AD-A199 804

DTIC
ELECTE
OCT 0 6 1988
S D
E

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

This document has been approved for public release and sale; its distribution is unlimited.

88 10 3 18 8

This paper presents a parallel version of the Fast Multipole Method (FMM). The FMM is a recently developed scheme for the evaluation of the potential and force fields in systems of particles whose interactions are Coulombic or gravitational in nature. The sequential method requires $O(N)$ operations to obtain the fields due to N charges at N points, rather than the $O(N^2)$ operations required by the direct calculation. Here, we describe the modifications necessary for implementation of the method on parallel architectures and show that the expected time requirements grow as $\log N$ when using N processors. Numerical results are given for a shared memory machine (the Encore Multimax 320).

A Parallel Version of the Fast Multipole Method

L. Greengard† and W. D. Gropp‡

Research Report YALEU/DCS/RR-640
August 1988

DTIC
ELECTE
OCT 0 6 1988
S D
E

Approved for public release: distribution is unlimited.

†The work of this author was supported in part by the Office of Naval Research under contract number N00014-86-K-0310 and by a National Science Foundation Postdoctoral Fellowship.
‡The work of this author was supported in part by the Office of Naval Research under contract N00014-86-K-0310 and the National Science Foundation under contract number DCR 8521451.

Keywords: *N-body problem, parallel computing, fast multipole method, potential theory*

1. Introduction

Numerical methods for computing N-body interactions generally fall into two categories. Continuum methods are based on the fact that the potential satisfies the Poisson equation and use fast Poisson solvers to obtain the field [6]. They are hindered by the limited resolution of the imposed grid and the degradation of performance seen with highly non-homogeneous distributions of particles. Hierarchical methods [1, 2] are based on the fact that the field due a cluster of particles can be represented at a great distance by the net mass acting at the center of mass. Tree structures are used to partition space and group particles at various length scales, so that the center of mass approximation can be applied. The CPU time requirements of these methods generally grow as $N \log N$. They handle non-homogeneous distributions better than the continuum methods, but yield only approximate results.

The Fast Multipole Method (FMM) [5, 3, 8, 7] shares certain characteristics with the hierarchical solvers. Tree structures are imposed to partition space, and the strategy is similar, but analytic observations concerning multipole and Taylor expansions are used to produce results that are accurate to within round-off error. The CPU time requirements are of the order $O(N \log(1/\epsilon))$, where ϵ is the desired accuracy.

In this paper, we will describe a parallel version of the non-adaptive two-dimensional FMM and present numerical results for an implementation on a shared memory machine (the Encore Multimax 320). We note that Zhao [9] has independently developed a parallel implementation of a non-adaptive three-dimensional multipole method for the Connection Machine.

2. Mathematical Preliminaries

In this paper, we will consider as a model the N-body problem in the complex plane $\mathbb{C}$. That is, given the positions z_i and strengths q_i of N charged particles, we wish to compute the net potential ϕ and electric field $\mathbf{E}$ at each particle position from Coulomb's law. These are given by the expressions

$$\phi(z_i) = Re\left(\sum_{j \neq i} q_i \cdot \log(z_i - z_j)\right)$$

and

$$E(z_i) = \left(-Re(\phi'(z_i)), Im(\phi'(z_i))\right) ,$$

respectively.

Suppose now that m charges with strengths q_i and positions z_i are located within a disk of radius r centered at the origin. Then, it is shown in [5], that for a point z with $|z| > r$, the potential $\phi(z)$ induced by the charges is given by a multipole expansion of the form

$$\phi(z) = Q \log(z) + \sum_{k=1}^{\infty} \frac{a_k}{z^k} , \tag{2.1}$$

where

$$Q = \sum_{i=1}^{m} q_i \qquad and \qquad a_k = \sum_{i=1}^{m} \frac{-q_i \cdot z_i}{k} .$$

The error in truncating the sum after s terms is

$$\left|\phi(z) - Q \log(z) - \sum_{k=1}^{s} \frac{a_k}{z^k}\right| \leq \left(\frac{A}{c-1}\right)\left(\frac{1}{c}\right)^s , \tag{2.2}$$

where

$$A = \sum_{i=1}^{m} |q_i| \qquad \textit{and} \qquad c = \left|\frac{z}{r}\right| . \tag{2.3}$$

In order to obtain a relative precision of ϵ (with respect to the total charge), the number of terms required in the series representation of ϕ is approximately $-\log_c(\epsilon)$, independent of m, the number of source charges. The Fast Multipole Method is based on making explicit use of this result.

