A Data-Parallel Implementation of $O(N)$ Hierarchical N-body Methods

Citation

Hu, Yu, and S. Lennart Johnsson. 1996. A Data-Parallel Implementation of $O(N)$ Hierarchical N-body Methods. Harvard Computer Science Group TR-05-96.

Permanent link

<http://nrs.harvard.edu/urn-3:HUL.InstRepos:24019784>

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 $O(N)$
Hierarchical N -body Methods**

Yu Hu
S. Lennart Johnsson

TR-05-96

May 1996

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

To appear in *Proceedings of Supercomputing '96*.

A Data-Parallel Implementation of $O(N)$ Hierarchical N -body Methods

Yu Hu[†] and S. Lennart Johnsson^{†‡}

[†]Division of Applied Sciences, Harvard University
Cambridge, Massachusetts 02138

[‡]Department of Computer Sciences, University of Houston
Houston, Texas 77204

Email: hu@das.harvard.edu, johnsson@cs.uh.edu

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 present 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%. The evaluation of the potential field of a system of 100 million particles takes 3 minutes and 15 minutes on a 256 node CM-5E, giving expected four and seven digits of accuracy, respectively. The speed of the code scales linearly with the number of processors and number of particles.

Keywords: N -body simulation, multipole algorithms, hierarchical N -body methods, data-parallel programming, massively parallel processors.

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 [13, 14, 12, 6, 36, 1]. The constant of proportionality is in the range 1,000–10,000. Earlier hierarchical algorithms, such as those proposed by Appel [2] and by Barnes and Hut [4], were believed to have an arithmetic complexity of $O(N \log N)$ and did not have a rigorous error bound, although Appel's version was later proved to be of $O(N)$ [10]. The two methods were later extended to be of $O(N)$ with analytical error bounds and by combining with the idea of multipole expansions [11, 35].

Parallel implementation of the $O(N \log N)$ or $O(N)$ hierarchical N -body methods have been of great interests as Massively Parallel Processors (MPPs) offer the primary storage and compute power for simulation of systems with several hundred million particles by using these fast algorithms. Table 1 gives a summary of sequential and parallel implementations of hierarchical N -body methods. In comparing performance

Author	Method & error	Prog. Model	Performance				
			N	Effi.	Cycles/ particle	P	System
adaptive $O(N \log N)$ methods							
Salmon [27]	BH, quadrupole	MP					Ncube
Warren–Salmon [33]	BH, quadrupole	MP	8.78M	26%	180K	512	Intel Delta
Warren–Salmon [34]	BH, $\epsilon_1 = 10^{-3}$	MP	8.78M	28%	266K	512	Intel Delta
Warren–Salmon [35]	BH, $\epsilon_1 = 10^{-2}$	MP	2M		111K	256	CM–5E
Liu–Bhatt [24]	BH, quadrupole	MP	10M	30%	97K	256	CM–5
Singh et al. [29]	BH	SM					DASH, KSR–1
nonadaptive $O(N)$ methods							
Leathrum–Board [23]	GR, p=8	SM	100K	65%	250K	1	RS/6000–360
	GR, p=8		1M	20%		32	KSR-1
Elliott–Board [9]	GR, FFT, p=8	SM	100K	73%	200K	1	RS/6000–360
	GR, FFT, p=8		1M	14%		32	KSR–1
Schmidt–Lee [28]	GR, p=8		40K	39%	312K	1	Cray YMP 8/864
	GR, p=16		40K	22%	1034K	1	Cray YMP 8/864
Zhao–Johnsson [37]	Zhao, p=3	DP	16K	12%	560K	8K	CM–2
Hu–Johnsson (this work)	Anderson, D=5	DP	100M	27%	37K	256	CM–5E
	Anderson, D=14	DP	100M	35%	183K	256	CM–5E
adaptive $O(N)$ methods							
Singh et al. [30]	GR, 2–D, adap	SM					DASH, KSR–1
Nyland et al. [26]	GR, 3–D, adap	DP					

Table 1: A summary of sequential and parallel implementations of hierarchical N –body methods. All performance numbers are for uniform particle distributions. Methods used are for three dimensions, unless otherwise stated. ϵ_1 is the error bound per partial acceleration relative to the mean acceleration of the system. Empty entries imply unavailable data. MP, SM, and DP are short for message–passing, shared–memory, and data–parallel, respectively.

results from implementations of different N –body methods, often with different parameters, on different platforms running at different clock speed, we propose the use of *efficiency* of floating–point operations and *cycles–per–particle* as the standard measure. Efficiency alone is insufficient in comparing differ algorithms that require different number of operations. Cycles–per–particle incorporates machine size, clock rate, arithmetic complexities of different methods, but it does not distinguish nodal architecture, e.g., superscalar architectures can perform multiple operations per cycle.

