A data parallel implementation of hierarchical N-body methods

Citation

Hu, Yu and S. Lennart Johnsson. 1994. A data parallel implementation of hierarchical N-body methods. Harvard Computer Science Group Technical Report TR-26-94.

Permanent link

<http://nrs.harvard.edu/urn-3:HUL.InstRepos:34325483>

Terms of Use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA>

Share Your Story

The Harvard community has made this article openly available.
Please share how this access benefits you. [Submit a story](#).

[Accessibility](#)

A Data Parallel Implementation of Hierarchical N -body Methods

Yu Hu
S. Lennart Johnsson

TR-26-94
September 1994

Parallel Computing Research Group
Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

A Data Parallel Implementation of Hierarchical N -body Methods

Yu Hu and S. Lennart Johnsson*

Aiken Computation Lab

Harvard University

Cambridge, MA 02138

Abstract

The $O(N)$ hierarchical N -body algorithms and Massively Parallel Processors allow particle systems of 100 million particles or more to be simulated in acceptable time. We describe a data parallel implementation of Anderson's method and demonstrate both efficiency and scalability of the implementation on the Connection Machine CM-5/5E systems. The communication time for large particle systems amounts to about 10–25%, and the overall efficiency is about 35%. On a CM-5E the overall performance is about 60 Mflop/s per node, independent of the number of nodes.

1 Introduction

The problem of computing the force (or the potential) exerted on one another by a system of electrical charges (or masses interacting gravitationally) has been widely studied and has applications in areas such as celestial mechanics, plasma physics and molecular dynamics. Algorithms that compute the forces for a system of N particles in $O(N)$ operations have been devised by Greengard and Rokhlin [6, 14, 12], Zhao [35], and Anderson [1]. The constant of proportionality is in the range 1,000–10,000. Other hierarchical algorithms, such as those proposed by Appel [4] and by Barnes and Hut [5], require $O(N \log N)$ operations, but with a smaller constant of proportionality. (Appel's method was later proved to be of order $O(N)$ [10].)

In many N -body applications in astrophysics and molecular dynamics, but also in boundary element methods where the fast N -body algorithms can be used, it is highly desirable to model systems with large numbers of particles. Massively Parallel Processors (MPPs) offer the primary storage and compute power for the simulation of systems with several hundred million particles by fast algorithms. Large state-of-the-art systems would allow for the simulation of systems with up to a billion particles using secondary storage.

In this paper, we describe a data parallel implementation of Anderson's method for N -body simulations. The implementation is made in Connection Machine Fortran [31], CMF. All but one of the features of CMF that we use are also available in High Performance Fortran [11], HPF. Data motion is managed through the use

* Also affiliated with Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142.

of data distribution directives and control of the storage-to-sequence association in mapping arrays to the MPP memory units. Additional performance gains are achieved through aggregation of computations, and by a careful trade-off between communication and redundant computation.

Our novel contributions to the implementation of hierarchical N -body methods on MPPs are

- minimal data motion in parent-children interactions,
- minimal data motion in neighbor interactions for interactive-field computations,
- redundant computation/communication trade-offs,
- representing translation operations as matrix-vector multiplications,
- aggregating multiple independent translation operations into multiple instances of matrix-matrix multiplications,
- reducing the number of translation operations through the use of supernodes,
- expressing hierarchical methods in an efficient way in a data parallel language,
- efficiency in memory usage.

Table 1 gives a summary of sequential and parallel implementations of hierarchical N -body methods. Barnes and Hut's $O(N \log N)$ method has been implemented using the message passing programming paradigm by Salmon et. al. [27, 33, 34] on the Intel Touchstone Delta and by Liu [23] on the CM-5. Salmon et. al. achieved efficiencies in the range 24 – 28%, while Liu using assembly language for time critical kernels achieved 30% efficiency. Zhao and Johnsson developed a data-parallel implementation of Zhao's method on the CM-2, and achieved an efficiency of 12% for expansions in Cartesian coordinates, resulting in more costly multipole expansion calculations. Leathrum and Board [21, 18] and Elliott and Board [9] achieved efficiencies in the range 14 – 20% in implementing Greengard-Rokhlin's method (see [13]) on the KSR-1. Schmidt and Lee [28] vectorized this method for the Cray YMP and achieved an efficiency of 39% on a single processor. Singh et. al. [29, 30] has implemented both $O(N \log N)$ and $O(N)$ methods on the Stanford DASH machine, but no measures of the achieved efficiency is available. Nyland et al. [26] discussed how to express the three-dimensional adaptive version of the Greengard-Rokhlin method in a data-parallel subset of the Proteus language, which is still under implementation on parallel machines. For comparison, we have also included the results reported in this paper. Additional details of the various implementations are given in Table 1.

In the next Section we will briefly describe hierarchical N -body methods and define several of the terms used in the above summary of contributions. Section 3 describes Anderson's multipole method in some detail. Section 4 presents the architecture of the Connection Machine system CM-5/5E. Our optimization techniques for programming hierarchical methods in CMF (HPF) are presented in Sections 5 – 10. Section 11 reports the performance results of our implementation. Section 12 discusses load-balancing issues and finally, Section 13 concludes the paper.

Author	Method & error	Prog. model	N	t (sec.)	P	Efficiency	Machine
Salmon [27]	BH, quadrupole	msg. passing					Ncube
Warren-Salmon-92 [33]	BH, quadrupole	msg. passing	8.78M	77	512	26%	Intel Delta
Warren-Salmon-93 [34]	BH, $\epsilon_1 = 10^{-3}$	msg. passing	8.78M	114	512	28%	Intel Delta
Liu [23]	BH, quadrupole	msg. passing	10M	59	256	30%	CM-5
Singh et al. [30]	BH						DASH, KSR-1
Leathrum-Board [22, 18]	GR, p=8		100,000	500	1	65%	IBM RS/6000-360
	GR, p=8		1M	1520	32	20%	KSR-1
Elliott-Board [9, 18]	GR, FFT, p=8		100,000	400	1	73%	IBM RS/6000-360
	GR, FFT, p=8		1M	1420	32	14%	KSR-1
Schmidt-Lee [28]	GR, p=8		40,000	94	1	39%	Cray Y-MP 8/864
	GR, p=16		40,000	311	1	22%	Cray Y-MP 8/864
Zhao-Johnsson [36]	Zhao, p=3	data parallel	16,000	5	8K	12%	CM-2
Hu-Johnsson	Anderson's method	data parallel	100 M	180	256	27 - 35%	CM-5E
Singh et al. [29]	GR,2D,adaptive						DASH, KSR-1
Nyland et al. [26]	GR,3D,adaptive						

Table 1: A summary of sequential and parallel implementations of hierarchical methods. All performance numbers are for uniform particle distributions. Methods used are 3D, unless otherwise stated. ϵ_1 is the relative error to the mean acceleration of the system. Empty items imply unavailable data.

2 Hierarchical N -body Methods

Hierarchical methods [2, 5, 14, 12] for the N -body problem partition potentials into two parts:

$$\phi_{total} = \phi_{near-field} + \phi_{far-field}, \quad (1)$$

where $\phi_{near-field}$ is the potential due to nearby particles and $\phi_{far-field}$ is the potential due to faraway particles. How far “far away” is affects the error in the solution. The *near-field* is defined by all particles not in the far-field. The near-field is evaluated through the classical N -body technique of pairwise interactions, while the far-field is evaluated hierarchically.

For the far-field evaluation, a cluster of particles sufficiently far away from an evaluation point are represented by a single computational element, called *far-field potential representation*. Greengard-Rokhlin [12] use multipole expansions as computational elements, while Anderson uses Poisson’s formula. The computational elements exactly represent the field of a single particle placed at some point within the element, e.g., the geometric center of the domain of the cluster, or its center of gravity. The exact computational elements are represented by an infinite number of terms and hence, in practice, are approximated by elements represented by a finite number of terms. The computational complexity and the error are both functions of the number of terms used. Using computational elements for the far-field potential, hierarchical methods transform the interactions among particles that are far away from each other into interactions among particles and computational elements (in the BH method) [5] or among computational elements (in the fast multipole methods) [2, 12].

A naive use of computational elements for clusters of particles leads to an $O(N\sqrt{N})$ algorithm. For example, for a uniform particle distribution and a decomposition of the domain into $\sqrt{N}$ subdomains, each having $\sqrt{N}$ particles, forming the computational elements for the subdomains takes $O(N)$ operations; $O(1)$ operations are

Figure 1: Recursive domain decompositions, the near-field, and the interactive-field in two dimensions.

required to compute the contribution of a particle to the computational element to which it belongs. For each particle, we need to evaluate $O(\sqrt{N})$ computational elements in the far-field and the total number of operations for the far-field evaluation is $O(N\sqrt{N})$. Evaluating near-field interactions also require $O(N\sqrt{N})$ operations, since there are $O(\sqrt{N})$ particles in the near-field of each particle.

In the hierarchical methods the computational domain is refined into smaller and smaller domains through a hierarchy of meshes. Mesh level 0 represents the entire domain (*box*). Mesh level $l + 1$ is obtained from level l by subdividing each subdomain at level l (*parent box*) into four (in two dimensions) or eight (in three dimensions) equally sized subdomains (*child boxes*). The number of distinct boxes at mesh level l is 4^l and 8^l for two and three dimensional nonadaptive methods, respectively. In an adaptive method, only subdomains with sufficiently many particles are further subdivided. Boxes that are not further subdivided are *leaves*. Hierarchical methods can be easily extended to rectangular domains in two dimensions and parallelepiped domains in three dimensions [2].

In the hierarchical methods, the idea is to form and use as few far-field computational elements as possible. In the approximation of the far-field of a cluster of particles by a single computational element, “far away” is measured relative to the size of the cluster. Close to an evaluation point, one can only combine particles in small domains but farther away from the evaluation point, combining can be done on particles in large domains. Hierarchical methods recursively combine the computational elements for small subdomains into computational elements for larger subdomains. Except for the leaf-level, the number of operations required to form computational elements is independent of the number of particles inside those domains. The number of operations required to form the leaf-level computational elements is $O(N)$ and that required for the hierarchical combining of computational elements is $O(M)$ with M leaf-level elements.

For uniform particle distributions and a hierarchy depth of $\log N$ the total number of computational elements in the hierarchy is $O(N)$ and there are only a few particles ($O(1)$) in each leaf-level computational element. The far-field potential evaluation for the $O(N \log N)$ methods can be explained by considering the particles in subdomain B in Figure 2. At level two of the hierarchy, the two subdomains marked with ‘i’ are well-separated from subdomain B and these two computational elements can be evaluated at the particles in B. At level three, the 15 subdomains marked with ‘i’ are well-separated from B and their computational elements can be evaluated at the particles in B. The above process continues until reaching the leaf-level, at which level the far-field

Figure 2: An $O(N \log N)$ hierarchical method.

potential at the particles in B have been computed. The domains labeled 'i' are known as the *interactive-field*, which is defined for each level of the hierarchy. Since there are at most 27 new subdomains at each level of the hierarchy that are well-separated from subdomain B , and the hierarchy depth is $\log N$, the total number of computational elements evaluated at each particle is $O(\log N)$. Thus, the evaluation of all particles' far-field potential requires $O(N \log N)$ operations. The *near-field* evaluation requires $O(N)$ operations, since there are $O(1)$ particles per leaf-level computational element and there are eight computational elements in the near-field (except close to the boundary) of each particle. Thus, the complexity is dominated by the interactive-field computations, and the total complexity is $O(N \log N)$. Barnes-Hut's method is such an $O(N \log N)$ hierarchical method.