2.1. Translation operators

In the FMM scheme, it is necessary not only to form multipole expansions as in (2.1), but to carry out a sequence of analytic transformations of the expansion coefficients. These transformations are described in the next three lemmas. Detailed proofs can be found in [5]. The first, Lemma 2.1, provides a mechanism for shifting the center of a multipole expansion.

Lemma 2.1. (Translation of a Multipole Expansion) *Suppose that*

$$\phi(z) = a_0 \log(z - z_0) + \sum_{k=1}^{\infty} \frac{a_k}{(z - z_0)^k} \tag{2.4}$$

is a multipole expansion of the potential due to a set of m charges of strengths $q_1, q_2, \ldots, q_m$, all of which are located inside the circle D of radius R with center at z_0. Then for z outside the circle D_1 of radius $(R + |z_0|)$ and center at the origin,

$$\phi(z) = a_0 \log(z) + \sum_{l=1}^{\infty} \frac{b_l}{z^l}, \tag{2.5}$$

where

$$b_l = -\frac{a_0 z_0^l}{l} + \sum_{k=1}^{l} a_k z_0^{l-k} \binom{l-1}{k-1}, \tag{2.6}$$

with $\binom{l}{k}$ the binomial coefficients. Furthermore, for any $s \geq 1$,

$$\left| \phi(z) - a_0 \log(z) - \sum_{l=1}^{s} \frac{b_l}{z^l} \right| \leq \left(\frac{A}{c-1}\right) \left(\frac{1}{c}\right)^s , \tag{2.7}$$

where A is defined in (2.3) and

$$c = \left| \frac{z}{|z_0| + R} \right| .$$

Lemma 2.2 describes the conversion of a multipole expansion into a local (Taylor) expansion inside a circular region of analyticity.

Lemma 2.2. (Conversion of a Multipole Expansion into a Local Expansion) *Suppose that m charges of strengths $q_1, q_2, ..., q_m$ are located inside the circle D_1 with radius R and center at z_0, and that $|z_0| > (c+1)R$ with $c > 1$. Then the corresponding multipole expansion (2.4) converges inside the circle D_2 of radius R centered about the origin. Inside D_2, the potential due to the charges is described by a power series:*

$$\phi(z) = \sum_{l=0}^{\infty} b_l \cdot z^l, \tag{2.8}$$

where

$$b_0 = a_0 \log(-z_0) \; + \; \sum_{k=1}^{\infty} \frac{a_k}{z_0^k}(-1)^k, \tag{2.9}$$

and

$$b_l = -\frac{a_0}{l \cdot z_0^l} \; + \; \frac{1}{z_0^l} \sum_{k=1}^{\infty} \frac{a_k}{z_0^k} \binom{l+k-1}{k-1} (-1)^k \; , \qquad \text{for } l \geq 1. \tag{2.10}$$

Furthermore, for any $s \geq max\left(2, \frac{2c}{c-1}\right)$, *an error bound for the truncated series is given by*

$$\left| \phi(z) - \sum_{l=0}^{s} b_l \cdot z^l \right| < \frac{A(4e(s+c)(c+1)+c^2)}{c(c-1)} \left(\frac{1}{c}\right)^{s+1}, \tag{2.11}$$

where A *is defined in (2.3) and* e *is the base of natural logarithms.*

Lemma 2.3 provides a formula for shifting the center of a local expansion within a region of analyticity. This translation is exact, and no error bound is needed.