Barnes and Hut’s $O(N \log N)$ method has been implemented using the message passing programming paradigm by Salmon and Warren [27, 33, 34] on the Intel Touchstone Delta and by Liu and Bhatt [24] on the CM–5. Both groups used assembly language for time critical kernels. Salmon and Warren achieved efficiencies in the range 24 – 28%, while Liu and Bhatt achieved 30% efficiency. Recently, Warren and Salmon [35] extended their code to incorporate multipole and local expansions and made it portable to a variety of parallel machines.

For nonadaptive $O(N)$ methods, Greengard and Gropp [12] implemented Greengard–Rokhlin’s method in 2–D on a shared–memory machine (the Encore Multimax 320), but data is not sufficiently complete for inclusion in Table 1. Zhao and Johnsson [37] developed a data–parallel implementation on the CM–2 of Zhao’s method, and achieved an efficiency of 12% for expansions in Cartesian coordinates, which yields more costly multipole expansion calculations than polar coordinates. Leathrum and Board [23, 5] and

Elliott and Board [9] achieved efficiencies in the range 14 – 20% in implementing Fast Fourier Transform accelerated Greengard–Rokhlin’s method [15] on the KSR–1. Schmidt and Lee [28] vectorized this method for the Cray Y–MP and achieved an efficiency of 39% on a single processor. For comparison, we have also included the results reported in this paper.

Little progress has been made in the implementation of adaptive $O(N)$ methods in distributed–memory machines. Singh et al. [29, 30] 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 Greengard–Rokhlin’s method [6] in a data–parallel subset of the Proteus language, which is still under implementation on parallel machines.

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 (CMF) [31] because no High Performance Fortran (HPF) [16] compiler was available at the time of this project. All but one of the features of CMF that we use are also available in HPF. Data motion is managed through the use 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 $O(N)$ hierarchical N –body methods on MPPs are

- minimal data motion in parent–children interactions,
- low data motion in neighbor interactions for interactive–field computations,
- redundant computation/communication trade–offs,
- representing translation operations as matrix–vector multiplications (level–2 BLAS),
- aggregating multiple independent translation operations into multiple instances of matrix–matrix multiplications (level–3 BLAS),
- reducing the number of translation operations through the use of supernodes,
- expressing hierarchical operations on “flattened” data structures efficiently in a data–parallel language,
- efficient memory usage.

Most of our optimization techniques apply to any distributed memory machine. However, the relative merit of the techniques depend upon machine metrics. We report on the performance trade–offs on the CM–5/5E.

To our knowledge, this work represents the first implementation of Anderson’s method on a parallel machine as well as the first implementation of an $O(N)$ N –body algorithm in a data–parallel language. Moreover, the efficiency of our implementation for particle systems with uniformly distribution is competitive to those highly efficient parallel implementations of Barnes–Hut’s algorithm using low–level message–passing and assembly language programming. Our implementation is also memory–efficient. To our knowledge, this is the first known long–range simulation that simulated systems of 100 million particles.

Section 2 briefly describes the computational structure of $O(N)$ N –body methods and the computational elements used in Anderson’s method, and defines several of the terms used in the above summary of contributions. Our optimization techniques for programming hierarchical methods in CMF are presented in Section 3. Section 4 reports the performance results of our implementation and the measured accuracy. Section 5 concludes the paper.

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

2 $O(N)$ N -body Methods

The $O(N)$ hierarchical N -body methods [14, 36, 1] share the same computational structure; they only differ in the computational elements used in approximating the aggregated potential or force due a cluster of faraway particles. We briefly describe the computational structure of the $O(N)$ methods and the computational elements used in Anderson's method in this section.

2.1 Domain Decomposition

The $O(N)$ methods start with refining the computational domain into a hierarchy of smaller and smaller subdomains (see Figure 1). 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*). 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 parallelepipedic domains in three dimensions [1].

With respect to each subdomain (box) in the hierarchy, the whole domain is partitioned into three regions. The definition of the three regions has a significant impact on the constant in the $O(N)$ asymptotic arithmetic complexity, as well as on the accuracy of the method. In the original formulation of multipole-based methods [14], the *near-field* is defined as those subdomains that share a boundary point with the considered subdomain in two dimensions, and those 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 in three dimensions. We denote these two kinds of near-fields as with *one-separation* and *two-separation*, respectively. In general, the d -*separation* near-field in two or three dimensions contain $(2d + 1)^2$ and $(2d + 1)^3$ subdomains, respectively. 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, these definitions yield $7(2d + 1)^3$ interactive-field subdomains.

In the rest of the paper, two-separation near-field is assumed unless otherwise stated.