The $O(N)$ methods evaluate the interactive-field hierarchically by introducing a *local-field potential representation* – a second kind of computational element. This element approximates the potential field in a “local” domain due to particles in the far domain. The domain of the new computational element encloses a number of evaluation points, which may be lower level computational elements of the same kind, or particles for the leaf-level computational element. The field representation of the new computational element is with respect to its geometric center. The new type of computational element allows contributions from different far domains to be combined for the far-field evaluation with respect to all subdomains making up the new computational element. The interactive-field evaluation consists in conversion of the far-field potential representation (of the far domains) to the local-field potential representation (of the subdomain under consideration). Since the evaluation is made hierarchically instead of for each leaf-level box, the computational complexity is $O(N)$ for a hierarchy of depth $\log N$.

Hierarchical methods can be abstracted in terms of three functions G, Φ, Ψ , three translation operators T_1, T_2 and T_3 , and a set of recursive equations. G is the potential function in an explicit Newtonian formulation, Φ_i^l is the contribution of subdomain i at level l to the potential field in domains in its far-field. Ψ_i^l represents the contribution to the potential field in subdomain i at level l due to particles in the far-field region of subdomain i , i.e., the local-field potential in subdomain i at level l . The computational structure is described as follows by Katzenelson [19]:

Algorithm: (A generic hierarchical method)

1. Compute Φ_i^h for all boxes i at the leaf level h .
2. Upward-pass: for $l = h - 1, h - 2, \dots, 2$, compute

$$\Phi_n^l = \sum_{i \in \{children(n)\}} T_1(\Phi_i^{l+1}).$$

3. Downward-pass: for $l = 2, 3, \dots, h$, compute

$$\Psi_i^l = T_3(\Psi_{parent(i)}^{l-1}) + \sum_{j \in \{interactive-field(i)\}} T_2(\Phi_j^l).$$

4. Far-field: evaluate local potential at particles inside every leaf-level subdomain

$$\phi_{k \text{ far-field}} = \Psi_{box(k)}^h(k).$$

5. Near-field: evaluate the potential field due to the particles in the near-field of leaf-level subdomains, using a direct evaluation of the Newtonian interactions with nearby particles,

$$\phi_{k \text{ near-field}} = \sum_{j \in \{near-field(k)\}} G_j(k).$$

The physical meanings of T_1, T_2 and T_3 are: shifting a far-field potential, converting a far-field potential to a local-field potential, and shifting a local-field potential.

The constants of the $O(N)$ methods are significantly impacted by the definition of the *near-field*. In two dimensions, the *near-field* usually is defined as those subdomains that share a boundary point with the considered subdomain, while in three dimensions the *near-field* is defined to contain *nearest neighbor* subdomains which share a boundary point with the considered subdomain and *second nearest neighbor* subdomains which share a boundary point with the nearest neighbor subdomains [15]. The *far-field* of a subdomain is the entire domain excluding the subdomain and its near-field subdomains. The *interactive-field* of a subdomain at level l is the part of the far-field that is contained in its parent's near-field. In three dimensions, the number of subdomains in the near-field is 124 (a $5 \times 5 \times 5$ subdomain excluding the considered domain) and the number of subdomains in the interactive-field is 875 (a $10 \times 10 \times 10$ subdomain excluding the near-field). In two dimensions, the near-field contains 8 subdomains and the interactive-field contains 27 subdomains, respectively.

For N uniformly distributed particles and a hierarchy of depth h having $M = 8^h$ leaf-level boxes, the total number of operations required for the above generic hierarchical method is

$$T_{total}(N, M, p) = O(Np) + O(f_1(p)M) + O((N_{int} \cdot f_2(p) + f_3(p))M) + O(Np) + O\left(\frac{N^2}{M}\right),$$

where p is the number of coefficients in the field representation for a computational element, $f_1(p)$, $f_2(p)$, and $f_3(p)$ are the operation counts for the three translation operators, respectively, and N_{int} is the number of interactive-field boxes for interior nodes, i.e., $N_{int} = 875$ for a three-dimensional problem. The five terms correspond to the operation counts for the five steps of the method. The minimum value of T_{total} is $O(N)$ for $M = c \cdot N$, i.e., the number of leaf-level boxes for the optimal depth of the hierarchy is proportional to the

Method	Order	
	Fourth	Ninth
Anderson	4,500	26,500
Barnes–Hut	6,000	
Greengard–Rokhlin	9,000	36,000

Table 2: Break-even points between the hierarchical and direct methods in three dimensions.

number of particles. Since the terms linear in M represent the operation counts in traversing the hierarchy, and the term $O(\frac{N^2}{M})$ represents the operation count in the direct evaluation in the near-field, the optimal hierarchy depth balances the time of the hierarchy traversal and the direct evaluation.

In three dimensions, converting the far-field potentials of interactive-field boxes to local-field potentials dominate the time in traversing the hierarchy. The use of supernodes [12, 35] reduces the effective value of N_{int} in three dimensions from 875 to 189, which brings about a dramatic improvement in the overall performance. Using supernodes, the break-even points between the hierarchical and direct methods are as summarized in Table 2.

3 Anderson’s method

Anderson [2] uses Poisson’s formula for representing solutions of Laplace’s equation. One advantage of this formulation is that the component operations of the multipole method are very easy to formulate for approximations based on Poisson’s formula (the translation operators in (2)–(4)). Another advantage is that the computations in two and three dimensions are very similar.

Let $g(x, y, z)$ denote potential values on a sphere of radius a and denote by Ψ the harmonic function external to the sphere with these boundary values. Given a sphere of radius a and a point $\vec{x}$ with spherical coordinates (r, θ, ϕ) outside the sphere, let $\vec{x}_p = (\cos(\theta)\sin(\phi), \sin(\theta)\sin(\phi), \cos(\phi))$ be the point on the unit sphere along the vector from the origin to the point $\vec{x}$. The potential value at $\vec{x}$ is (equation (14) of [2])

$$\Psi(\vec{x}) = \frac{1}{4\pi} \int_{S^2} \left[\sum_{n=0}^{\infty} (2n+1) \left(\frac{a}{r}\right)^{n+1} P_n(\vec{s}_i \cdot \vec{x}_p) \right] g(a\vec{s}) ds, \quad (2)$$

where the integration is carried out over S^2 , the surface of the unit sphere, and P_n is the n th Legendre function.

Given a numerical formula for integrating functions on the surface of the sphere with K integration points $\vec{s}_i$ and weights w_i , the following formula (equation (15) of [2]) is used to approximate the potential at $\vec{x}$:

$$\Psi(\vec{x}) \approx \sum_{i=1}^K \left[\sum_{n=0}^M (2n+1) \left(\frac{a}{r}\right)^{n+1} P_n(\vec{s}_i \cdot \vec{x}_p) \right] g(a\vec{s}_i) w_i. \quad (3)$$

This approximation is called an *outer-sphere approximation*. Note that in this approximation two approximations are made compared to Equation (2): the series is truncated, and the integral is evaluated with a finite number of terms.

Figure 3: Translations as evaluations of the approximations.

The approximation used to represent potentials inside a given region is (equation (16) of [2])

$$\Psi(\vec{x}) \approx \sum_{i=1}^K \left[\sum_{n=0}^M (2n+1) \left(\frac{r}{a}\right)^{n+1} P_n(\vec{s}_i \cdot \vec{x}_p) \right] g(a\vec{s}_i) w_i \quad (4)$$

and is called an *inner-sphere approximation*.

The outer-sphere and the inner-sphere approximations define the computational elements in Anderson's method. Outer-sphere approximations are constructed for clusters of particles in the leaf-level boxes. During the upward pass, outer-sphere approximations of child boxes are combined into a single outer-sphere approximation of their parent box (T_1) by simply evaluating the potential induced by the component outer-sphere approximations at the integration points of the parent outer-sphere, as shown in Figure 3. The situation is similar for the other two translations used in the method, which are shifting a parent box's inner-sphere approximation to add to its children's inner-sphere approximations (T_3) and converting the outer-sphere approximations of a box's interactive-field boxes to add to the box's inner-sphere approximation (T_2).

In addition to the depth of the hierarchy, suitable values for the three parameters, K , M , and a associated with each computational element must be determined. All three parameters affect the accuracy of the result.

A suitable value for K depends upon the choice of integration method, and required accuracy. The traditional integration methods that use a product grid and trapezoidal integration in the θ direction and Gaussian quadrature in the ϕ direction are inefficient, since the integration points are crowded near the poles. Integration formulas using non-product grids, as described in [25], avoid this inefficiency and are used by Anderson. The 5th, 7th, 9th, 11th, and 14th order integration formulas require 12, 24, 32, 50, and 72 integration points, respectively.

A suitable choice of M depends upon the order of the integration method. Aliasing occurs if there are more terms in the series expansion than can be resolved by the sampling points used in the integration. Anderson showed that if an integration formula of degree D is used, then an appropriate choice for M is $M \leq \frac{D}{2}$. The reason is that for a function on the sphere comprised of spherical harmonics of degree m , the spherical expansion coefficients of this function can be determined exactly by an integration formula of degree $2m$.

Finally, a , the radius of the sphere representing the domain of the computational element must be determined. Assume that the smallest sphere in which a collection of particles are contained has radius α . Then, we can choose

Figure 4: Overview of the CM-5/5E system.

any radius $a \geq \alpha$ for the domain of the computational element representing the outer-sphere approximation. The error in the outer-sphere potential approximation induced by a collection of particles inside a sphere of radius α at distance $r > \alpha$ from the center of the sphere is $O(\frac{\alpha}{r})^{M+2}$. Thus, the larger the value of a , the smaller the error in the field evaluation at the integration points of the sphere. On the other hand, for very large values of a the fact that floating-point accuracy is of limited precision could result in loss of numerical accuracy in accumulating terms. Anderson suggested that the proper choice of a for the outer-sphere approximations is $a = 2\alpha$, and that for the inner-sphere approximations $a = \alpha/2$ shall be used.

4 The Connection Machine system CM-5/5E architecture

A CM-5/5E system contains up to 16,384 parallel processing nodes (the largest configuration available today has 1,024 nodes), each with its own memory (see Figure 4). A collection of nodes, known as a “partition”, is supervised by a control processor called partition-manager, although the nodes may operate independently in MIMD mode. Each node is connected to two low latency, high bandwidth interconnection networks, the Data Network and the Control Network. The Data Network is generally used for point-to-point inter-node communication, and the Control Network for operations such as synchronization, broadcasting and parallel prefix operations. A third network, the Diagnostics Network, is used to ensure the proper operation of the system.

Figure 5 illustrates the architecture of a single processing node (PN) of a CM-5/5E. Each node is a SPARC microprocessor, with four virtual Vector Units (VUs) emulated by two physical VUs for enhanced floating-point performance. The VUs are memory mapped into the SPARC address space. The SPARC serves as a controller and coordinator for the four VUs. Each VU consists of an MBus interface and an instruction decoder as well as an ALU. Each virtual VU has its own Register File. The assembly instruction set contains vector instructions for a four-stage pipeline. The clock frequency of the node processor and the two physical VUs is 40 MHz for the CM-5E. The four virtual VUs run at half this clock frequency, and so do the four memory banks, one for each of the four virtual VUs. The ALU can perform a floating-point or integer multiply-add or multiply-subtract operation on 64-bit operands per clock cycle. Thus, the peak performance of a VU is 40 Mflops/s and the performance of a CM-5E node is 160 Mflops/s. The ALUs also support 32-bit operations,

Figure 5: Overview of the CM-5/5E Node with Vector Units.

but the computational rate is the same as in 64-bit precision.

Each VU can address up to 128 Mbytes of memory, giving a maximum of 512 Mbytes/PN. The path between each VU and its memory is 64-bit wide, and the pipelined VUs can access memory on each clock cycle. The maximum bandwidth to memory is 640 Mbytes/s/PN. The memory per VU is 8 Mbytes and 32 Mbytes, respectively for 4 and 16 Mbit memory chips. The VU memory is dynamic RAM, DRAM, with a page size of 8 Kbytes for 4 Mbit memory chips, and 16 Kbytes for 16 Mbit memory chips. If successive memory accesses are to locations which are not on the same DRAM page, then a DRAM page fault occurs, which on the CM-5/5E implies that the pipeline is stalled for 5 VU cycles. Hence, it is desirable to organize the scheduling of operations such that DRAM page faults are minimized. In addition, not all of the VU memory is mapped into the SPARC address space at all times. Hence, in order to minimize TLB thrashing, the order in which DRAM pages are traversed is also important.