Lemma 2.3. **(Translation of a Local Expansion)** *For any complex* z_0, z *and* $\{a_k\}$, $k = 0, 1, 2, \ldots, n$,

$$\sum_{k=0}^{n} a_k (z - z_0)^k = \sum_{l=0}^{n} b_l \cdot z^l \tag{2.12}$$

where

$$b_l = \sum_{k=l}^{n} a_k \binom{k}{l} (-z_0)^{k-l} \tag{2.13}$$

3. Informal Description of the FMM

In this section, we briefly outline the sequential FMM procedure. A more detailed discussion is available in [5, 7]. The algorithm uses a divide and conquer strategy to cluster particles at various levels of spatial discretization, and then uses multipole and Taylor expansions to evaluate the interactions between distant clusters. Once all distant interactions are accounted for by this expansion technique, the interactions between neighboring particles are computed by the direct application of the pairwise force law.

We now introduce the notation necessary for a description of the algorithm. Since we are considering the non-adaptive scheme, we assume that N charges are more or less homogeneously distributed within a square with sides of length one, and refer to this square as the computational box. We impose a hierarchy of meshes on the computational box which refine it into smaller and smaller regions. More specifically, mesh level 0 refers to the entire computational box, while mesh level $l+1$ is obtained recursively from level l by subdividing each box into four equal parts. A tree structure is imposed on this hierarchy, so that if *ibox* is a box at level l, then the four boxes at level $l+1$ obtained by its subdivision are considered its children. In general, the maximum number of refinements (the tree depth) is chosen to be on the order of $\log_4 N$, at which point there is on the order of 1 particle in each box at the finest level. For every box i at level l, we define the *nearest neighbors* to be the box itself and any other box at the same level with which it shares a boundary point. There are clearly at most 9 nearest neighbors.

Two boxes (at a given level) with sides of length D, are said to be *well-separated* if they are separated by a distance D. It is shown in [7] that, in using s-term expansions to account for

the interactions between *well-separated* boxes, the error bounds (2.2), (2.7) and (2.11) apply with $c = (4-\sqrt{2})/\sqrt{2} \approx 1.8$. For a given precision ϵ, we therefore choose $s = \lceil -\log_c(\epsilon) \rceil$.

Both multipole and local expansions are associated with each box. $\Phi_{l,i}$ is the s-term multipole expansion about the center of box i at level l which describes the far field potential due to the particles contained inside the box. $\Psi_{l,i}$ is the s-term local expansion about the center of box i at level l which describes the potential field due to all particles outside the box and its nearest neighbors. $\tilde{\Psi}_{l,i}$ is the s-term local expansion about the center of box i at level l which describes the potential field due to all particles outside i's *parent* box and the *parent* box's nearest neighbors. Finally, an *interaction list* is associated with each box i at level l. This is the set of boxes which are children of the nearest neighbors of i's parent and which are well-separated from box i.

The algorithm computes interactions between groups of particles at the coarsest possible mesh level. Two passes are executed.

Initialization Choose a level of refinement $n \approx \lceil \log_4 N \rceil$, a precision ϵ, and set $s = \lceil -\log_c(\epsilon) \rceil$. Assign particles to boxes at finest mesh level.

Upward Pass

Step 1: Form multipole expansion $\Phi_{n,i}$ about the box center for each box i at finest mesh level. Uses equation (2.1).

Step 2: Recursively form multipole expansions about the centers of all boxes at all coarser mesh levels, each expansion representing the potential field due to all particles contained in the box. Uses Lemma 2.1.