2.2 Computational Structure

There are two key ideas in $O(N)$ methods that lead to the linear arithmetic complexity. The first, also used in $O(N \log N)$ methods, is to represent a cluster of particles sufficiently far away from an evaluation point by a single computational element, called *far-field potential representation*. The exact computational elements are represented by an infinite number of terms in the multipole-based methods or sphere integrations in Anderson’s method, and hence, in practice, are approximated by elements represented by a finite number of terms or discretized integrations. The $O(N)$ methods also introduces 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 second key idea is to hierarchically form and use as few computational elements as possible. The method is to hierarchically combine children’s far-field potential to form parent’s and pass parent’s local-field potential to children’s, as shown in the algorithm below.

$O(N)$ 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. 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. 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 subdomain i ’s far-field region, i.e., the local-field potential in subdomain i at level l . The computational structure is described as follows [21].

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-field potential at particle k inside every leaf-level subdomain

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

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,

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

2.3 Optimal Hierarchy Depth

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 using the Greengard–Rokhlin neighborhood definition. 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 hierarchy depth* is proportional to the 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 in two-separation [36, 12] reduces the effective value of N_{int} in three dimensions from 875 to 189, which brings about a dramatic improvement in the overall performance, at the cost of slightly decreased accuracy.

2.4 Anderson’s Method

Anderson [1] uses Poisson’s formula for representing solutions of Laplace 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 equations (2)–(4)). Another advantage is that the computations in two and three dimensions are very similar. Therefore, a code for three dimensions is easily obtained from a code for two dimensions, or vice versa.

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) in [1])

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

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 their corresponding weights w_i , the following formula (equation (15) in [1]) is used to approximate the potential at $\vec{x}$:

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

This approximation is called an *outer-sphere approximation*. In this approximation two approximations are made compared to Equation (1): the series is truncated, and the integral is discretized.

In approximating Poisson’s formula, one first chooses an integration order D , which determines the error decay rate of the approximation. One then chooses among different integration formulas the one requiring fewest integration points which translates into fewest arithmetic operations for the integration. The optimal choices of K , M , and a in Table 2 are given by Anderson [1].

The approximation used to represent potentials inside a given region is (equation (16) in [1])

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

Order of integration D	K	M ($\leq D/2$)	$\beta_0 = \frac{\alpha_{outer}}{\alpha}$	$\gamma_0 = \frac{\alpha}{\alpha_{inner}}$	Expected error decay rate (D/2+2)
5	12	2	10	2	4
7	24	3	7	2	5
9	32	4	5	2	6
11	50	5	2.5	2	7
14	72	7	2	2	9

Table 2: Parameter selections and expected error decay rate of outer/inner sphere approximations in Anderson’s method. α is the side length of a box.

Figure 2: Translations as evaluations of the approximations.

and is called an *inner-sphere approximation*.

The outer-sphere and 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 2. The situation is similar for the other two translations used in the method; 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).

3 A Data-Parallel Implementation

In this Section, we present a data-parallel implementation of Anderson’s method in CMF on the CM-5/5E. The optimizations mainly focus on minimizing the data movement through careful management of data distribution and data references and on improving arithmetic efficiency through aggregating field-translation operations into high-level BLAS operations. Most optimizations make use of the *array aliasing* feature of CMF [32].

Since a single processing node of CM-5/5E has four (virtual) Vector Units (VU), each with its own ALU, Register File, and memory, for clarity, we will use VUs instead of processing nodes in the following discussion.

Figure 3: Embedding of a hierarchy of grids in two 4-D arrays.

3.1 Data Structure and Distribution

Maximizing concurrency and minimizing communication among nodes are crucial in achieving high performance on distributed-memory machines in addition to exploiting spatial and temporal locality in the local memory hierarchies. The fact that data distribution, or *layout*, usually is not known until run-time further complicates memory management on distributed memory architectures. Run-time data allocation is the norm when (parallel) codes can be executed on systems with different configurations without recompilations.

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.

Far-field potentials are stored for all levels of the hierarchy, since they are computed in the upward pass and used in the downward pass. We embed the hierarchy of far-field potentials in one five-dimensional (5-D) array as follows (see Figure 3): the leaf-level is embedded in one layer of the 4-D array, i.e., $\text{FAR_POT}(1, :, :, :, :)$, and level $(h - i)$ is embedded in $\text{FAR_POT}(2, :, 2^{i-1} : L : 2^i, 2^{i-1} : M : 2^i, 2^{i-1} : N : 2^i)$. 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 a box. 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 allocated to the same VU as the box itself.

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 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 L, M , and 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 axis, 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 slices of 4-D subarrays in the 5-D potential array are shown in Figure 4.

Axis	Extent	VU address $b_{p+n-1}b_{p+n-2}\dots b_n$			Local memory address $b_{n-1}b_{n-2}\dots b_0$		
0	K				b..b		
1	L	b..b				b..b	
2	M		b..b				b..b
3	N			b..b			b..b

Figure 4: The allocation of the local potential arrays LOCAL_POT to processing nodes.

