the sons of a node in a specific manner while constructing

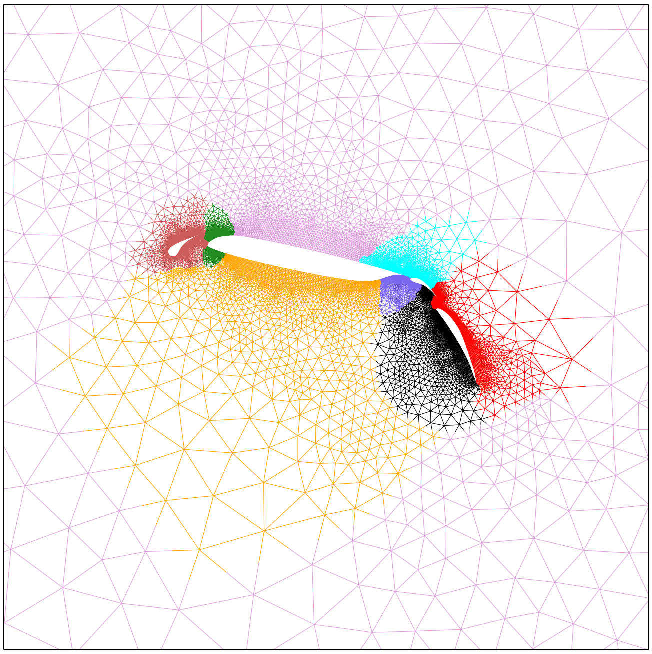


Figure 3: Graph corresponding to problem 4elt.xyz. The graph contains 15606 nodes.

the tree, it is possible to ensure that the partitions assigned to processors are contiguous in space. We have shown [4] that this ordering is identical to constructing a Peano-Hilbert ordering of the nodes.

4.3 Performance of the New Formulation

The parallel formulation presented in this section offers several enhancements over the existing formulations. It does not require direct addressing of a large number of nodes, thus eliminating the memory and performance overheads associated with hashing. It involves no receiver initiated communication, and is therefore more amenable to message passing machines. It does not require sorting of a large number of keys. The construction of the Peano-Hilbert sequence is implicit in the tree construction phase and does not cost anything.

Increasing the accuracy of the hierarchical method increases the communication required in data shipping formulations. This is a direct consequence of the fact that higher accuracy is gained by using higher degree polynomials. Therefore, a larger volume of data will have to be shipped when a node is communicated. The increase in communication may be offset by a corresponding increase in the computation and might not impact the efficiency adversely. However, in function-shipping schemes like ours, the volume of data remains constant as the accuracy is increased. The computation however increases. This implies that as the accuracy is increased, the efficiency of our formulation increases.

5 Experimental Results

In this section, we report on the implementation of our parallel formulation. The objectives of this experimental study are as follows:

- Accurate methods for computing mat-vecs parallelize very well [9]. Approximate methods for computing mat-vecs are considerably faster in the serial context. However, their parallelizability has not been explored. The objective is to demonstrate that the speedup offered by approximate methods can be applied in the parallel context to achieve combined speedups of several orders of magnitude.
- Study the performance of our parallel algorithm and its scalability properties with increasing number of processors and various problem sizes.
- Study the error properties of approximate mat-vecs and their impact on overall computation time and parallel efficiency.
- Study the impact of the α parameter of Barnes-Hut method on the accuracy of the mat-vec.

The parallel algorithm was implemented on a Thinking Machines CM5 parallel computer and tested for up to 256 processors on a number of problem instances ranging in size from 28,060 to 213,671 nodes. Problem 4elt.xyz has 15606 vertices and problem pscan has 213,671 vertices. Both of these problems correspond to unstructured finite element graphs. The discretizations for solving integral equations are different from those for finite element methods. However, our intention here is to highlight the ability of our parallel formulation to handle extremely irregular distributions. Figure 3 shows the graph corresponding to problem 4elt.xyz. Problems g_28060 and g_108196 are highly unstructured single Gaussians of 28060 and 108196 vertices respectively. The program takes as input the coordinates of the points and the value of the vector at each point. The domain of simulation is initially partitioned into $\sqrt{p} \times \sqrt{p}$ partitions. The nodes are distributed to the various processors based on this partitioning. The Barnes-Hut tree is constructed and load is balanced assuming that each particle has an equal computational load associated with it. It can be seen that after this load balancing phase, each processor has an equal number of nodes. After this reassignment of particles, the algorithm executes a complete mat-vec. The number of interactions in this iteration are now used to balance the load more accurately. The mat-vec is now computed and the timings are recorded. All timings reported in this section correspond to the second mat-vec iteration. This is indeed reasonable because in most applications, one needs to compute the mat-vec repeatedly and therefore, load information from the previous iteration can be used to balance load.