In the downward pass, the local expansions $\Psi_{l,i}$ are formed for all boxes, beginning at the coarsest level. This process is somewhat more complex. Suppose, however, that at level $l-1$, the local expansion $\Psi_{l-1,i}$ has been computed. Then Lemma 2.3 can be used to shift the expansion to each of the box's children. For each child box j at level l, what we have obtained is a local representation of the field due to all particles outside the parent's nearest neighbors, namely $\tilde{\Psi}_{l,i}$. The interaction list defined above is precisely the set of boxes whose contribution to the potential must be added to $\tilde{\Psi}_{l,i}$ to create $\Psi_{l,i}$. The initialization of this pass is simple. Since there are no well-separated boxes at level 0 or 1, we may set $\Psi_{0,i}$, $\tilde{\Psi}_{1,i}$, $\Psi_{1,i}$ and $\tilde{\Psi}_{2,i}$ to zero.

Downward Pass

Steps 3,4: Begin at level 2, and proceed to finer levels as follows: form $\Psi_{l,ibox}$ by using Lemma 2.2 to convert the multipole expansion $\Phi_{l,j}$ of each box j in the *interaction list* of box *ibox* to a local expansion about the center of box *ibox*, adding these local expansions together, and adding the result to $\tilde{\Psi}_{l,ibox}$. If finest level has been reached, process is complete. Otherwise form the expansion $\tilde{\Psi}_{l+1,j}$ for *ibox*'s children by using Lemma 2.3 to expand $\Psi_{l,ibox}$ about the children's box centers and continue procedure.

Step 5: Evaluate local expansions at particle positions to obtain the far-field potential and/or force.

Step 6: Compute potential (or force) due to particles in nearest neighbor boxes directly.

Step 7: For every particle, add direct and far-field terms together.

4. Description of the parallel algorithm

The algorithm described in the previous section has several opportunities for parallelism. The best of these opportunities are the completely parallel operations such as the computation of the initial moments and the evaluation of the local expansions for each particle. The other parallel operations require some coordination between processors. For example, evaluation of the forces between particles in neighboring boxes in step 6 requires secure data access if Newton's third law

N	T_{alg} $p=1$	T_{dir} $p=1$	T_{alg} $p=16$	T_{dir} $p=16$	Speedup Alg	Speedup Dir
625	14	54	1.2	3.45	11.7	15.7
1250	52	216	3.6	13.9	14.4	15.5
2500	68	872	4.6	54.9	14.7	15.9
5000	235	3490	15.5	220.8	15.2	15.9
10000	301	14020	19.7	910.4	15.3	15.4
20000	1008	56385	65.0	3560.4	15.5	15.8

Table 1: Table of times for algorithm (alg) and direct method (dir) on Encore Multimax 320. All times in seconds.