5 Data structure and distribution

Memory management on distributed memory machines include not only exploiting spatial and temporal locality in the local memory hierarchies, but also maximizing concurrency and minimizing communication among nodes. These issues are further complicated by the fact that very often the data distribution, or *layout*, is not known until run-time. This is the case, in particular, when codes can be executed on different size systems without recompilation.

There are two main data structures in a hierarchical method: one for storing the potential field in the hierarchy and the other for storing particle information.

Figure 6: Embedding of a hierarchy of grids in two 4-D arrays.

Far-field potentials are stored for all levels of the hierarchy, since they are computed in the upward pass but used in the downward pass. We embed the hierarchy of far-field potentials in one five-dimensional (5-D) array as follows (see Figure 6): the leaf-level is embedded in one layer of the 4-D array, i.e., **FAR_POT**(1, :, :, :), and level $(h - i)$ is embedded in **FAR_POT**(2, :, $2^{i-1} : L : 2^i$, $2^{i-1} : M : 2^i$, $2^{i-1} : N : 2^i$). Three of the axes represent the organization of the boxes in the three spatial dimensions, while the fourth axis is used to represent data local to an element. The embedding preserves locality between a box and its descendants in the hierarchy. If at some level, there is at least one box per VU, then for each box, all its descendants will be on the same VU as the box itself.

The 5-D array representation of the potential field is quite efficient with respect to memory utilization. Moreover, it is easy to use for any depth of the hierarchy; only the extent of the three spatial axes depend on the depth of the hierarchy. Representing each level of the hierarchy as a separate array can clearly be made memory-efficient, but the number of arrays depends on the depth of the hierarchy. Using arrays with one of the axis representing the levels of the hierarchy would require ragged arrays for space efficiency. But, ragged arrays are neither supported in CMF nor in HPF.

The input to the program consists of a bounding box and relevant particle data given in the form of a collection of 1-D arrays, one for each attribute. For particle-box interactions it is however both convenient and efficient to represent particle attributes as 4-D arrays, with three of the axes representing the domains of the leaf-level potential array boxes, and the fourth representing the particles in those boxes. Our implementation uses 1-D arrays for input and output of particle attributes, but 4-D arrays for particle-box and particle-particle interactions.

Given an array declaration with compiler directives that only specify whether or not an axis is distributed (parallel) or local to a VU (serial), the Connection Machine Run-Time System attempts to balance subgrid extents and minimize the surface-to-volume ratio. Since communication is minimized for the nonadaptive hierarchical methods when the surface-to-volume ratio of the subgrids is minimized, the default layout is ideal.

Let the extents of the three spatial axes of the 5-D potential arrays be $\mathbf{L}$, $\mathbf{M}$, and $\mathbf{N}$, respectively. The extents are equal to the number of leaf-level boxes along the three spatial dimensions, and hence are powers of 2 for a nonadaptive method. The global address space, denoted by $b_{p+n-1}b_{p+n-2}\dots b_nb_{n-1}b_{n-2}\dots b_0$, is mapped onto the underlying physical machine. With block allocation, the address field is broken into two parts – the high

order bits form the VU address and the low order bits form the local memory address. For a multidimensional array with block allocation for each axes, the VU address field and the local memory address field are both broken into segments, one for each axis. Moreover, since on the Connection Machine systems the number of VUs along any axis is constrained to be a power-of-two and the number of leaf-level boxes along any axis is a power-of-two as well, it suffices to consider address bits in studying the layout of boxes. The address fields for each of the two 4-D subarrays of the 5-D potential array are shown in Figure 7.

	processor address			local memory address		
	$b_{p+n-1}b_{p+n-2}\dots b_n$			$b_{n-1}b_{n-2}\dots b_0$		
K				b..b		
L	b..b				b..b	
M		b..b				b..b
N			b..b			b..b

Figure 7: The allocation of the local potential arrays **LOCAL_POT** to processing nodes.

Note that if we use a collection of 4-D arrays for the particle attributes, and the allocation of the three spatial axes of these arrays are the same as that of the arrays for the local potential of leaf-level boxes, then the particle attributes will be allocated to the same VU as the leaf-level box of the hierarchy to which the particles belong. In the next section we discuss how to accomplish this form of *alignment* of particle attributes and leaf-level boxes.

6 Optimizing communication

In this section we discuss issues in optimizing the three kinds of communication required in hierarchical N -body methods. First, before the traversal of the hierarchy, *particle-box* interactions are required to exchange information between particles and the potentials of leaf-level boxes. Second, during the upward pass and the downward pass through the hierarchy, translations between the far-field or local-field potentials for parent and child boxes require *parent-child (box-box) interactions*. Third, at each level of the hierarchy during the downward pass, *neighbor (box-box) interactions* are needed to convert the far-field potential of interactive-field boxes to the local-field potential of each box.

6.1 Particle-box interactions

Particle-box interactions occur in forming the far-field potential for leaf-level boxes before traversing the hierarchy, and in evaluating the local-field potential of leaf-level boxes at the particles inside each box after traversing the hierarchy. The direct evaluation of the near-field can also very conveniently be based on the particle-box structures, since the particle-particle interactions are defined by the collection of boxes defining the near-field.

For the leaf-level particle-box interactions before traversing the hierarchy, the contributions of all particles in a box to each of the integration points on the sphere in Anderson's method (or to the multipole expansion for the

Figure 8: Sorting particles for maximum locality in reshaping particle arrays.

box), must be accumulated. Different boxes have different number of particles, and the number of terms added varies with the leaf-level box. Once the particles are sorted such that particles belonging to the same box are ordered together, a segmented $+$ -scan is a convenient way of adding the contributions of all the particles within each of the boxes in parallel. A *send* communication is needed to move data between 1-D sorted particle arrays used for the $+$ -scan and the 4-D potential arrays for particle-box interactions. A *send* communication is also required in evaluating the local-field potential at the particles inside the leaf-level boxes.

Both scan and send communication may be quite time consuming on the CM-5/5E. However, if the particles are sorted in such a way that they are allocated to the same VU as the leaf-level boxes to which they belong, then both the scan and the send require no communication. The segmented scan becomes a set of scans local to each VU and can be implemented very efficiently. The sends become local memory references (copy). In order to approach this situation, we sort the particles based on keys constructed from the particle locations, the leaf-level box coordinates, and their allocation to VUs. For a uniform (almost-uniform) distribution, after the sorting all (most) particles will be allocated to the same VU as the leaf-level box to which they belong. The following *coordinate sort* (see Figure 8) accomplishes this task.

Algorithm: (Coordinate sort)

1. Find the layout of the 4-D potential arrays using intrinsic mapping functions, e.g., the number of bits for the VU address and the local memory address for each axis;
2. For each particle, generate the coordinates of the box to which it belongs, denoted by $xx..x$, $yy..y$, and $zz...z$;
3. Split the box coordinates into VU address and local memory address according to the layout of the potential arrays, denoted as $x..x|x..x$, $y..y|y..y$, $z..z|z..z$;
4. Form keys for sorting by concatenating the VU addresses with local memory addresses, denoted as $z..zy..yx..x|z..zy..yx..x$;
5. Sort.

Particles belonging to the same box are adjacent to each other after sorting. Furthermore, for a uniform particle distribution, if there is at least one leaf-level box per VU, then each particle in the coordinate sorted 1-D array

Figure 9: Exploiting symmetry in the direct evaluation in the near-field.

will be allocated to the same VU as the leaf-level box to which it belongs in the 4-D array of particle attributes. Therefore, no communication is needed in copying particle attributes from the sorted 1-D array to the 4-D array with the same layout as the arrays of local-field potentials. For a near-uniform particle distribution, it is expected that the coordinate sort will leave most particles in the same VU memory as the leaf boxes to which they belong. Using 4-D particle arrays, the scan and send in the particle-box interactions require no communication.

The direct evaluation in the near-field based on 4-D particle arrays can be viewed as neighbor box-box interactions: each box interacts with its 124 neighbor boxes in the near-field. Each neighbor box-box interaction involves all-to-all interactions between particles in one box and particles in the other. Exploiting the symmetry of interaction (Newton’s third law) result in 62 instead of 124 box-box interactions. One way of exploiting symmetry is shown in a 2-D example in Figure 9. As box 0 traverses boxes 1-4, the interactions between box 0 and each of the four boxes will be computed. The interactions from the four boxes to box 0 are accumulated and traverse along with box 0. Using data parallel programming, while box 0 traverses boxes 1-4, boxes 5-8 will traverse box 0 and the interactions between them and box 0 will be computed. The interactions from these four boxes to box 0 will be accumulated and stored in box 0. Finally, the two contributions to box 0 will be combined with interactions among particles in box 0. A similar idea for exploiting symmetry applied directly to particles instead of boxes was used in [3].

6.2 Box-box interactions

Parent-child box-box interactions are required both in combining far-field potentials in the upward pass through the hierarchy, and in local-field evaluations in the downward pass.

Neighbor box-box interactions are required for the far-field evaluation of interactive-field boxes in the downward pass of the hierarchy, and in the direct evaluation of the near-field using the 4-D array representation of the particles. In our implementation of interactive-field computations (which does not exploit the parallelism among the boxes in the interactive-field) each target box needs to fetch the potential vectors of its 875 neighbor boxes (if supernodes [35] are not used).

Figure 10: Optimizing communication in neighbor interactions.

6.2.1 Interactive-field box-box communication

The simplest way to fetch potential vectors of neighbor boxes is to use individual CSHIFTs, one for each neighbor, as shown in Figure 10(a). In the Connection Machine Run-Time System, composite CSHIFTs are implemented as a sequence of independent shifts, one for each axis, resulting in excessive data motion.

A better way to structure the CSHIFTs is to impose a linear ordering upon the interactive-field boxes, as shown in Figure 10(b). The potential vectors of neighbor boxes are shifted through each target box, using a CSHIFT with unit offset at every step. The three axes use different bits in their VU addresses. The rightmost axis uses the lowest order bits and the leftmost axis uses the highest order bits in the default axes ordering. Nodes that differ in their lowest order bits are adjacent in many networks. In such networks, the best linear ordering should use CSHIFTs along the rightmost axis most often. Due to the construction of the fat-tree network on the CM-5/5E and the array layout, the outlined shift order is advantageous.

Unfortunately, the scheme just outlined results in excessive data motion, though less so than the direct use of CSHIFTs. Assume that in two dimensions every VU has a $S1 \times S2$ subgrid of boxes¹, and that the CSHIFTs are made most often along the y -axis. Every CSHIFT with unit offset involves a physical shift of boundary elements off-VU and a local copying of the remaining elements. After shifting six steps along the y -axis in Figure 10(c), the CSHIFT makes a turn and moves along the x -axis in the next step, followed by a sequence of steps along the y -axis in the opposite direction. All the elements in a VU, except the ones in the last row before the turn, are moved back through the same VUs during the steps after making the turn. Thus, this seemingly efficient way of expressing neighbor communication in CMF involves excessive communication in addition to the local data movement. Nevertheless, on a CM-5E it was 7.4 times faster than direct use of CSHIFTs for a subgrid with axes extents 16 and $K = 12$.

¹ We ignore the local axis in this section since communication only happens on parallel axes.

Figure 11: A plane of the near-field and interactive-field in three dimensions.