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 (step 1 and 4 in the algorithm) and the direct evaluation in the near-field (step 5), 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. The particle attributes in the 4-D arrays 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 with leaf-level boxes.

3.2 Particle-box Interactions at the Leaf-level

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.

For the leaf-level particle-box interactions before traversing the hierarchy, the contributions of all particles in a box to each integration point on the sphere in Anderson's method (or to each term in the multipole expansion for the box) must be accumulated. Different boxes have different number of particles, and therefore the number of terms added varies with the leaf-level boxes. 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. Similarly, a scan and a send communication are 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, 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). Unfortunately, as long as an array assignment involves arrays of different shape, e.g., 1-D particle arrays and 4-D potential arrays, the CMF compiler generates run-time system calls which handles the most general case, i.e., those that involves inter-node data movement. Such run-time system calls incur a high overhead even if inter-node data movement never occurs. Using the 4-D particle array representation can avoid such scenario. The 4-D particle arrays have the same three parallel axes as the 4-D potential arrays. The scan operations and the send communication become indexing on the fourth local axis, and no communication is required.

Now the problem turns into how to make the 1-D to 4-D reshaping of particle arrays efficient. Since

Figure 5: Sorting particles for maximum locality in reshaping particle arrays.

the input particles have to be sorted once to bring particles belonging to the same leaf-level box together, and the cost of sorting is relatively independent of distributions of source and destinations, we want the sort to maximize the locality in reshaping the sorted 1-D particle arrays to the 4-D particle arrays, in addition to bring particles belonging to the same box together. The following *coordinate sort* (see Figure 5) sorts particles based on keys constructed from the particle locations, the leaf-level box coordinates, and their allocation, and 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 sorted 1-D array will be allocated to the same VU as the leaf-level box to which it belongs in the local-field potential array. Therefore, no communication is needed in copying particle attributes from the sorted 1-D array to the 4-D array of particle attributes which has the same layout as the 4-D potentials array. 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.

3.3 Box-box Interactions during Hierarchy Traversal

During the upward pass, the combining of far-field potentials of child boxes to form the far-field potential of the parent box (T_1) require *parent-child box-box interactions*. During the downward pass, converting the local-field potentials for parent boxes to that for child boxes (T_3) also require *parent-child (box-box)*

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

Table 3: Leaf-level arithmetic efficiencies on a 256 node CM-5E. The aggregation of T_2 translations involves copying and masking.

interactions. The downward pass also requires *neighbor (box-box) interactions* for the conversion of the far-field potential of interactive-field boxes to local-field potentials (T_2).

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 Basic Linear Algebra Subroutines (BLAS) [8, 7, 22]. The Connection Machine Scientific Software Library (CMSSL) [31] supports both single-instance and multiple-instance BLAS.

For entire hierarchy traversal (step 2 and 3 of the algorithm), our techniques for optimizing the computations result in an arithmetic efficiency of 40% for $K = 12$ and a hierarchy of depth eight, and an arithmetic efficiency of 69% for $K = 72$ and a hierarchy of depth seven. Table 3 summarizes the leaf-level arithmetic efficiencies for $K = 12$ and $K = 72$ on a 256 node CM-5E. The peak arithmetic efficiency of about 74% for $K = 12$ and 85% for $K = 72$ at the leaf-level is degraded due to the cost of copying, masking, and a relatively lower efficiency at the higher levels of the hierarchy. 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 for $K = 12$ than for $K = 72$.

On the communication side, by using our techniques for avoiding excess data movement in prefetching interactive-field boxes in neighbor (box-box) interactions and for extracting/embedding parent and child boxes from the embedded potential arrays in parent-child interactions, communication only contributes 12% of the total time for hierarchy traversal for $K = 12$ and a hierarchy of depth eight. For $K = 72$ and a hierarchy of depth seven communication amounts to 25%.

3.3.1 Interactive-field Box-box Communication

The interactive-field computation dominates the hierarchical part of the code. The interactive-field of a child box contains all boxes inside a $10 \times 10 \times 10$ subgrid centered at the the center of the child box’s parent box, but outside a $5 \times 5 \times 5$ subgrid centered at the child 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. The near-field and interactive-fields of siblings differ. Each box needs to fetch the potential vectors of its 875 interactive-field boxes during the interactive-field computation.

The simplest way to fetch potential vectors of neighbor boxes is to use individual CSHIFTs, one for each neighbor, as shown in Figure 6(a). In the Connection Machine Run-Time System, CSHIFTs along more than one axis are implemented as a sequence of independent shifts, one for each axis, resulting in excessive

Figure 6: Optimizing communication in neighbor interactions.

data motion.

A better way to structure the CSHIFTs is to impose a linear ordering upon the interactive-field boxes, as shown in Figure 6(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. In some networks, like meshes and hypercubes, node-addresses are often defined such that nodes that differ in their lowest order bits are adjacent. In such networks, a linear ordering that uses CSHIFTs along the rightmost axis most often implies less data motion between nodes than other orderings. This shift order is also advantageous in our implementation due to the bandwidth limitations and the addressing of the fat-tree network of the CM-5/5E.