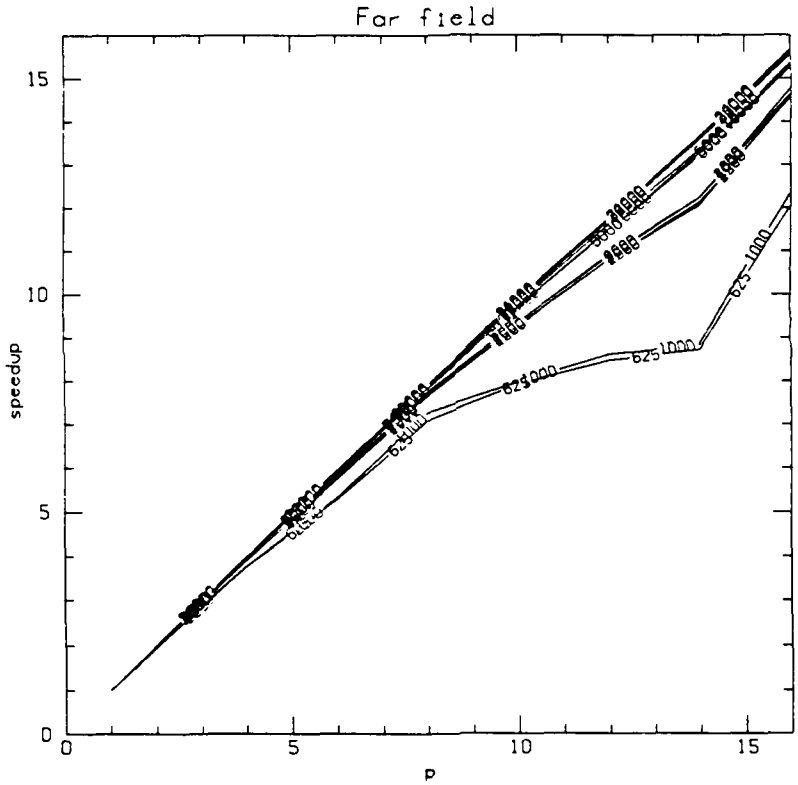

Figure 1: Speedup for the calculation of the far-field by the Fast Multipole Method. The labels are the number of particles.

Accession For
NTIS GRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

is used. However, it is the *reductions*, the communications between mesh levels, that cause the greatest difficulty.

The fact that the entire program is not completely parallel opens the question of how efficient the algorithm can be, particularly on a large number of processors. We will address this by analyzing the computational complexity of the parallel algorithm. We do not discuss the initialization of the

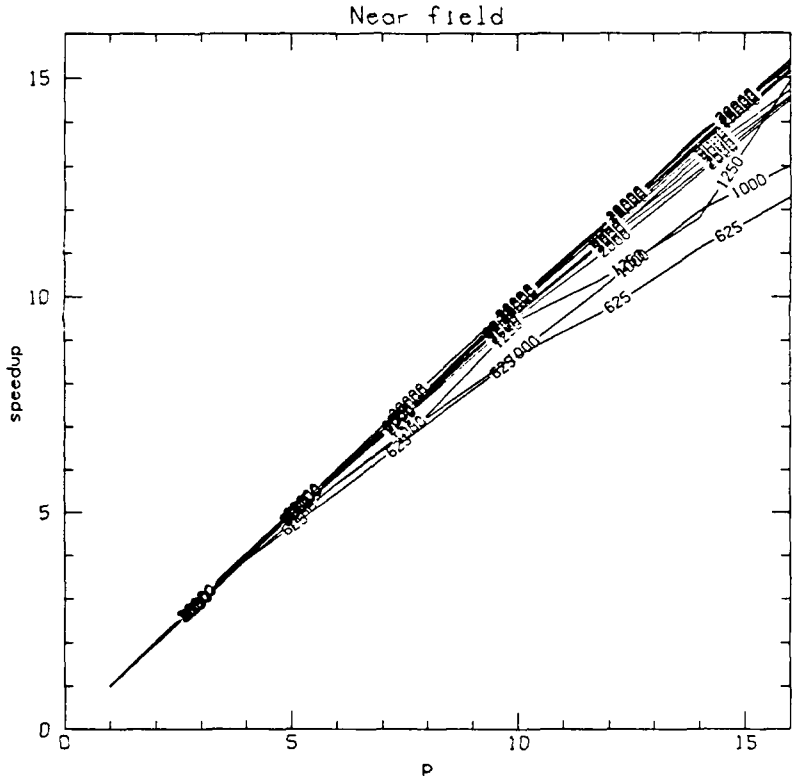

Figure 2: Speedup for the near-field calculation (step 6). The labels are the number of particles.

algorithm since the initialization is essentially a parallel sort and is performed only at the beginning of a computation. Since a common use of the Fast Multipole Method is to compute the forces at each time step in a dynamical simulation, this initial sort may be amortized over all the time steps. Further, in a time dependent calculation, it is possible to exploit slowly varying changes in the potential to reduce the amount of computation; we will not consider this effect either.

For the actual implementation, we can consider each of the steps in the algorithm separately. We use the term "communication" to denote any coordination between processors. In a message passing system, this would be a message; in a shared memory system, this would be some critical section (e.g., a spinlock). We use N to denote the number of particles, n the number of levels, and p the number of processors. Let B denote the average number of particles per box at the finest level. Then $N = B4^n$.