In order to eliminate excess data movement, non-local interactive-field boxes for all boxes in the local subgrid can be identified and the communication structured to fetch only those boxes. Figure 11 shows a plane through a target box and its near-field and interactive-field boxes. For a child box on the boundary of the subgrid in a VU, the interactive-field box furthest away from it is at distance four along the axis normal to the boundary of the subgrid. Hence, the “ghost” region is four boxes deep on each face of the subgrid. Using the array aliasing feature of CMF, the ghost boxes can be easily addressed by creating an array alias that separates the VU address from the local memory address. With the subgrids identified explicitly, fetching boxes in ghost regions in three dimensions require that six surface regions, 12 edge regions and eight corner regions be fetched.

Fetching the potential vectors of the boxes in the ghost region directly through array sections and CSHIFTs in principle results in no excess data motion. However, due to the implementation of CSHIFTs on the CM-5/5E excess data motion is incurred, as mentioned above. Hence, this seemingly optimal way of fetching ghost boxes is not the most efficient technique on the CM-5/5E. Creating a linear ordering through all the VUs containing ghost boxes and using CSHIFTs to move whole subgrids followed by array sectioning after subgrids are moved to the destination VU reduces the communication time by a factor of about 1.5. Moving whole subgrids is necessary in order to keep the continuity of the linear ordering of the subgrids. Although some redundant data motion takes place, it is considerably reduced compared to using a linear ordering on the “unfactored” array. Table 3 summarizes the data motion requirements for the four methods for a subgrid of shape $S1 \times S2 \times S3$ with $S1 = S2 = S3 = 8$.

The prefetched ghost boxes and the local subgrid can be stored in a new array with a $(S1+8) \times (S2+8) \times (S3+8)$ subgrid. Alternatively, the ghost boxes can be stored in a separate array. The benefits of using a separate array for the boxes fetched from other VUs is that copying of the local subgrid is avoided, and storage is saved by not storing the subgrid local to a VU twice. The drawback with this approach is more complex control in the interactive-field evaluation, and lower arithmetic efficiency because of shorter vectors and fewer instances for each vector operation compared to using a single subgrid. The excess storage for a single array is relatively modest; for a $8 \times 8 \times 8$ subgrid, the ghost region alone contains 3,584 boxes compared to 512 boxes for the local subgrid.

Method	Number of non-local boxes fetched	Number of local box moves	Number of CSHIFTS	Relative time	
				$K = 12$	$K = 72$
Direct on unfactored arrays				169	178
Linearized unfactored arrays	85,888	596,608	1,333	19.4	18.2
Direct on factored arrays	3,584	7,680	54	1.63	1.48
Linearized factored arrays	4,352	6,400	10	1	1

Table 3: Comparison of data motion needs for interactive-field evaluation on a 32 node CM-5E, with the local subgrid of extents 8 and ghost boxes stored in a $16 \times 16 \times 16$ subgrid when using factored arrays.

The memory requirements can be decreased by reducing the number of ghost boxes prefetched (at a time). For example, instead of prefetching all the ghost boxes required by all interactive-field computations, a column of $(S1 + 8) \times S2 \times S3$ ghost boxes can be fetched and used for interactive-field computations with some fixed offsets along the y -axis and the z -axes, but different offsets along the x -axis. As the offset along the y -axis or the z -axis changes by one, most ghost boxes fetched in the previous step can be reused. However, since in prefetching all the ghost boxes at once, the memory requirement in traversing the hierarchy is about the same as in the direct evaluation in the near-field, we did not explore partial prefetching.

Note that for subgrid extents of less than four along any axis, communication beyond nearest neighbor VUs is required.

6.2.2 Near-field box-box communication

Due to an optimization trading memory requirements for arithmetic efficiency in the particle-particle interactions, the memory requirements in the near-field interactions are high. Hence, we fetch particles in neighbor boxes by using CSHIFTS on unfactored arrays with a linear ordering of boxes in the ghost region. For the near-field the ghost region is two boxes deep in each direction of all axes.

6.2.3 Parent-child box-box interaction

Using the embedding described in Section 5, the far-field potentials of boxes at all levels of the hierarchy are embedded in two layers of a 4-D array. During traversal of the hierarchy, temporary arrays of a size equal to the number of boxes at the current level of the hierarchy are used in the computation.

We abstract two generic functions **Multigrid-embed** and **Multigrid-extract** for embedding/extracting a temporary array of potential vectors corresponding to some level of the hierarchy into/from the 4-D array. The reduction operator used in the upward pass is abstracted as a **Multigrid-reduce** operator, while the distribution operator used in the downward pass is abstracted as a **Multigrid-distribute** operator. By first creating an array alias separating the local address from the physical address, then using array sectioning, the embedding/extraction operation is performed as a local copy operation if both source and destination locations are local to a VU. If source and destination addresses are on different VUs, which occurs close to the root of the hierarchy of grids, then a two step procedure is used. First, a temporary array corresponding to the level of

Figure 12: Performance improvement of **Multigrid-embed** using array sectioning and aliasing. The two-step scheme was used for the first two cases and the remaining cases used only local copying.

the hierarchy that has the least number of boxes larger than the number of VUs, i.e., at least one box on each VU, is allocated. Then, for **Multigrid-embed** the source is first embedded in this temporary array, which then is embedded in the 4-D destination array. The second step is a local copy operation, as before, while the first requires communication. But this communication is much more efficient than the communication in embedding the source directly in the destination array because of the smaller overhead in computing send addresses, which is about linear in the array size. For array sectioning, the overhead may dominate the actual communication, which is proportional to the number of elements selected.

On the CM-5E, the performance of **Multigrid-embed** is enhanced by a factor of up to two orders of magnitude using the local copying or the two-step scheme, as shown in Figure 12.

7 Optimizing computation

Our techniques for optimizing the computations in hierarchy traversal resulted in an efficiency of 40% for $K = 12$ and a hierarchy of depth eight, and 69% for $K = 72$ and a hierarchy of depth seven, excluding communication. The peak arithmetic efficiency of about 74% for $K = 12$ and 85% for $K = 72$ is degraded due to copying, masking, a measurable impact of overheads for the higher levels of the hierarchy, and poor vectorization in the direct evaluation in the near-field.

In Anderson’s variant of the fast multipole method, each of the three translation operators used in traversing the hierarchy can be aggregated into matrices and their actions on the potential field further aggregated into *multiple-instance* matrix-matrix multiplications. Since there are no other computations in the hierarchy, the entire hierarchical part takes the form of a collection of matrix-matrix multiplications, which are implemented efficiently on most computers as part of the BLAS (Basic Linear Algebra Subroutines) [7, 8, 20]. The Connection Machine Scientific Software Library [32], CMSL, supports both single-instance and multiple-instance BLAS.

Table 4 summarizes the leaf-level arithmetic efficiencies for $K = 12$ and $K = 72$ on a 256 node CM-5E. Since the matrix multiplication has complexity $O(K^2)$ and the cost of copying and masking is linear in K , the arithmetic efficiency when including copying and masking decreases more significantly for $K = 12$ than for $K = 72$.

Operation	$K = 12, h = 8$		$K = 72, h = 7$	
	$T_1(T_3)$	T_2	$T_1(T_3)$	T_2
Arithmetic	54	74	60	85
Arithmetic incl. copy		60		79
Arithmetic incl. copy and masking		44		74

Table 4: Leaf-level arithmetic efficiencies on a 256 node CM-5E.

7.1 Translations as BLAS operations

The translation operators evaluate the approximations of the source spheres at the integration points of the destination spheres (see Figure 3). A sphere approximation (equation (3) or (4)), is defined by

$$\Phi(\vec{x}_j) \approx \sum_{i=1}^K f(\vec{s}_i, \vec{x}_j) \cdot g(a\vec{s}_i), \quad j = 1, K, \quad (5)$$

where $f(\vec{s}_i, \vec{x}_j)$ represents the inner summation in the original approximation. $f(\vec{s}_i, \vec{x}_j)$ is a function of the unit vector $\vec{s}_i$ from the origin of the source sphere to its i th integration point and the vector $\vec{x}_j$ from the origin of the source sphere to the j th integration point on the destination sphere. Due to the construction of the hierarchy of boxes and the approximation formulas used, $f(\vec{s}_i, \vec{x}_j)$ is unique to each child of a parent, but location and level independent. It is preferably precomputed. The translation of the integration points of a child to each integration point of the parent is an inner-product computation. The translation of the integration points of the child to all of the integration points of the parent constitutes a matrix-vector multiplication, where the matrix is of shape $K \times K$. Thus, equation (5) indeed defines a matrix-vector multiplication.

7.1.1 Translation matrices for T_1 and T_3

Since in three dimensions a parent has eight children, each of the translation operators T_1 and T_3 can be represented by eight matrices, one for each of the different parent-child translations. The same matrices can be used for all levels, and for the translations between any parent and its children irrespective of location. In fact, the eight matrices required to represent T_1 are permutations of each other. One matrix can be obtained through suitable row and column permutations of another matrix.

Let the potential vectors of eight child boxes of a parent box be $f_1, \dots, f_8$, and the translation matrix from one of the child boxes to the parent box be M . In matrix form, the application of T_1 can be written as

$$f = Mf_1 + P_2MP_2'f_2 + \dots + P_8MP_8'f_8, \quad (6)$$

where f is the potential vector of the parent box, and $P_i, P_i', 2 \leq i \leq 8$ are suitable permutation matrices.

Similarly, let the potential vector of a parent box be f , and the translation matrix from the parent box to one of the child boxes be M . The application of T_3 to compute the potential vectors of child boxes can be expressed as

$$Mf, \quad P_2MP_2'f, \quad \dots, \quad P_8MP_8'f. \quad (7)$$

If the permutation property is exploited, then it suffices to store one matrix for T_1 and one for T_3 in each VU, since the matrices for T_1 and T_3 are shared by all the boxes at all levels. Equation (6) can be evaluated by first permuting the potential vectors of seven of the eight child boxes, i.e., generating $P'_2 f_2$, $P'_3 f_3$, etc., then performing a matrix–matrix multiplication of the matrix M and a matrix with the eight potential vectors of the children as columns followed by a permutation of the columns of the product matrix which then are added to form f . For T_3 , seven different permutations of f are generated first, then a matrix–matrix multiplication performed as for T_1 followed by a permutation of the columns of the product matrix. No reduction is required. This approach saves on the computation and storage of translation matrices and may achieve better arithmetic efficiency through the aggregated matrix–matrix multiplication. However, on the CM-5E, the time for the permutations exceeds the gain in arithmetic efficiency. In our code we store all eight matrices for each translation operator.

Even though permutations are not used in applying the translation operators to the potential field, they could be used in the precomputation phase. Since the permutations depend on K , the number of integration points in a non-trivial fashion, using permutations in the precomputation stage would require storage of the permutations for all different K s. In order to conserve memory we explicitly compute all matrices at run-time (when K is known). We discuss redundant computation–communication trade-offs in Section 8.

7.1.2 Translation matrices for T_2

The interactive-field computations dominate the hierarchical part of the code. In three dimensions the interactive-field contains no boxes inside a $5 \times 5 \times 5$ subgrid centered at the target box. Depending upon which child box of a parent is the target, the interactive-field extends two or three boxes beyond the near-field at the level of the child box in the positive and negative direction along each axis. Together, the target box and its near-field and interactive-field boxes form a $10 \times 10 \times 10$ subgrid. The subgrid is centered at the center of the parent, and is the same for all children of the parent, although the near-field and interactive-fields of siblings differ.

Each box, except boxes sufficiently close to the boundaries, has 875 boxes in its interactive-field. Though each of the eight children of a parent requires 875 matrices, the siblings share many matrices. The interactive-field boxes of the eight siblings have offsets in the range $[-5 + i, 4 + i] \times [-5 + j, 4 + j] \times [-5 + k, 4 + k] \setminus [-2, 2] \times [-2, 2] \times [-2, 2]$, $i, j, k \in \{0, 1\}$, respectively. For illustration, see Figure 11. Each offset corresponds to a different translation matrix. The union of the interactive-fields of the eight siblings has $11 \times 11 \times 11 - 5 \times 5 \times 5 = 1206$ boxes with 1206 offsets in the range $[-5, 5] \times [-5, 5] \times [-5, 5] \setminus [-2, 2] \times [-2, 2] \times [-2, 2]$. For ease of indexing, we generate the translation matrices also for the 125 subdomains excluded from the interactive-field, or a total of $11 \times 11 \times 11 = 1331$ matrices.