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, 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 (see Figure 6(c)). 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 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 is 7.4 times faster than direct use of CSHIFTs for a subgrid with axes extents 16 and $K = 12$.

In order to eliminate excess data movement, we identify non-local interactive-field boxes for all boxes in the local subgrid, and structure the communication to fetch only those 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

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

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

Table 4: 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 aliased arrays.

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 requires fetching six surface regions, 12 edge regions, and eight corner regions.

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 an unaliased array. Table 4 summarizes the data motion requirements for the four methods for a subgrid of shape $S1 \times S2 \times S3$ with $S1 = S2 = S3 = 8$.

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

3.3.2 Parent-child Box-box Communication

Using the embedding described in Section 3.1, 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 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 the overhead in computing send addresses, which is about linear in the array size, is smaller. This overhead

Figure 7: Performance improvement of Multigrid-embed using array sectioning and aliasing. The two-step scheme was used for the first two cases and local copying was used for the remaining cases.

may dominate the actual communication, which is proportional to the number of elements selected.

On the CM-5E, the performance of Multigrid-embed is improved by a factor of up to two orders of magnitude using the local copying or the two-step scheme, as shown in Figure 7.

3.3.3 Translations as BLAS Operations

In Anderson’s method, the translation operators evaluate the approximations of the field on the source spheres at the integration points of the destination spheres (see Figure 2). A field approximation (equation (2) or (3)) can be rewritten as

$$\Phi(\vec{x}_j) \approx \sum_{i=1}^K f(\vec{s}_i, \vec{x}_j) \cdot g(a\vec{s}_i), \quad j = 1, K, \quad (4)$$

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 unit vector $\vec{x}_j$ from the origin of the source sphere to the j th integration point on the destination sphere. The evaluation of the field at an integration point on the destination sphere due to the field values at all integration points on the source spheres is an inner-product computation. Hence, the evaluation of the field at all integration points on the destination sphere due to the field values at all the integration points of the source sphere constitutes a matrix-vector multiplication, where the matrix is of shape $K \times K$. We refer to this matrix as a *translation matrix*, since the net effect can be interpreted as a translation of the field from the integration points on the source sphere to the integration points on the destination sphere. The entries of the translation matrix only depends on the relative locations of the source and destination spheres.

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.

Combining the far-field of eight child boxes to form the far-field of their parent (T_1) can be expressed as eight matrix-vector multiplications, followed by an addition of the resulting vectors. Due to the symmetry

of the distribution of the integration points on the spheres, the eight matrices required to represent T_1 (T_3) are permutations of each other. One matrix can be obtained through suitable row and column permutations of another. Based on this fact, the above combining translation can be expressed as a matrix–matrix multiplication of the translation matrix and a matrix containing eight permuted potential vectors of the children as columns, followed by permutations of the columns of the product matrix which then are added to form the potential vector of the parent box. A similar approach can be used for T_3 . 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 3.3.5.

Translation Matrices for T_2 As described before, 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. 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.

3.3.4 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 scheduling operations at a given time.

In Anderson’s method aggregation of computations in parent–child interactions results in multiple–instance matrix–matrix multiplications. Since the same translation matrix is used for all parent–same–child interactions, the matrix–vector multiplications can be aggregated into a 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 routine. On a CM–5E, 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 16 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

minor improvement for this case is because the matrices are large that before aggregation, the matrix–vector multiply already achieves good performance.

For the far–field to local–field conversions in the interactive–field interactions, the union of the interactive–fields of all the boxes in a subgrid is four boxes deep on all faces of the subgrid, and is prefetched and stored in a subgrid of shape $(S1 + 8) \times (S2 + 8) \times (S3 + 8)$. Similar to parent–child interactions, each interactive–field conversion for a single box pair is performed as a matrix–vector multiplication. Since all box pairs with the same relative location use the same translation matrix, conversions for all local boxes and their corresponding interactive–field boxes with the same relative location can be aggregated into a single matrix–matrix multiplication. For the interactive–field computations we rearrange the arrays in local memory via copying 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 is also an alternative in parent–child interactions, but is not used due to its relatively higher cost. See [17] for detail.

3.3.5 Precomputing Translation Matrices

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. to compute all the translation matrices on every VU,
2. to 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 [20]. 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

Figure 8: Computation vs. replication in precomputing translation matrices for T_1 (T_3) on a 256 node CM-5E.

Figure 9: Computation vs. replication in precomputing translation matrices for T_2 on the CM-5E.

replication is always a winning choice. For T_1 and T_3 replication among eight VUs instead of all VUs is an option. Figure 8 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 across all the nodes is up to an order of magnitude faster than computing all on every VU, as shown in Figure 9(a) for a 256 node CM-5E. The time for computing 1331 matrices in parallel decreases on larger CM-5Es, as shown in Figure 9(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.