Upward Pass

Step 1: Formation of expansions at finest level. There is no communication; the complexity is N/p.
Step 2: Merge upward. Communication is within box and with parent box. The complexity is

$$\sum_{i=0}^{n-1} \left\lceil \frac{4^i}{p} \right\rceil = \sum_{i=\log_4 p}^{n-1} \left\lceil \frac{4^i}{p} \right\rceil + \sum_{i=0}^{\log_4 p - 1} 1$$

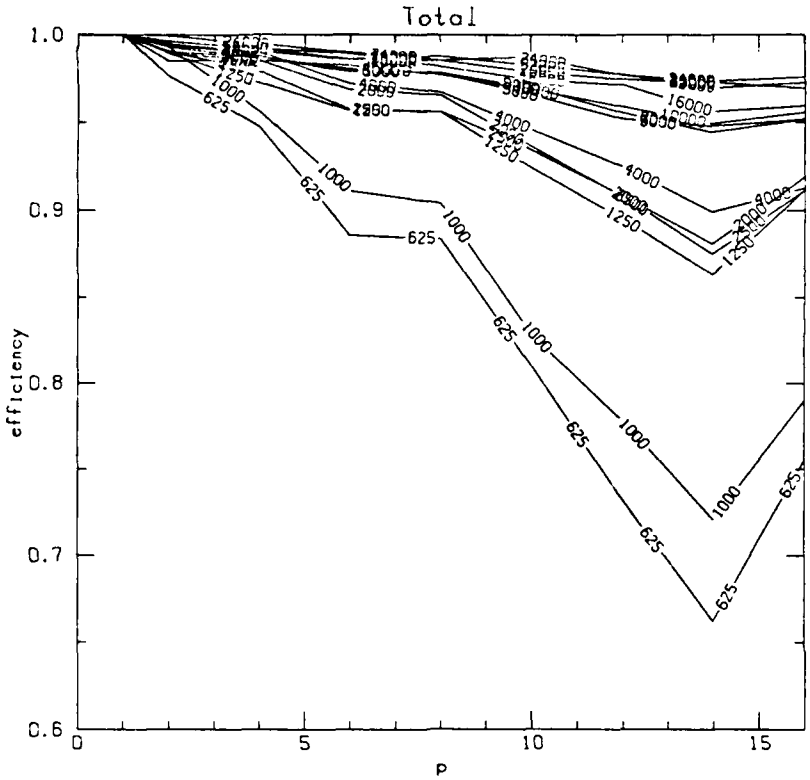

Figure 3: Parallel efficiency for the full algorithm. The labels are the number of particles.

times the cost per box. This is one of the most important terms because it gives a limit on the available parallelism. The second sum shows the bottleneck in the reduction: when there are fewer than p boxes, some processors go idle. This bottleneck is an essential part of the algorithm.

Downward pass

Steps 3,4: Convert the multipole expansions into local expansions and move down. The complexity is of the same form as above (but with a different constant).

Step 5: Evaluation of local expansions at the finest level. This is perfectly parallel and has complexity N/p.

Step 6: Compute the potential or force due to particles in neighboring boxes directly. This involves the neighboring boxes through direct interaction. There is no "reduction" overhead; however, there is some communication due to the fact that the field at a particle position may be updated by several adjacent processors/particles. The complexity is roughly $4^n B^2/p = NB/p$. If Newton's third law is not used, this is completely parallel (i.e., we can eliminate any possibility of memory contention at the cost of twice as much arithmetic).

Step 7: Add the components (direct and far-field) together at each particle. This is perfectly parallel; the complexity is N/p.

Summing the contributions from each step, the overall complexity is

$$T = \frac{aN}{p} + b\log_4 p + \frac{cN}{Bp} + \frac{dNB}{p} + e(N,p) \tag{4.1}$$

where a, b, c, and d are constants determined by the floating point speed and the requested precision, and e is a lower order term which includes things like the communication or synchronization overhead.