We had noted in Section 2 that the Green's function for the Laplace equation are given by $1/||x - x'||$ in three dimensions and $-\log(||x - x'||)$ in two dimensions. These functions can be expressed as:

$$\frac{1}{x - x'} = \frac{1}{x} \sum_{k=0}^{\infty} (\frac{x'}{x})^k \tag{2}$$

$$\log(x - x') = \log(x) - \sum_{k=1}^{\infty} \frac{1}{k} (\frac{x'}{x})^k \tag{3}$$

Both of these multipole expansions converge for $|x| > |x'|$. By truncating the series at different degrees, the accuracy and computation time can be traded off. In our code, we use the expansion corresponding to Equation 3, although, as can be seen, it is easy to incorporate other expansions. The translation of a multipole series centered at point x' to point x can be derived similarly [6].

We now discuss the specific results of our experimental study.

5.1 Parallel Efficiency and Overheads

We present the parallel runtime and efficiency of four different problem instances. It is impossible to run these instances on a single processor because of their memory requirements. Therefore, we use the force evaluation rates of the serial and parallel versions to compute the efficiency. To compute the MFLOP ratings of our code, we count the number of floating point operations inside the force computation routine and in applying the MAC to internal nodes. In our code, each force computation requires $13+k*16$ floating point instructions, where k is the degree of polynomial used. The MAC routine requires 14 floating point instructions. The square root instruction is assumed to be a single floating point instruction. Using this and the number of MACs and force computations, we determine the total number of floating point operations executed by the code. This is divided by the total time to obtain MFLOP rating of the code. Table 1 presents the runtimes, efficiencies, and computation rates for four problems. The value of the α parameter in each of these cases is 0.7, and the degree of the multipole expansion is 6. The efficiencies were computed by determining the sequential time for each MAC and force computation. The sequential times for the larger problem instances were projected using these values and the efficiencies computed.

Table 1: Runtimes (in seconds), efficiency, and computation rates for different problems for $p = 64$ and 256.

Problem	$p = 64$			$p = 256$		
	Runtime	Eff.	MFLOPS	Runtime	Eff.	MFLOPS
4elt.xyz	3.41	0.79	122	1.18	0.57	353
pscan	27.35	0.92	142	7.51	0.84	517
g_28060	3.99	0.86	133	1.25	0.69	425
g_108196	16.28	0.81	125	4.64	0.71	440

A number of observations can be made from Table 1. Our parallel formulation yields excellent performance for a wide range of processors. The speedup scales almost linearly unto 256 processors even for very small problems (4elt.xyz and g_28060). For the range of problem sizes where hierarchical methods are really useful (pscan and g_108196), the formulation yields a relative speedup of over 3.5 on going from 64 to 256 processors. Clearly, for these problem instances, the overhead due to communication and load imbalance is low.

Our parallel implementation does *not* make use of the vector units and this reflects in the low per-processor FLOPS rating. We are in the process of restructuring our code to make use of the vector units. It may be argued that using vector units would reduce computation time while keeping communication time constant, thus leading to poorer efficiencies. However, it must be noted that using vector units would allow us to solve significantly larger problems. We have shown here that the communication scales slower than useful computation. Therefore, using vector processors and solving larger problem instances (in the same wall-clock time of a few seconds) would allow us to achieve similarly high efficiency. We expect to demonstrate this in the final version of the paper presented at the conference.