7.2 Aggregation of translations

Aggregation of computations lowers the overheads in computations. In addition, the aggregation of computations may allow for additional optimizations by providing additional degrees of freedom in operations to be scheduled at a given time.

In Anderson’s method aggregation of computations in parent–child interactions results in multiple-instance

matrix-matrix multiplications. Inner-products are aggregated into matrix-vector multiplication for a single parent-child interaction. Since the same matrix is used for all parent-same-child interactions, the matrix-vector multiplications can be further aggregated into matrix-matrix multiplication. With the data layout used for the 4-D potential arrays, aggregation can only be performed along one of the three space dimensions without a data reallocation in local memory. Thus, for a $K \times K$ matrix, aggregation without data reallocation results in a $K \times K$ by $K \times Si$ matrix multiplication, where Si is the axis of aggregation. But, Sm such matrix multiplications can be treated as one multiple-instance matrix-matrix multiplication, where Sm is the length of the axis chosen for the multiple-instance call to a CMSSL multiple-instance matrix multiplication. This aggregation improved the performance of the T_1 and T_3 translations from 58 Mflops/s/PN to 87 Mflops/s/PN for $K = 12$ and a subgrid of extents $32 \times 32 \times 16$. The matrices being multiplied are of shape 12×12 and 12×8 with sixteen such instances handled in a single call. For $K = 72$ and a subgrid of extents $16 \times 16 \times 8$ the performance improved from 95 Mflops/s/PN to 96 Mflops/s/PN. Matrix shapes are 72×72 and 72×4 and the number of instances is 8.

The conversion of the far-field to local-field potential of the boxes in the interactive-field is made using an array with subgrid of shape $(S1 + 8) \times (S2 + 8) \times (S3 + 8)$. The ghost region is four boxes deep on all faces of the subgrid. As for parent-child interactions the interactive-field conversion for a single box pair can be aggregated into a matrix-vector multiplication. All box pairs with the same relative location use the same matrix, and computations for all local boxes and the interactive-field boxes with the same relative location with respect to each one of them can be aggregated into a single matrix-matrix multiplication. For the interactive-field computations we rearrange the arrays in local memory such that a single-instance matrix multiplication is performed on matrices of shape $K \times K$ and $K \times (\frac{S1}{2} \cdot \frac{S2}{2} \cdot \frac{S3}{2})$. For $S1 = S2 = 32, S3 = 16$ and $K = 12$, the execution rate of the 12×12 by 12×2048 matrix multiplication is 119 Mflops/s/PN. If there are no DRAM page faults, the copying requires $2K$ cycles for a potential vector for which the matrix multiplication ideally takes K^2 cycles. Thus, the relative time for copying is $2/K$. For $K = 12$ this amounts to about 17%. With the cost of copying included, the measured performance of the translation is 85 Mflops/s/PN. For $S1 = 16, S2 = 16, S3 = 8$ and $K = 72$, the execution rates of the 12×12 by 12×256 matrix multiplication is 136 Mflops/s/PN. Including the cost of copying, the measured performance is 124 Mflops/s/PN.

The copying cost can be reduced by copying a whole column block of $(S1 + 8) \times S2/2 \times S3/2$ boxes into two linear memory blocks; one for even slices of the column, and the other for odd slices. Each local column copy can be used on average 8.75 times. The cost of copying is therefore reduced to $\frac{4 \cdot 100}{875} \cdot \frac{(S1+8)}{(S1 \cdot K)}$ of that of matrix multiplication, assuming no page faults. Including the cost of copying, the performance of translations in neighbor interactions reaches 96 and 127 Mflops/s/PN for $K = 12$ and $K = 72$, respectively.

Copying of sections of subgrids to allow for a $K \times K$ by a $K \times (\frac{S1}{2} \cdot \frac{S2}{2} \cdot \frac{S3}{2})$ matrix multiplication can also be used in parent-child interactions, but the copying cost is relatively higher. In estimating the copy cost for the interactive-field computations, we ignored the copy back cost after the accumulation. With this cost included, the total copy cost for the T_2 operator is $2K + 2K/875$ cycles per matrix-vector multiplication. For the parent-child interactions the total copy cost is $2K + 2K/8$ cycles, ideally, per $K \times K$ matrix-vector multiplication. The copy cost is about 10% higher for the parent-child interactions than for interactive-field computations. Using copying in parent-child interactions, the performances for the T_1 and T_3 matrix operations drops from 87 to 82 Mflops/s/PN for $K = 12$ but increases from 96 to 123 Mflops/s/PN for $K = 72$, due to the relative cost of

copying to matrix multiplication. We did not incorporate the copying of arrays for increased performance in parent-child interactions mainly because the time for parent-child interactions accounts for a very small portion of the hierarchy traversal time.

8 Redundant computation vs. replication

All translation matrices are precomputed. Since the translation matrices are shared by all boxes at all levels, only one copy of each matrix is needed on each VU. Two extreme ways of computing these translation matrices are:

1. compute all the translation matrices on every VU,
2. compute each translation matrix only once with different VUs computing different matrices followed by a spread to all other VUs as a matrix is needed.

In the first method the computations are embarrassingly parallel and no communication is needed. However, redundant computations are performed. In the second method there is no redundant computation, but replication is required. If there are fewer matrices to be computed than there are VUs, then the VUs can be partitioned into groups with as many VUs in a group as there are matrices to be computed. Each group computes the entire collection of matrices, followed by spreads within that group when a matrix is needed. The replication may also be performed as an all-to-all broadcast [17]. The load-balance with this amount of redundant computation is the same as with no redundancy, but the communication cost may be reduced.

On the CM-5E, for K varying from 12 to 72, replicating a $K \times K$ translation matrix to all nodes is about three to twelve times faster than computing it. Thus, computing the matrices in parallel followed by replication is always a winning choice. For T_1 and T_3 replication among eight VUs instead of all VUs is an option. Figure 13 shows the performance of the three methods: no replication, replication among groups of eight VUs, and replication to all VUs. The cost of computing the matrices in parallel followed by replication without grouping is 66% to 24% of that of computing all matrices on each VU, as K varies from 12 to 72. With grouping, the computation cost is the same as without grouping, but the cost of replication is reduced by a factor of 1.75 to 1.26 as K varies from 12 to 72. The reason for the decrease of the difference as K increases is that for larger K , the replication time is dominated by bandwidth, while for small K latency and overhead dominate.

For T_2 , computing one copy of each of the 1331 translation matrices and replicating it is up to an order of magnitude faster than computing all on every VU, as shown in Figure 14(a) for a 256 node CM-5E. The time for computing 1331 matrices in parallel decreases on larger CM-5Es, as shown in Figure 14(b), while the replication time, which dominates the total time, increases by about 10-20% for large K as the number of nodes doubles. As a result, the total time for the method increases by at most 62% as the number of nodes changes from 32 to 512.

Storing all 1331 translation matrices in double-precision on each VU requires $1331 \cdot 8 \cdot K^2$ bytes of memory, i.e., 1.53 Mbytes for $K = 12$ and 53.9 Mbytes for $K = 12$. Therefore, replication of a matrix is delayed until it is needed. The replication is made through one-to-all broadcast rather than all-to-all broadcast. The total

Figure 13: Computation vs. replication in precomputing translation matrices for T_1 (T_3) on a 256 node CM-5E.

Figure 14: Computation vs. replication in precomputing translation matrices for T_2 on the CM-5E.

number of replications is $1331 \cdot (h - 1)$, where h is the depth of the hierarchy, since the T_2 translations are used first at level two.

9 Direct evaluation in the near-field

The $O(N)$ N -body methods minimize their execution time when the hierarchy traversing time is about the same as the time for the direct evaluation in the near-field. The efficiency in the direct evaluation is crucial to the performance of fast hierarchical methods. The near-field is evaluated as a sequence of particle-particle interactions ordered with respect to the boxes to which they belong. The 124 neighbor boxes of a target box, can be ordered linearly and brought to the target box through 124 single-step CSHIFTs. Another way is to fetch non-local near-field boxes from other VUs using 4-D arrays factored into local subgrids through array aliasing, much in the same way as in fetching non-local interactive-field boxes. The first method requires less temporary storage, and is used for the near-field evaluations. The CSHIFTs accounts for about 10-15% of the time for the direct evaluation.

Once a neighbor box has been brought to the target box, an all-to-all interaction between the particles in the two boxes is required. We investigated three alternatives for the all-to-all interaction. The simplest way is to loop through the particles in both boxes using two nested loops. Unrolling the inner loop can improve the

performance of compiler generated code by 25% on the CM-5E. The vectorization can be further improved by replicating each particle in the neighbor box to every particle in the target box, followed by elementwise particle interactions. But, the broadcast operation for each neighbor particle is relatively time consuming. A third approach, called ‘duplicate-and-slide’, duplicates the target box, i.e., a new 4-D array with a local axis of twice the length of the original array is created. The original 4-D particle array is copied to both the first and the second half of the new array. One sequential loop over the particles in the neighbor box is used. Let b be the length of the serial axis of the original 4-D particle array, i.e., the maximal number of particles per leaf-level box. At the i th iteration, an elementwise interaction between the neighbor box and a b -long segment along the local axis of the new array starting at the i th element is evaluated. It is easy to see that the looping covers all particle interactions between the two boxes. The ‘duplicate-and-slide’ approach duplicates particles once and the computations inside the loop are perfectly vectorized on each VU. On the CM-5E it is the fastest of all three approaches. However, it requires 33% more memory than the other alternatives, or a total of $4N$ memory locations for each particle attribute; N locations for the input 1-D array, $2N$ locations for the 4-D duplicated target, and N locations for the 4-D neighbor.

10 Using supernodes

The computations in traversing the hierarchy are dominated by the far-field to local-field conversion for the interactive-field. The idea of *supernodes* [35] reduces the complexity of these computations by a constant factor of close to five. Zhao made the observation that of the 875 boxes in the interactive field of a destination box for the far-field to local-field conversion, in many cases all eight sibling nodes of a parent belong to the interactive-field of the destination box. Zhao then converted the far-field of the parent node instead of the far-field of all eight sibling nodes. In three dimensions, 98 parent nodes have all their children in the interactive-field, and using the concept of supernodes reduces the number of far-field to local-field conversions per destination node to 189 from 875.

Another way of reducing the number of far-field to local-field conversions is to perform the conversion for several destination boxes that share an interactive-field box once only, then share the conversion among the destination boxes. Specifically, if a box in the interactive-field is common to all sibling nodes of a parent, then the conversion is made with the parent box as a target instead of the sibling boxes. The local-field of the conversion is added to the local-field of the parent. When all such conversions have been made, then the local-field is passed to each of the children through the eight translation operations T_3 , where the local-field conversions from the remaining interactive-field boxes are added. This second approach also reduces the number of field conversions per target box to 189.