One important observation to be made concerns the choice of the parameter B which determines the number of refinement levels n. For the uni-processor case, this is discussed in [3]. In the parallel case, there is a temptation to use $B = 1$. While a value for B of 1 has the advantage of placing (at most) one particle per processor, this can result in a serious loss in efficiency. The optimal value of B, denoted B_{opt}, for a given N and p, is the one that minimizes the time T in (4.1):

$$\frac{dT}{dB} = -\frac{cN}{B^2 p} + \frac{dN}{p} = 0,$$

or

$$B_{opt} = \sqrt{\frac{c}{d}}. \tag{4.2}$$

While the exact values for c and d are difficult to determine (they depend on the floating point rates, memory speed, and details of the coding and the compiler used), c is roughly proportional to $25s^2$, where s is the number of terms in the expansion, while d is roughly 9, the number of neighboring boxes. The optimal blocking is therefore $B_{opt} \approx 2s$. For single precision accuracy, $s \approx 15$, so that $B_{opt} \approx 30$ and the execution time for $B = 1$ is about 15 times slower. Note that B_{opt} is independent of the number of processors, and should be used for both parallel and sequential implementations.

It is clear from (4.1) that with $\mathcal{O}(N)$ processors, the overall complexity is $\mathcal{O}(\log N)$. The parallel efficiency can be estimated as the ratio of the time on a single processor (without the overhead) to p times the time on p processors:

$$\begin{aligned} \text{efficiency} &= \frac{\alpha N}{p\left(\frac{\alpha N}{p} + b\log_4 p + e(N,p)\right)} \\ &= \frac{1}{1 + \frac{pb}{\alpha N}\log_4 p + \frac{pe}{\alpha N}} \\ &\approx 1 - \frac{pb}{\alpha N}\log_4 p - \frac{pe}{\alpha N}, \end{aligned} \tag{4.3}$$

where $\alpha = a + c/B + dB$.

It is interesting to look at the behavior of equation (4.3) for the cases $B = 1$ and for $B = B_{opt}$, with $p = N/B$ (i.e., one box at the finest level per processor). We do not consider using more than one processor per box. In the first case, the efficiency is roughly

$$1 - \frac{b}{a}\log_4 N$$

where we have neglected the overhead term $e(p, N)$. With the optimal B, the efficiency is

$$1 - \frac{b}{aB_{opt}}\log_4 \frac{N}{B_{opt}}.$$

To give an idea of what this means, assume that the $B = 1$ case gives 90% efficiency, and let $B_{opt} = 10$ (this is quite conservative). Then the choice $B = B_{opt}$ will achieve more than 99% efficiency. Moreover, if the $B = 1$ case gives 10% efficiency, the choice $B = B_{opt}$ will achieve more than 90% efficiency (assuming that the parallel overhead terms $\epsilon(p, N)$ are negligible).

The above analysis gets even worse if we consider the case of true speedup, defined as

$$\frac{\text{Time for best uni-processor algorithm}}{\text{Time for parallel algorithm}}.$$

In this case, the efficiency for $B = 1$ is roughly $2/B_{opt}$ times the formula in equation (4.3). In short, for good processor utilization, an implementation of the Fast Multipole Method should use the optimal blocking factor B_{opt}.

Our last consideration is given to minimizing the total time T, given a fixed number of particles N but varying the number of processors p. In this case, the minimum time depends in a complicated way on the various parameters, but is achieved at roughly $p = N$. This can be seen from the efficiency figures above. For example, if the case $B = 1$ gave 90% parallel efficiency relative to a sequential program with $B = 1$, it would give 18% efficiency relative to a sequential program with $B_{opt} = 10$. Each processor is busy roughly one fifth of the time. However, there are 10 times as many processors for $B = 1$ as for $B = 10$, so the total time for $B = 1$ and $p = N$ is roughly half that for $B = 10$ and $p = N/B$. Thus, even with the loss of efficiency, the $B = 1$ case has a smaller absolute time. This fact is of interest *only* when there are enormous numbers of processors available, *and* when it is impractical to use the excess processors to work together on the computations for each box.