5.2 Impact of Polynomial Degree on Performance and Accuracy

We run two sets of problems with different degree polynomials for $p = 64$ and $p = 256$. Fixing the α parameter at 0.7, we note the parallel runtime and compute corresponding efficiencies and fractional error. Since it is difficult to compute an exact mat-vec, we compute the fractional error relative to the mat-vec computed by a degree 14 polynomial. This was the largest degree polynomial we could use for some problems because of memory constraints. If x_k is the mat-vec returned by the k-degree polynomial approximation and x is the mat-vec returned by the degree 14 polynomial, then the fractional error is defined as $||x - x_k||/||x||$. When expressed as a percentage, we refer to this as the fractional percentage error of the mat-vec. Table 2 shows the parallel runtime, efficiency, and fractional percentage errors of several instances.

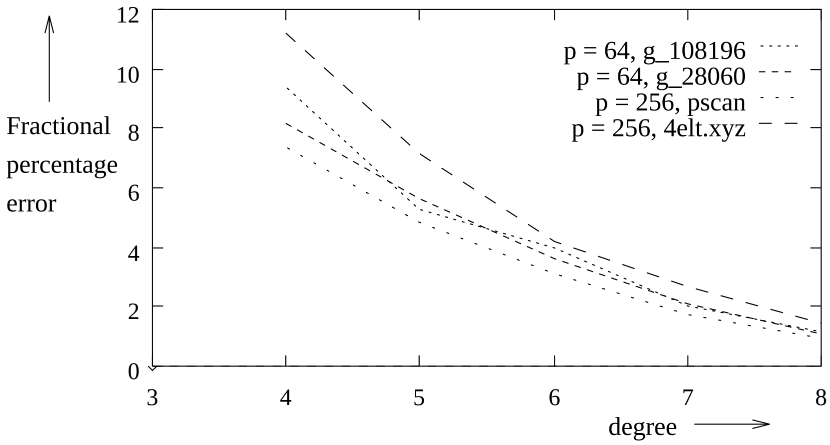
Table 2: Runtimes (in seconds), efficiency, and fractional percentage errors for different degree polynomials.

Problem	degree = 4			degree = 6			degree = 8		
	Time	Eff.	Error	Time	Eff.	Error	Time	Eff.	Error
$p = 64, \alpha = 0.7$									
g_28060	3.48	0.78	8.14	3.99	0.86	3.63	4.68	0.89	1.08
g_108196	14.24	0.75	9.41	16.28	0.81	3.97	18.28	0.85	1.13
$p = 256, \alpha = 0.7$									
4elt.xyz	1.03	0.54	11.19	1.18	0.57	4.20	1.38	0.64	1.41
pscan	6.35	0.73	7.40	7.51	0.84	3.11	8.31	0.89	0.97

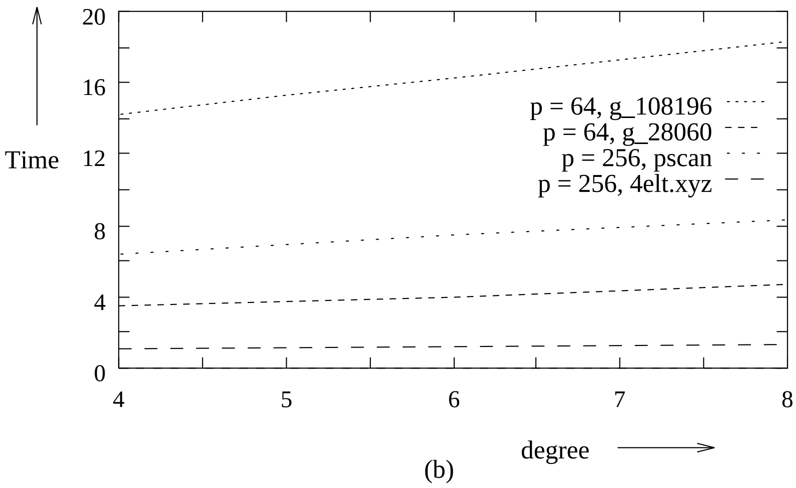
Figure 4 presents the corresponding graphs depicting the drop in fractional percentage error and increase in parallel runtime with increase in the degree of the polynomial. It can be seen that whereas the reduction in error begins to level off beyond degree 6 polynomials, the parallel runtime increases almost linearly. This implies that it is important to determine the right error tolerance for the mat-vec. Another observation that can be made from Table 2 is that the efficiency of the parallel formulation increases with the degree of polynomial. This is consistent with our expectations since the communication remains constant but computation increases with the degree of the polynomial.