Figure 10: Exploiting symmetry in the direct evaluation in the near-field.

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 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.

3.4 Particle-particle Interactions in the Near-field

Since the direct evaluation in the near-field accounts for about half of the total arithmetic operations at optimal hierarchy depth, its efficiency is crucial to the overall performance of $O(N)$ methods.

It is both efficient and convenient to use 4-D particle arrays in the direct evaluation in the near-field. The particle-particle interactions can then 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 10. As box 0 traverses boxes 1 to 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. In data-parallel programming, while box 0 traverses boxes 1 to 4, boxes 5 to 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 by Applegate et al. [3].

In three dimensions, the boxes involved in box-box interactions of a target box can be ordered linearly and brought to the target box through 62 single-step CSHIFTS. Another way is to fetch non-local near-field boxes from other VUs using 4-D arrays aliased into local subgrids through array aliasing, much in the same way as in fetching non-local interactive-field boxes, as described in Section 3.3.1. Due to an optimization trading memory requirements for arithmetic efficiency described below, the memory requirements in the near-field interactions are high. For this reason we choose the first method since it requires less temporary storage. Moreover, the CSHIFTS account for only about 10-15% of the time for the direct evaluation.

3.5 Load-balancing issues in nonadaptive $O(N)$ Methods

Nonadaptive hierarchical methods use nonadaptive domain decomposition, and the hierarchy of recursively decomposed domains is balanced. Three sources of parallelism exist in traversing the hierarchy, namely, among all the boxes at the same level in parent-child and interactive-field interactions, among each box's interactive-field boxes in the far-field to local-field conversions, and among all boxes at all levels in the

far-field to local-field conversions. Our implementation only exploits parallelism among boxes at the same level. The limited number of subdomains near the root will result in underutilization of processing nodes for massively parallel systems. However, large-scale N -body simulations often involve millions of particles and therefore require millions of leaf-level boxes in the hierarchy, since a leaf-level box contains a small constant number of particles for the optimal hierarchy depth. For these simulations there is excess parallelism even for the largest MPPs for several levels close to the leaf-level. Moreover, since there are many fewer boxes at levels close to the root than at levels close to the leaf-level, the cost of computation close to the root is insignificant. Hence, improved node utilization at those levels will not affect the total execution time significantly. In addition, a program that traverses the hierarchy level by level and sequentializes interactive-field computations for one target domain and parallelizes the computations for different target domains results in much simpler data and control structures than one that exploits parallelism beyond 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 matters. 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 after sorting and reshaping the 1-D particle arrays into 4-D particle arrays, 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. In contrast, using 1-D arrays for particle-box interactions results in more balanced computations, since 1-D arrays will always be laid out across the nodes evenly, which also implies balanced memory usage. In general, an adaptive hierarchical method is needed in order to achieve good performance for nonuniform particle distributions.

4 Performance and Accuracy Results

4.1 Performance

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 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%. To our knowledge, both the number of particles and the running time are the best reported to date. Prior to this work, only short-range interactions were simulated for systems of up to 100 million particles [25]. Furthermore, the running time per particle update of our code for 100 million particle systems is within a factor 3 of that reported by using the best short-range MD code [25].

In considering the execution times it should be mentioned that our implementation uses the idea of *supernodes*. Zhao [36] made the observation that of the 875 boxes in the interactive-field, in many cases all eight siblings of a parent are included in the interactive-field. By converting the far-field of the parent box instead of the far-fields of all eight siblings the number of far-field to local-field conversions are reduced to 189 from 875. The supernode idea must be modified somewhat for Anderson’s method, but the same reduction in computational complexity can be achieved [17].

For gravitational and Coulombic fields division and square roots represent a significant fraction of the arithmetic time. In counting the floating-point rates of the method, we use CM-5E/VU normalized FLOP count for the two operations, i.e. each division is counted as five FLOPs and each square root eight FLOPs.

Breakdown	$K = 12$		$K = 72$	
	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 5: The breakdown of the communication and computation time for the potential evaluation of 100 million particles on a 256 node CM-5E.

The timings breakdown for the potential field calculation of 100 million particles on a 256 node CM-5E is shown in Table 5 for $K = 12$ and $K = 72$. The hierarchy depths are 8 and 7, respectively. The predicted optimal hierarchy depth based only on minimizing the total number of floating-point (FLOP) operations are 7.97 and 7.10. Thus, for $K = 12$, the FLOP counts for the hierarchy and for the direct evaluation are well balanced. In fact, they differ by about 10%. Furthermore, the FLOP rates for $K = 12$ 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 we include the time for forming the far-field potential for leaf-level boxes, the BLAS operations for the T_1 , T_2 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

Figure 11: Scalability on the CM-5s.