4.1. Comments on the parallel implementation

Our parallel implementation is based on a version of the serial code described in [5]. The implementation is a "minimum distance" change, and does not attempt to rearrange the computation to be more parallelizable. In particular, it is possible to identify subclasses of boxes for which completely parallel operations may be performed; within these classes it can be proven that no data-access conflicts can occur [4]. This can reduce the overhead in steps 2, 3, and 4 by reducing the number of memory locks (in a shared memory implementation) or the number of messages (in a message passing implementation).

5. Experimental Results

These results are for the non-adaptive algorithm, and do not include the work of the initial sorting of the particles. The data in Table 1 were obtained on an Encore Multimax 320 with 18 processors. There are two important points to remember in interpreting these times. One is that they were taken on a time sharing system; even though no other users were present, various daemons will consume some resources. To reduce this effect, we used only 16 of the 18 processors in our experiments. The second is the effect of the choice of number of levels. While the complexity estimates predict time linear in the number of particles, in fact the actual times display a "ratchet" behavior as the number of levels increase. However, over a large enough range of number of particles, the behavior is linear.

Figures 1–3 show a breakdown of the results for the Encore Multimax 320. Figure 1 shows the speedup for the calculation of the far field by the Fast Multipole Method. Note that the results are clustered into four groups; these represent the number of levels (4–7). The speedup is lower when the number of processors is not a power of four, a result of the poor load-balancing in the reduction stages (steps 2–4). Figure 2 shows the speedup for the calculation of the near field in step 6. The deviation from perfect speedup is due mainly to the overhead connected with secure access to data (critical sections). Figure 3 shows the overall efficiency. Note that even for small

numbers of particles 75% efficiency is achieved and for 5000 or more particles, 95% efficiency is achieved.

A version of the three-dimensional multipole method has been implemented and studied by Zhao [9]. His results show the predicted $\log N$ growth as N ran from 64 to 16384. His timings, done on the Connection Machine, are somewhat slow. Different formulations of the algorithm presented here, in particular with respect to constant terms or terms in $-\log \epsilon$, should significantly reduce the timings.

6. Conclusions

Our results have shown that the Fast Multipole Method is very suitable for shared memory parallel computers. Both our experience and the results of Zhao indicated that it is suitable for message passing parallel computers as well. The overall complexity of the N-body calculation (with $p = N$ processors) is $\log N$; for fixed N it is $N/p + \log p$.

The non-adaptive algorithm described here has very regular memory access or communication patterns which can be exploited to reduce the parallel overhead. Many of these are intrinsic to the Fast Multipole Method itself, and should be exploitable by the adaptive version.

References

[1] A. W. Appel, *An Efficient Program for Many-body Simulation,* Siam. J. Sci. Stat. Comput., 6 (1985), pp. 85–103.

[2] J. Barnes and P. Hut, *A Hierarchical $O(N \log N)$ Force-Calculation Algorithm,* Nature, 324 (1986), pp. 446–449.

[3] J. Carrier, L. Greengard, and V. Rokhlin, *A Fast Adaptive Multipole Algorithm for Particle Simulations,* Siam J. Sci. Stat. Comput., 9 (1988), pp. 669–686.

[4] L. Greengard and W. Gropp, *The Fast Multipole Method on Vector Computers,* in preparation.

[5] L. Greengard and V. Rokhlin, *A Fast Algorithm for Particle Simulations,* J. Comput. Phys., 73 (1987), pp. 325–348.

[6] R. W. Hockney and J. W. Eastwood, *Computer Simulation Using Particles,* McGraw-Hill, New York, 1981.

[7] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems,* MIT Press, Cambridge, 1988.

[8] L. Greengard and V. Rokhlin, *Rapid Evaluation of Potential Fields in Three Dimensions,* Technical Report 515, Yale Computer Science Department, 1987.

[9] F. Zhao, *An O(N) Algorithm for Three-dimensional N-body Simulations,* Technical Report 995, Massachusetts Institute of Technology, 1987.