The above two ways of reducing the number of field conversions differ in the error introduced when used in different methods. In Greengard-Rokhlin’s or Zhao’s fast multipole method, the far-field and local-field potentials are represented using multipole and local expansions, respectively. If a multipole expansion from a sphere O' is converted into a local expansion inside a sphere O of radius α , then the error of the local expansion when evaluated at any point inside the sphere O is bounded by $C_2(\frac{\alpha}{r})^p$ [35], where r is the distance from the center of O to the surface of the sphere O' , as shown in Figure 15, and p is the number of terms in the expansions. In this case, the first approach is advantageous since it retains $(\frac{r}{\alpha}) > 2$. In Anderson’s method, the far-field and

$$\frac{r_1}{a} = 2.46 \quad \frac{r_2}{a} = 2.08$$

Figure 15: A supernode in the fast multipole method.

$$\frac{r_1}{a} = 2.46 \quad \frac{r_2}{a} = 2.08$$

Figure 16: A supernode in Anderson's hierarchical method.

local-field potentials are represented as functions of the potential values on the spheres surrounding or contained in the boxes. To convert a far-field potential into a local-field potential, the outer-sphere approximation for the far-field potential is *evaluated* at the integration points of the destination sphere. The error in evaluating an outer-sphere approximation (equation (3)) with the series expansion truncated after M terms is $O(\frac{\alpha}{r})^{M+2}$, where α is the radius of the sphere containing the particles and r is the distance of the evaluation point from the center of the sphere, as shown in Figure 16. The fact that the field conversions are actually evaluations of the far-field potentials requires the destination spheres to be in the far-field of the source spheres. Thus, the second type of supernode is favored for Anderson's method, as it keeps the destination sphere in the far-field of the source spheres.

11 Performance results

11.1 Overview

Our CMF implementation of Anderson's method with $K = 12$ integration points on the sphere performs the potential evaluation for 100 million particles uniformly distributed in a cubic region in 180 seconds on a 256 node CM-5E. The evaluation for a system of 100 million uniformly distributed particles is estimated to take around 60 seconds on a 1024 node CM-5E. The overall efficiency is about 27%, and is fairly independent of machine size. With $K = 72$ integration points on the sphere the efficiency improves to 35%. We first give a summary of the timings breakdown in computing the potential field for 100 million uniformly distributed particles on a 256 node CM-5E, then give a more detailed analysis of the effectiveness of the techniques we have presented, followed by results demonstrating the scalability of our implementation.

We implemented both two-dimensional and three-dimensional methods, but all performance results reported are for three-dimensional cubic domains. Though the timings refer to cubic domains, the code for 2-D domains adapts to rectangular regions and the code for 3-D domains adapts to parallelepiped domains, as suggested in [2].

Order of integration (D)	Number of integration points (K)	Number of terms in kernel (M)	Expected error decay rate
5	12	2	4
7	24	3	5
9	32	4	6
11	50	5	7
14	72	7	9

Table 5: Relationship among K , M and the error decay rate in Anderson’s method.

I Native	always 1
II Hennessy & Patterson[16]	ADD,SUB,MULT - 1 DIV,SQRT - 4
III CM-5E/VU normalized	ADD,SUB,MULT - 1 DIV - 5 SQRT - 8

Table 6: Weights for floating-point operations in our three methods for FLOP counts.

The 2-D rectangular (3-D parallelepiped is similar) decomposition in [2] starts with an enlarged rectangular bounding region that has an aspect ratio being a power of two and contains the particles. The refinement of this rectangular domain proceeds by successively dividing the larger side by powers of two until the resulting boxes are square. From this stage on, the refinement is made in both directions.

The accuracy of the method affects the balance between computation and communication, as well as the efficiency of individual operations. The accuracy is determined by the number of integration points K on the spheres, and the number of terms M in the expansions. K and M are related as described in Section 3. The relationship among K , M and the error decay rate in Table 5 was given in [2].

For gravitational and Coulombic fields division and square roots represent a significant fraction of the arithmetic time. We report floating-point rates for three different weights of these operations as specified in Table 6.

In order to determine the minimum execution time the optimal hierarchy depth must be determined. Though the arithmetic efficiency depend upon the depth of the hierarchy and so does the relative impact of communication, determining the optimal depth based on minimizing the number of arithmetic operations is a very good approximation of the true optimal height, as is verified in Section 11.3.

The FLOP counts for various stages of Anderson’s method are listed in Table 7, where $\{a,b,c\}$ denotes the FLOP counts using Method I, II, and III, from Table 6, h is the depth of the hierarchy, and m the number of particles per leaf-level box. The optimal hierarchy depth for a given number of particles N is either $\lfloor \log \frac{N}{m_{opt}} \rfloor$ or $\lceil \log \frac{N}{m_{opt}} \rceil$, whichever gives the lower FLOP count. Note that the number of FLOPs per particle at the optimal hierarchy depth is linear in K with a constant of proportionality in the range 1323 – 1678 depending upon which FLOP count method is used. The optimal FLOPs per particle is shown in Figure 17.

	FLOPs
Outer-sphere (upward pass)	
leaf-level	$N * K * \{13, 19, 24\}$
rest levels	$2K^2 * \sum_{d=0}^{h-3} 8^{h-d}$
Inner-sphere (downward pass)	
from parent	$2K^2 * \sum_{d=0}^{h-3} 8^{h-d}$
from neighbor	$2K^2 * 189 * \sum_{d=0}^{h-2} 8^{h-d}$
Far-field	$N * K * (\{9, 12, 13\} * M + 10)$
Direct evaluation in the near-field	$\{943, 1321, 1536\} \frac{N^2}{8^h}$
I Total FLOPs	$N \cdot K \cdot (9M + 23) + K^2 \cdot (436.5 \cdot 8^d - 3748.5) + 943 \cdot \frac{N^2}{8^h}$
m_{opt}	0.68K
optimal h	$\lceil \log_8 \frac{N}{0.68K} \rceil, \lfloor \log_8 \frac{N}{0.68K} \rfloor$
FLOPs/particle at $\frac{N}{8^h} = m_{opt}$	$1323.5K(1 - 2.83K/N)$
II Total FLOPs	$N \cdot K \cdot (12M + 29) + K^2 \cdot (436.5 \cdot 8^d - 3748.5) + 1321 \cdot \frac{N^2}{8^h}$
m_{opt}	0.575K
optimal h	$\lceil \log_8 \frac{N}{0.575K} \rceil, \lfloor \log_8 \frac{N}{0.575K} \rfloor$
FLOPs/particle at $\frac{N}{8^h} = m_{opt}$	$1560.2(1 - 2.40K/N)$
III Total FLOPs	$N \cdot K \cdot (13M + 34) + K^2 \cdot (436.5 \cdot 8^d - 3748.5) + 1536 \cdot \frac{N^2}{8^h}$
m_{opt}	0.533K
optimal h	$\lceil \log_8 \frac{N}{0.533K} \rceil, \lfloor \log_8 \frac{N}{0.533K} \rfloor$
FLOPs/particle at $\frac{N}{8^h} = m_{opt}$	$1678.4K(1 - 2.23K/N)$

Table 7: FLOP count and optimal hierarchy depth of Anderson’s method.

The timings breakdown for the potential field calculation of 100 million particles on a 256 node CM-5E is shown in Table 8 for $K = 12$ and $K = 72$. The hierarchy depths are 8 and 7, respectively. The predicted optimal hierarchy depth based only on the number of floating-point (FLOP) operations using Method III are 7.97 and 7.10, respectively. Thus, for $K = 12$, the FLOP counts for the hierarchy and for the direct evaluation are very balanced. In fact, they differ by about 10%. Furthermore, the FLOP rates for $K = 12$ using Method III are 55.6 and 46.8 Mflops/s/PN, respectively. The overall FLOP rate is 43.7 Mflops/s/PN, with sorting accounting for most of the degradation in the overall FLOP rate. For $K = 72$ the FLOP rates for traversing the hierarchy, the direct evaluation, and overall are 81.8, 52.9, and 56.6 Mflops/s/PN, respectively.

The communication time for $K = 12$ is 22.3% of the total running time and 10% for $K = 72$, demonstrating that our techniques for reducing and managing data motion are very effective. The communication time includes the time for sorting the input particles, reshaping 1-D particle arrays to 4-D particle arrays, the multigrid functions in parent-child and neighbor interactions, the fetching of ghost boxes in neighbor interactions at all levels, replicating translation matrices for T_2 at every level, and the CSHIFTs in the near-field direct evaluation for fetching particles in the near-field boxes.

The computation time is 77.7% of the total running time for $K = 12$ and 90% for $K = 72$. In the computation time includes the time for forming the far-field potential for leaf-level boxes, the BLAS operations for the T_1, T_2

Figure 17: FLOP count per particle for optimal hierarchy depth, $K=12$.

and T_3 translations, the copying in the aggregation of BLAS operations for better arithmetic efficiency in T_2 , the masking in distinguishing boundary boxes from interior boxes in T_2 , the evaluation of the potential due to particles in the far-field, and finally the direct evaluation in the near-field.

One concern for hierarchical methods is the impact of the poor load-balance close to the root. Figure 9 shows the fraction of time spent in the various levels of the hierarchy for $K = 12$ and $K = 72$ and the optimal depths (eight and seven, respectively). The impact of the poor load-balance close to the root is minor.

11.2 Detailed performance results

The performance results reported here are all based on uniform particle distributions, cubic grids, and $K = 72$ integration points on the spheres. The measurements were carried out on a 256 node CM-5E, using hierarchies of depth 4, 5, 6, and 7. The number of particles on each VU ranges from 1 up to 65,536. The maximum number of particles per VU is 30,000 for 8 Mbytes/VU, and 120,000 for 32 Mbytes/VU. In the following, FLOP *rate* refers to the number of FLOPs per second. Whenever the FLOP counts for Method I, II and III differ, then the FLOP rates for all three methods are plotted. The peak performance numbers are for Method III.

11.2.1 Outer-sphere computations

The FLOP rates for the leaf-level outer-sphere computations are shown in Figure 18. Using the 4-D array representation of particle distributions, the execution time is expected to be linear in the number of particles per leaf-level box. However, for a hierarchy of fixed depth, before the number of particles equals the number of leaf-level boxes, the execution time is constant, and the FLOP rate doubles when the number of particles doubles. Once there is at least one particle per leaf-level box, the FLOP rate grows slowly as the subgrid size of the arrays increases and the various overheads are amortized over more operations. The peak FLOP rate of about 48 Mflops/s/PN is achieved for any hierarchy depth with a sufficient number of particles and leaf-level boxes per VU.

	$K = 12$		$K = 72$	
	time (sec.)	% of total	time (sec.)	% of total
Communication	39.75	22.3	89.01	9.99
sort	19.60	11.0	16.04	1.80
reshape	2.618	1.47	2.482	0.28
upward pass – multigrid in T_1	0.107	0.06	0.092	0.01
downward pass	8.410	4.71	56.39	6.33
– multigrid in T_3	0.215	0.12	0.162	0.02
– multigrid in T_2	0.484	0.27	0.385	0.04
– fetching ghost boxes in T_2	5.160	2.89	8.610	0.97
– replicate (T_2)	2.550	1.43	47.23	5.30
near-field – CSHIFTs	9.013	5.05	14.01	1.57
Computation	138.6	77.7	802.2	90.01
precompute T_1 matrices	0.006	0.00	0.575	0.06
precompute T_3 matrices	0.005	0.00	0.572	0.06
precompute T_2 matrices	0.003	0.00	0.235	0.03
init-potential	2.506	1.40	14.01	1.57
upward pass – BLAS for T_1	0.783	0.44	3.459	0.39
downward pass	63.62	35.7	166.5	18.7
– BLAS for T_3	0.601	0.34	4.320	0.48
– BLAS for T_2	34.98	19.6	141.6	15.9
– copy in T_2	12.90	7.23	9.990	1.12
– masking in T_2	15.14	8.49	10.53	1.18
far-field	4.678	2.62	90.74	10.2
near-field – direct evaluation	65.63	36.8	525.2	58.9
near-field – masking	1.371	0.77	0.952	0.11
Total	178.4	100	891.2	100

Table 8: The breakdown of the communication and computation time for 100 million particles on a 256 node CM-5E.

Level d		2	3	4	5	6	7	8
% of total	$K = 12, h = 8$	0.50	1.11	1.02	1.14	2.19	11.3	82.8
	$K = 72, h = 7$	1.47	5.49	5.26	5.98	13.4	68.4	

Table 9: The fraction of time spent at the different levels of the hierarchy on a 256 node CM-5E.