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

evaluation of the potential due to particles in the far-field, and finally the direct evaluation in the near-field.

Figure 11 shows that the speed of our code scales linearly with the number of nodes and number of particles. The timings are collected on CM-5s due to the unavailability of a variety of configurations of CM-5E systems. All cases use uniform particle distribution in a 3-D cubic domain, 12 integration points per sphere, and optimal hierarchy depths. It is clear from Figure 11 that for a fixed number of particles per node, the efficiency remains independent of the number of nodes. The slight fluctuation is mainly due to the fluctuation in the number of FLOPs per particle for the optimal hierarchy depth, as shown in Figure 12.

4.2 Accuracy

To verify the accuracy of the code, we implemented the all-pair $O(N^2)$ direct method on the CM-5E. The code for the direct method is also written in CMF and uses CSHIFTs to perform all-to-all communication. It exploits symmetry in the potential/force evaluation from Newton's third law. Both the code using the direct method and the code using Anderson's method perform floating operations in 64-bit precision, and the potential values computed by the direct method are used as true values. Our experiments assume that particles are uniformly distributed within a unit cube and have equal charge strength, with the total charge being one. The root-mean-square errors of the potential fields are measured.

Our simulations with systems of up to one million particles show that the accuracy of the code is almost independent of the number of particles and the hierarchy depth. Figures 13–15 plot the accuracies for using one-separation (1-Sep), two-separation with supernodes (2-Sep/super), and two-separation (2-Sep), respectively, varying hierarchy depth and the integration order D of sphere approximations. Note that for small number of particles, the accuracies of all three methods become worse as hierarchy depth increases. This is because as the hierarchy depth increases, there are more and more leaf-level boxes than the number of particles, and the particle distribution relative to the spheres during traversing the hierarchy become less balanced (less centered), and larger integration error occurs. Also note for 2-Sep/super in Figure 14, the accuracy for depth three are better than for depth four and depth five, since our code only

Figure 13: RMS error of potential, 1-Sep at depth 3,4,5

Figure 14: RMS error of potential, 2-Sep/super at depth 3,4,5

Figure 15: RMS error of potential, 2-Sep at depth 3,4,5

Figure 16: RMS error of potential, at optimal depth

starts to use supernodes at level four. Starting supernodes at level three increases the error by an order of magnitude but affects the performance of the code very little, since there are very few boxes and thus little computation at level three. It is for the same reason that in Figure 16 which plots the accuracy at optimal hierarchy depth, there is an ascent in the plots for two-separation with supernode, before which the optimal depth is three and supernodes are not used.

Figure 17 plots the RMS error of potential as a function of integration order D . It shows that using two-separation with supernodes together with optimal hierarchy depths, our code evaluates the potential with four to seven digits of accuracy as D varies from five to fourteen, independent of the number of particles once supernodes are actually used.

We have also conducted an empirical study on the cost-accuracy tradeoffs of $O(N)$ methods varying different parameters using our implementation of Anderson's method. The details could not be included here due to the page limitation and can be found in [18].

Figure 17: RMS error of potential, at optimal depth

5 Conclusions

We have presented optimization techniques for programming $O(N)$ N -body algorithms for MPPs and shown how the techniques can be expressed in data-parallel languages, such as Connection Machine Fortran and High Performance Fortran. 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 most performance critical language features are the `FORALL` statement, array sectioning, array aliasing, `CSHIFT`, `SPREAD` and array inquiry intrinsics. All these features, except array aliasing, are included in HPF. The purpose of array aliasing, i.e., separating local axes from processor axes, can be achieved in HPF through the use of extrinsic functions.

The effectiveness of our techniques is demonstrated on an implementation in Connection Machine Fortran of Anderson's hierarchical $O(N)$ N -body method. Performance results comparable to those achieved from parallel implementations of Barnes-Hut algorithm highly optimized using message-passing and assembly language programming shows that nonadaptive $O(N)$ methods can be implemented efficiently in data-parallel languages through carefully structuring the code, even with currently commercially available compilers. To demonstrate the generality of the optimization techniques, we are currently porting the CMF code into HPF.

For highly clustered particle distributions, an adaptive version of N -body methods is needed in order to retain $O(N)$ arithmetic complexity. We report on issues in an efficient data-parallel implementation of adaptive $O(N)$ methods in HPF in [19].