5.3 Impact of α-Parameter on Performance and Accuracy

We now study the impact of changing the α parameter. We run the same problem instances. In this case, the degree of the polynomial is fixed at 6 and the value of α is varied between 0.7 and 1.1. Once again, the parallel efficiency and the fractional percentage error are computed. As before, the fractional error is computed with respect to $\alpha = 0.7$ for a 14 degree polynomial. The results are presented in Table 3.



(a)



(b)

Figure 4: Variation of fractional percentage error and parallel runtime with increase in degree of polynomial.

Table 3: Runtimes (in seconds), efficiency, and fractional percentage errors for different values of α.

Problem	$\alpha = 0.7$			$\alpha = 0.9$			$\alpha = 1.1$		
	Time	Eff.	Error	Time	Eff.	Error	Time	Eff.	Error
$p = 64$, degree = 6									
g_28060	3.99	0.86	3.63	3.01	0.87	4.46	2.45	0.87	12.81
g_108196	16.28	0.81	3.97	10.65	0.84	4.90	8.17	0.86	11.29
$p = 256$, degree = 6									
4elt.xyz	1.18	0.57	4.20	0.82	0.63	6.19	0.65	0.65	14.91
pscan	7.51	0.84	3.11	5.28	0.85	5.50	4.42	0.85	8.81

From this table, it can be noticed that for smaller values of α, the efficiency of the parallel formulation is lower. This can be attributed to the fact that communication volume and distances both increase as α is reduced. The overall time taken also increases since more interactions are computed as direct interactions than as multipole interactions.

Tables 2 and 3 illustrate some interesting trends. The gains from increasing the degree of the multipole expansion diminish but the corresponding runtime increases linearly. Therefore, it is desirable to identify the desired error level and the corresponding degree. Higher degree polynomials are more effective in reducing overall error than smaller values of α (for a given operation count). This is favorable to the parallel formulation presented in two ways:

- Lower values of α result in lower communication overhead.
- The performance of the parallel formulation improves with increasing degree of polynomial. This is not the case for data-shipping schemes.

It must be noted that the value of α must be lower than a certain threshold.

6 Conclusions and Future Work

In this paper, we have presented a parallel formulation of an augmented Barnes-Hut method for computing dense matrix-vector products. We have demonstrated excellent performance for our formulations even for very small problems. We have shown that the performance increases almost linearly up to 256 processors of a CM5 parallel computer. We have also studied the error properties of our algorithm and the interaction between errors, parallel runtimes, and various program parameters. We have shown that it is more effective to control error by increasing the degree of the multipole expansion than by increasing α. Furthermore, we have shown that our scheme is particularly amenable to higher degree approximations. To the best of our knowledge, this is among the first exhaustive application of hierarchical methods to computing mat-vecs in the parallel context.

We are currently porting this formulation to more state-of-the-art machines such as the Cray T3D. We are implementing an iterative solver based on this hierarchical matrix-vector product. This would enable us to solve domain discretizations of extremely high resolution.

7 Acknowledgements

The authors would like to thank Vivek Sarin for many useful discussions and suggestions.

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

[5] Ananth Grama, Vipin Kumar, and Ahmed Sameh. On n-body simulations using message passing parallel computers. In *Proceedings of the SIAM Conference on Parallel Processing*, San Francisco, 1995.

[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] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Algorithm Design and Analysis.* Benjamin/Cummings, Redwod City, 1994.

[10] 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.

[11] Keith Nabors and Jacob White. Fastcap: A multipole accelerated 3-d capacitance extraction program. *IEEE Transactions on Computer-Aided Design*, 10(11):1447–1459, Nov 1991.

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

[13] 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).

[14] 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.

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

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

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