Figure 18: FLOP rate for the leaf-level outer-sphere computations, $K=72$. a) Method I, b) Method II, c) Method III.

Figure 19: FLOP rates for box-box outer-sphere computations as a function of the number of boxes per VU, $K=72$.

Figure 20: The FLOP rate for neighbor interactions as a function of the number of boxes per VU, $K=72$.

Since division and square root operations are used only at the leaf-level, Method I, II and III yield the same FLOP counts for all levels of outer-sphere computations, except the leaf-level. Figure 19 shows that for outer-sphere computations at all levels except the leaf-level, the larger the subgrids the more efficient the CMSSL BLAS. Figure 19 represents the average performance over all levels, except the leaf-level. Hence, the data accounts for the fact that the larger the subgrids the larger the fraction of parent-child interactions that are local to a node as well as the increased efficiency with subgrid size. Thus, though the peak performance for the outer-sphere computations is 96 Mflops/s/PN for $S1 = S2 = 16, S3 = 8$ and $K = 72$, the peak average performance is only 24 Mflops/s/PN.

11.2.2 Inner-sphere computations

Inner-sphere computations contain parent-child interactions and neighbor interactions; the latter dominates the former. For the neighbor interactions the CMSSL BLAS yields 136 Mflops/s/PN. The array copying, masking, and fetching of ghost boxes and averaging over all levels reduce the overall peak FLOP rate to 82 Mflops/s/PN for a depth seven hierarchy, which has 2048 leaf-level boxes on each VU of a 256 node CM-5E. The overall

Figure 21: FLOP rates for the local-field/particle computations, $K=72$. a) Method I, b) Method II, c) Method III.

FLOP rate as a function of the hierarchy depth is shown in Figure 20.

11.2.3 Leaf-level local-field potentials

Computing the local-field potential at the particles after traversing the hierarchy is similar to the leaf-level outer-sphere computations. The evaluation of the local-field approximations at the particles is less efficient than forming outer-sphere approximations, because the former has a higher ratio of load/store operations to floating-point operations than the latter. The peak achieved rate is 32 Mflops/s/PN, as shown in Figure 21.

11.2.4 Direct evaluation

The direct evaluation in the near-field uses 4-D particle arrays, just as the leaf-level outer-sphere computations, and the performance is similar. The FLOP rate doubles as the number of particles double until there is one particle in every leaf-level box. After that, the performance improves as the subgrid size of the particle arrays increases, until a peak of 55 Mflops/s/PN is reached, as shown in Figure 22. Note that the peak performance for the direct evaluation decreases with the depth of the hierarchy for a fixed number of particles. The reason is that the CSHIFTs used in fetching neighbor boxes is relatively more expensive in a deeper hierarchy. The CSHIFTs are dominated by exchanging the particles in the boxes on the surfaces of the 3-D subgrids in each VU. For a fixed number of particles on each VU, adding one more level to the hierarchy increases the number of boxes on the surface of the subgrid by a factor of four, and decreases the number of particles per box by a factor of eight. Thus, the number of particles in the surface boxes in each VU decreases by a factor of two. But, the computation which has complexity $O(N/8^h)$ decreases by a factor of eight. Therefore, the CSHIFTs become relatively more expensive.

11.3 Total execution rates and times

Figure 23 shows the overall FLOP rates as well as the FLOP rates for the downward pass and the direct evaluation in the near-field for optimal hierarchy depths as a function of the number of particles. For most

Figure 22: FLOP rates for the direct evaluation as a function of the number of particles per box, $K=72$. a) Method I, b) Method II, c) Method III.

Figure 23: Overall FLOP rates at optimal hierarchy depths, Method III, $K = 72$

cases, the FLOP rate for the downward pass is lower than that for the near-field direct evaluation, because the percentage of communication in the former is higher than that in the latter. The higher peak arithmetic efficiency in the direct evaluation does not have a measurable impact until the hierarchy has several levels local to a VU. The overall FLOP rate is lower than the average of the FLOP rates for the above two time-dominant stages, mainly due to the cost of sorting of the input particles.

Figure 24 provides some additional insight into the overall FLOP rates as a function of the number of particles. At the optimal hierarchy depth, the time for traversing the hierarchy and direct evaluation in the near-field are approximately equal, while hierarchy traversal dominates the FLOP rate when the number of particles per leaf-level box is below optimal, and the direct evaluation dominates when the particles per box is above optimal. Traversing the hierarchy has a higher FLOP rate than the direct evaluation for sufficiently deep hierarchies. This is the reason why the FLOP rate decreases for hierarchies of depth seven before reaching 65,536 particles per VU, for which depth seven is optimal.

The significance of choosing the optimal hierarchy depth is illustrated in Figure 25. Table 10 lists the timings for different stages of the method using optimal hierarchy depths, as well as the predicted optimal hierarchy depths using the FLOP count formulas from Section 11.1. It is clear from the timings that the downward pass and the near-field direct evaluation are the two time-dominant stages and are about balanced when the number of particles per leaf-level box is optimal. Table 10 also shows that the optimal hierarchy depth predicted by the

Figure 24: Overall FLOP rates, $K=72$. a) Method I, b) Method II, c) Method III.

Figure 25: Total running time, $K = 72$.

FLOP count formulas matches the measured optimal depth well. The cases when the two differ by one are when both depths give very similar total running time. For example, for 128K particles, the total running time using the predicted hierarchy depth 4 is 29.4 seconds, compared with 24.6 seconds using measured optimal hierarchy depth 3.

The execution times for $K = 12$ are about 4–10 times lower than those for $K = 72$. The efficiencies for $K = 12$ are at most 30% lower than for $K = 72$, and in many cases, about the same as for $K = 72$. For the same number of particles, the optimal hierarchy depth for $K = 12$ is often one larger than that for $K = 72$. Therefore the efficiency of traversing the one-level deeper hierarchy for $K = 12$ is often comparable to that of traversing the hierarchy for $K = 72$. The efficiencies for the direct evaluation remain about the same once there are relatively large number of particles per box.

11.4 Scalability

The speed of our code scales linearly with the number of nodes and number of particles. For a fixed number of particles per node, the efficiency remains independent of the number of nodes. For example, the numbers listed on the diagonal of Table 11 are for the same number of particles per node, and their running time for optimal hierarchy depths remains almost the same.

Particles N	1K	2K	4K	8K	16K	32K	64K	128K	256K
Particles/VU	1	2	4	8	16	32	64	128	256
optimal h	2	2	2	2	2	3	3	3	4
pred. optimal h	2	2	2	3	3	3	4	4	4
sort	0.014	0.014	0.015	0.015	0.017	0.020	0.026	0.033	0.059
reshape	0.025	0.050	0.101	0.200	0.398	0.109	0.218	0.442	0.082
init-potential	0.006	0.008	0.012	0.023	0.044	0.013	0.023	0.045	0.038
upward pass	0.000	0.000	0.000	0.000	0.000	0.374	0.376	0.376	0.772
downward pass	3.424	3.428	3.426	3.428	3.427	15.20	15.29	15.21	27.28
far-field	0.025	0.039	0.066	0.125	0.247	0.068	0.125	0.249	0.249
near-field	0.075	0.182	0.550	2.017	7.643	0.578	2.076	7.630	2.070
total	4.149	4.302	4.749	6.386	12.36	16.94	18.71	24.57	31.13

Particles N	512K	1M	2M	4M	8M	16M	32M	64M	
Particles/VU	512	1K	2K	4K	8K	16K	32K	64K	
optimal h	4	5	5	6	6	6	7	7	
pred. optimal h	5	5	5	6	6	6	7	7	
sort	0.093	0.195	0.476	0.781	1.392	2.754	5.818	11.24	
reshape	0.164	0.055	0.107	0.107	0.213	0.423	0.744	1.490	
init-potential	0.074	0.144	0.288	0.581	1.160	2.322	4.691	9.340	
upward pass	0.772	0.791	0.789	0.968	0.968	0.965	3.545	3.552	
downward pass	27.23	40.73	40.73	68.23	68.24	69.04	224.5	225.5	
far-field	0.502	0.974	1.937	3.804	7.635	15.28	30.29	60.52	
near-field	7.667	4.127	15.51	8.414	31.37	121.1	63.58	243.6	
total	37.08	47.61	60.44	83.52	111.7	212.7	334.3	556.8	

Table 10: Execution times in seconds for potential field evaluation on a 256 node CM-5E, $K = 72$.

Table 11 summarizes the timings for a variety of CM-5 systems with the number of particles, N , ranging from 1 million to 60 million. The timings were collected on CM-5s due to the unavailability of a variety of configurations of CM-5E systems. For all cases, the particles were uniformly distributed in a 3-D cubic domain. Twelve integration points per sphere are used for all simulations. The optimal hierarchy depth is used for all but the 60 million particle system. For this case, an execution time of 153 seconds on a 512-node CM-5 was required, or 2.55 micro-seconds per particle update. The optimal hierarchy depth of 8 would require more than 32 Mbytes/PN. A careful extrapolation of the case with 12.5 million particles on a 128-node CM-5 to 100 million particles on a 1024-node CM-5 predicts an execution time of 80 seconds for the potential field evaluation, with a sustained performance of 40 Gflops. This timing corresponds to about 1 micro-second per particle update. To our knowledge, both the number of particles and the running time are the best reported to date. Furthermore, the running time is within a factor 3 of that reported by using the best short range MD code [24] on the CM-5.

Table 12 summarizes the timings and the FLOP rates for runs using 72 integration points. As shown in Table 7,

Particles	Processors				
	32	64	128	256	512
1,024,000	24.67(30.06)	15.20(24.01)	11.04(18.79)	9.484(9.622)	8.286(12.24)
2,048,000	42.31(32.46)	24.06(27.62)	15.51(22.14)	11.98(13.87)	9.268(8.962)
4,096,000	107.4(34.56)	55.20(33.33)	31.47(30.70)	20.36(22.59)	13.72(16.76)
8,192,000		165.3(37.55)	45.73(32.08)	28.41(25.70)	18.08(20.19)
16,384,000			70.06(32.44) [†]	47.41(28.04)	27.47(24.19)
32,768,000				101.3(31.33) [†]	60.29(30.51)
60,000,000					153.0(35.9)

Table 11: Timing results for potential field evaluation (Mflops/s/PN in parenthesis), $K = 12$. [†] is for 16,000,000 particles, and ^{††} is for 30,000,000 particles.

the FLOPs/particle at optimal hierarchy depth is linear in K and thus increases with K , and therefore the running time increases with K . But, because the efficiency also increases, the running time does not increase in direct proportion to K . There are two reasons for the increase in FLOP rate with an increase in K . First, the BLAS operations for the translations in the hierarchy traversal achieves higher efficiency with larger K due to larger operands; translation matrices are of shape $K \times K$. Second, the arithmetic operations in the translations grows as K^2 while the amount of communication in fetching ghost boxes is linear in K .

Particles	Processors				
	32	64	128	256	512
1,024,000	121.9(33.98)	89.14(23.23)	69.90(15.10)	70.00(7.394)	71.41(14.50)
2,048,000	240.0(37.65)	150.0(30.13)	100.8(23.20)	85.87(13.15)	79.18(28.53)
4,096,000	661.5(40.76)	374.7(37.42)	145.8(39.75)	117.6(24.20)	103.7(13.72)
8,192,000		1198.9(41.00)	322.7(39.25)	154.9(26.73)	122.4(16.92)
16,384,000			443.3(39.43) [†]	271.7(33.26)	183.7(24.60)
32,768,000				628.9(38.11) [†]	408.9(34.29)

Table 12: Timing results for potential field evaluation (Mflops/s/PN in parenthesis), $K = 72$. The rate of error decay is 9. [†] is for 16,000,000 particles, and ^{††} is for 30,000,000 particles.