Acknowledgment

We would like to thank Christopher Anderson for providing us with the sequential program and for many helpful discussions. We also thank Thinking Machines Corp., 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] C. R. Anderson. An implementation of the fast multipole method without multipoles. *SIAM J. Sci. Stat. Comput.*, 13(4):923–947, July 1992.
- [2] A. Appel. An efficient program for many-body simulation. *SIAM J. Sci. Stat. Comput.*, 6(1):85–103, 1985.
- [3] J. H. Applegate, M. R. Douglas, Y. Gursel, P. Hunter, C. L. Seitz, and G. J. Sussman. A digital orrery. *IEEE Trans. on Computers*, C-34(9):822–831, September 1985.
- [4] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force calculation algorithm. *Nature*, 324:446–449, 1986.
- [5] J. A. Board Jr., Z. S. Hakura, W. D. Elliott, D. C. Gray, W. J. Blanke, and J. Leathrum Jr. Scalable implementations of multipole-accelerated algorithms for molecular dynamics. In *Proc. Scalable High Performance Computing Conference SHPCC94*. IEEE Computer Society Press, May 1994.
- [6] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM Journal of Scientific and Statistical Computing*, July 1988.
- [7] J. J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms: Model implementation and test programs. Technical Report Reprint No. 2, Argonne National Laboratories, Mathematics and Computer Science Division, August 1988.
- [8] J. J. Dongarra, J. D. Croz, S. Hammarling, and R. 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] K. Esselink. The order of Appel’s algorithm. *Information Processing Letter*, 41:141–147, 1992.
- [11] K. Esselink. A comparison of algorithms for long range interactions. Technical Report 500.90.200GRA204-IN73916 AMER.94.006, Koninklijke/Shell-Laboratorium, Amsterdam, August 1994.
- [12] L. Greengard and W. D. Gropp. A parallel version of the fast multipole method. In *Parallel Processing for Scientific Computing*, pages 213–222. SIAM, Philadelphia, 1989.
- [13] L. Greengard and V. Rokhlin. A fast algorithm for particle simulation. *Journal of Computational Physics*, 73:325–348, 1987.
- [14] L. F. Greengard. *The rapid evaluation of potential fields in particle systems*. MIT Press, 1988.
- [15] L. F. Greengard and V. Rokhlin. On the efficient implementation of the fast multipole algorithm. Technical Report YALEU/DCS/RR-602, Dept. of Computer Science, Yale Univ., February 1988.
- [16] High Performance Fortran Forum. High performance fortran; language specification, version 1.0. *Scientific Programming*, 2(1 - 2):1–170, 1993.
- [17] Y. Hu and S. L. Johnsson. A data parallel implementation of hierarchical N -body methods. Technical Report TR-26-94, Harvard University, Division of Applied Sciences, September 1994.
- [18] Y. Hu and S. L. Johnsson. On the accuracy of Anderson’s fast N -body algorithm. Technical report TR-06-96, Harvard University, Division of Applied Sciences, January 1996.
- [19] Y. Hu and S. L. Johnsson. A data-parallel adaptive $O(N)$ hierarchical N -body method. Technical report, Harvard University, Division of Applied Sciences, 1996.
- [20] S. L. Johnsson and C.-T. Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.
- [21] J. Katzenelson. Computational structure of the N -body problem. *SIAM J. Sci. Statist. Comput.*, 10(4):787–815, 1989.

- [22] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Algorithm 539: Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5(3):324 – 325, March 1988.
- [23] J. F. Leathrum. *The parallel fast multipole method in three dimensions*. PhD thesis, Duke University, 1992.
- [24] P. Liu and S. N. Bhatt. Experiences with parallel N-body simulation. In *6th Annual ACM Symposium on Parallel Algorithms and Architecture*, pages 122–131. ACM, 1994.
- [25] 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 Computer Society, Los Alamitos, 1993.
- [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] J. K. Salmon. *Parallel Hierarchical N-Body Methods*. PhD thesis, 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. Singh, J. Hennessey, and A. Gupta. Implications of hierarchical N-body methods for multiprocessor architectures. Technical Report CSL-TR-92-506, Stanford University, 1992.
- [30] J. Singh, C. Holt, J. Hennessey, and A. Gupta. A parallel adaptive fast multipole method. In *Supercomputing '93*, pages 54 – 65. IEEE Computer Society, Los Alamitos, 1993.
- [31] Thinking Machines Corp. *CMSSL for CM Fortran, Version 3.1*, 1993.
- [32] Thinking Machines Corp. *CM Fortran Libraries Reference Manual, Version 2.2*, 1994.
- [33] M. Warren and J. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. In *Supercomputing '92*, pages 570 – 576. IEEE Computer Society, Los Alamitos, 1992.
- [34] M. Warren and J. Salmon. A parallel hashed oct-tree N-body algorithm. In *Supercomputing '93*, pages 12 – 21. IEEE Computer Society, Los Alamitos, 1993.
- [35] M. S. Warren and J. K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87, 1995.
- [36] F. Zhao. An $O(N)$ algorithm for 3-dimensional N-body simulations. Technical Report AI-TR-995, AI Lab, MIT, 1987.
- [37] F. Zhao and S. L. Johnsson. The parallel multipole method on the Connection Machine. *SIAM Journal of Scientific and Statistical Computing*, 12(6):1420–1437, Nov. 1991.