11.5 Memory usage

The memory requirements of our implementation of Anderson’s method is shown in Table 13. The two bottlenecks are the downward pass of the hierarchy and the direct evaluation in the near-field. In fetching only ghost boxes from other nodes in the downward pass, memory is traded for efficient communication in neighbor interactions. The direct evaluation uses the duplicate-and-slide technique for computational efficiency in all-to-all particle interactions between boxes. At the optimal depth, $\frac{N}{8h} = m_{opt} = 0.533K$ using method III, the formulas in Table 13 predicts a memory usage of 230 Bytes/particle. Thus, for the optimal hierarchy depth, 30,000 particles can be represented in each VU with 8 Mbytes/VU and 120,000 particles fit in 32 Mbytes. Note

stage	memory used (double words)
input particle arrays	$5N$
init-potential & upward pass	$17.5N + (K + 9.5)8^h$
downward pass	$5.5N + (9K + 1.5)8^h$
far-field	$17.5N + (K + 1.5)8^h$
near-field	$23N + (K + 25)8^h$

Table 13: Memory requirements of Anderson’s method.

that this number does not change (actually decreases a little) as K increases, since the optimal hierarchy depth will decrease.

Both the memory usage and the execution time for traversing the hierarchy and the direct evaluation in the near-field are balanced at the optimal hierarchy depth in our implementation. For example, using Method II, the execution time for the two dominating stages are balanced at $\frac{N}{8^h} = 0.575K$; while the memory usage for the two dominating stages are balanced at $\frac{N}{8^h} = 0.457K$. If the technique for fetching and storing ghost boxes in the downward pass is not used, then the arrays for embedding the hierarchy will use less memory but run slower. This implies that fewer levels should be used in the hierarchy, and thus the hierarchy would use even less memory. The opposite situation occurs if the duplicate-and-slide technique is not used in the direct evaluation. Thus, we claim that our implementation is memory-efficient.

12 Discussions

Nonadaptive hierarchical methods use nonadaptive domain decomposition, and the hierarchy of recursively decomposed domains is balanced. There are three sources of parallelism in traversing the hierarchy. First, the computations in the parent-child interactions for all boxes at the same level can be performed in parallel in the upward pass and the downward pass of the hierarchy. Second, at every level of the downward pass of the hierarchy, the conversion of the far-field potential of each box’s interactive-field boxes into the local-field potential of that box can be performed in parallel. Third, since neighbor interactions are between boxes at the same level, the neighbor interactions at all levels can be performed in parallel.

We only exploit parallelism among boxes at the same level of the hierarchy, which potentially could result in poor load-balance. However, hierarchical methods are advantageous compared to direct methods only when more than a few thousand particles are considered. Since for the optimal depth of the hierarchy there only are a few particles per leaf-level box, the number of leaf-level boxes is at least about a hundred. For large-scale simulations there may be several million leaf-level boxes. Hence, for most interesting simulations there is excess parallelism even for the largest of MPPs, not only at the leaf-level but also for several levels close to the leaf-level. At levels of the hierarchy close to the root, there are much fewer boxes per level, and the cost of computation is already insignificant. Hence, although the load may be unbalanced, improved load-balance will not affect the total execution time significantly. Also, a program that traverses the hierarchy level by level and

Phase	Computation	Communication	Comp. vs. Comm.
sort		locality in particle-box interactions	
leaf-level particle-box interactions		“local” send and scan	
upward pass & downward pass 1. parent-child interactions 2. neighbor interactions	BLAS for translations supernodes	local copy or two-step scheme on factored arrays	computation + spread of T_i 's
direct evaluation particle-box interactions 1. neighbor interactions 2. all-to-all	duplicate-&-slide	align particle arrays with leaf-level boxes CSHIFTs with linear ordering	

Table 14: Summary of optimization techniques used in our data parallel implementation of $O(N)$ N -body methods.

sequentializes neighbor interactions requires much simpler data and control structures than one that exploits parallelism beyond just among boxes at the same level.

The computations in traversing the hierarchy are box-box interactions, each of which involves the same amount of computation. The computations for the particle-box interactions at the leaf-level of the hierarchy do not have the same uniformity since the particle distribution matter. For a nonuniform distribution of particles the reshaping of the 1-D particle arrays into 4-D arrays following the coordinate sort can result in extensive communication and load-imbalance. Furthermore the memory utilization may be poor, since some boxes may be empty and others may have a very large number of particles, but all of them occupy the same amount of memory on each VU.

Using 1-D arrays for particle-box interactions results in more balanced computations, since 1-D arrays will always be laid out across the VUs evenly on the CM-5/5E, which also implies balanced memory usage. But, the need for communication may increase significantly. In general, an adaptive hierarchical method is needed in order to achieve good performance for nonuniform particle distributions.

13 Conclusions

We have presented optimization techniques for programming $O(N)$ N -body algorithms for MPPs. The optimizations mainly focus on minimizing the data movement through careful management of the data distribution and the data references and on improving arithmetic efficiency through aggregating translation operations into high-level BLAS operations. The optimizations are summarized in Table 14.

The effectiveness of our techniques is demonstrated on an implementation in Connection Machine Fortran of Anderson's hierarchical $O(N)$ N -body method. The evaluation of the potential field of 100 million uniformly

distributed particles and $K = 12$ integration points on the sphere takes 180 seconds on a 256 node CM-5E, with an efficiency of about 27% of the peak performance. For $K = 72$ integration points the efficiency is about 35%. The amount of memory required for a particle at optimal hierarchy depth is about 230 Bytes, independent of the error decay rate of the method.

Acknowledgment

We would like to thank Christopher Anderson for providing us with the sequential program and for many helpful discussions. We also thank the National Center for Supercomputer Applications at the University of Illinois at Urbana/Champaign, the Navy Research Laboratory Washington D.C. and the Massachusetts Institute of Technology for providing Connection Machine system CM-5 access. The support of the Office of Naval Research through grant N00014-93-1-0192 and the Air Force Office of Scientific Research through grant F49620-93-1-0480 is gratefully acknowledged.

References

- [1] Christopher R. Anderson. An implementation of the fast multipole method without multipoles. *SIAM J. Sci. Stat. Comp.*, 13(4):923–947, July 1992.
- [2] Christopher R. Anderson. An implementation of the fast multipole method without multipoles. *SIAM J. Sci. Stat. Comp.*, 13(4):923–947, July 1992.
- [3] James H. Applegate, Michael R. Douglas, Yekta Gursel, Peter Hunter, Charles L. Seitz, and Gerald J. Sussman. A digital orrery. *IEEE Trans. on Computers*, C-34(9):822–831, September 1985.
- [4] Appel A.W. An efficient program for many-body simulation. *Siam J. Sci. Stat. Comput.*, 6(1):85–103, 1985.
- [5] Josh Barnes and Piet Hut. A hierarchical $o(n \log n)$ force calculation algorithm. *Nature*, 324:446–449, 1986.
- [6] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM J. of Scientific and Statistical Computations*, 9(4):669–686, July 1988.
- [7] Jack J. Dongarra, Jeremy Du Croz, Iain Duff, and Sven Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. Technical Report Reprint No. 1, Argonne National Laboratories, Mathematics and Computer Science Division, August 1988.
- [8] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An Extended Set of Fortran Basic Linear Algebra Subprograms. Technical Report Technical Memorandum 41, Argonne National Laboratories, Mathematics and Computer Science Division, November 1986.
- [9] W. D. Elliott and J. A. Board. Fast fourier transform accelerated fast multipole algorithm. Technical Report 94-001, Dept. of Electrical Engineering, Duke Univ., 1994.
- [10] Klaas Esselink. The order of Appel’s algorithm. *Information Processing Letter*, 41:141–147, 1992.

- [11] High Performance Fortran Forum. High performance fortran; language specification, version 1.0. *Scientific Programming*, 2(1 - 2):1–170, 1993.
- [12] Leslie Greengard and William D. Gropp. A parallel version of the fast multipole method. In *Parallel Processing for Scientific Computing*, pages 213–222. SIAM, 1989.
- [13] Leslie Greengard and V. Rokhlin. On the efficient implementation of the fast multipole method. Technical Report YALEU/DCS/RR-602, Dept. of Computer Science, Yale Univ., New Haven, CT, February 1988.
- [14] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.
- [15] Leslie Greengard and Vladimir Rokhlin. Rapid evaluation of potential fields in three dimensions. Technical Report YALEU/DCS/RR-515, Dept. of Computer Science, Yale Univ., New Haven, CT, February 1987.
- [16] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantative Approach*. Morgan Kaufmann Publishers, Inc, 1990.
- [17] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.
- [18] J. A. Board Jr., Z. S. Hakura, W. D. Elliott, D. C. Gray, W. J. Blanke, and J.F. Leathrum Jr. Scalable implementations of multipole-accelerated algorithms for molecular dynamics. In *Proc. Scalable High Performance Computing Conference SHPCC94*, 1994.
- [19] J. Katzenelson. Computational structure of the N-body problem. *SIAM J. Sci. Statist. Comput.*, 4:787–815, 1989.
- [20] C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM TOMS*, 5(3):308–323, September 1979.
- [21] James F. Leathrum. *The parallel fast multipole method in three dimensions*. PhD thesis, Duke University, 1992.
- [22] James F. Leathrum and John A. Board. The parallel fast multipole algorithm in three dimensions. Technical Report Report CSL-TR-92-506, Department of Computer Science, Duke University, April 1992.
- [23] Pangfeng Liu. *The parallel implementation of N-body algorithms*. PhD thesis, Yale University, 1994.
- [24] P. S. Lomdahl, P. Tamayo, N. Gronbech-Jensen, and D. M. Beazley. 50 GFlops molecular dynamics on the Connection Machine 5. In *Supercomputing '93*, pages 520–527. IEEE, 1993.
- [25] A. D. McLaren. Optimal numerical integration on a sphere. *Math. Comput.*, 17:361 – 383, 1963.
- [26] L. S. Nyland, J. F. Prins, and J. H. Reif. A data-parallel implementation of the adaptive fast multipole algorithm. In *Proceedings of the DAGS '93 Symposium*. Dartmouth Institute for Advanced Graduated Studies in Parallel Computation, Hanover, NH, 1993.
- [27] John K. Salmon. Parallel heirarchical n -body methods. Technical Report CRPC-90-14, California Institute of Technology, 1990.

- [28] K. E. Schmidt and M. A. Lee. Implementing the fast multipole method in three dimensions. *J. Stat. Phys.*, 63(5/6), 1991.
- [29] J.P. Singh, Chris Holt, J.L. Hennessey, and Anoop. Gupta. A parallel adaptive fast multipole method. In *Supercomputing '93*, pages 54 – 65. IEEE, 1993.
- [30] J.P. Singh, Chris Holt, Takashi Tsuka, Anoop. Gupta, and J.L. Hennessey. Load balancing and data locality in hierarchical N-body methods. Technical Report CSL-TR-92-505, Stanford University, 1992.
- [31] Thinking Machines Corp. *CM Fortran Reference Manual, Version 2.1*, 1993.
- [32] Thinking Machines Corp. *CMSSL for CM Fortran, Version 3.1*, 1993.
- [33] M. Warren and J. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. In *Supercomputing '92*, pages 570 – 576. IEEE, 1992.
- [34] M. Warren and J. Salmon. A parallel hashed oct-tree N-body algorithm. In *Supercomputing '93*, pages 12 – 21. IEEE, 1993.
- [35] Feng Zhao. An $O(N)$ algorithm for three-dimensional N-body simulations. Technical Report AI Memo 995, MIT, Artificial Intelligence Laboratory, October 1987.
- [36] Feng Zhao and S. Lennart Johnsson. The parallel multipole method on the Connection Machine. *SIAM J. Stat. Sci. Comp.*, 12(6):1420–1437, 1991.