An $O(N)$ Algorithm for Three-dimensional N-body Simulations

by

Feng Zhao

Submitted to the Department of Electrical Engineering and Computer Science on October 2, 1987 in partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering and Computer Science

Abstract

We develop an algorithm that computes the gravitational potentials and forces on N point-masses interacting in three-dimensional space. The algorithm, based on analytical techniques developed by Rokhlin and Greengard, runs in order N time. In contrast to other fast N-body methods such as tree codes, which only approximate the interaction potentials and forces, this method is exact—it computes the potentials and forces to within any prespecified tolerance up to machine precision. We present an implementation of the algorithm for a sequential machine. We numerically verify the algorithm, and compare its speed with that of an $O(N^2)$ direct force computation. We also describe a parallel version of the algorithm that runs on the Connection Machine in order $O(\log N)$ time. We compare experimental results with those of the sequential implementation and discuss how to minimize communication overhead on the parallel machine.

Keywords: N-body algorithm, particle simulation, Multipole expansion, 3-D tree, spatial refinement, parallel computing

Thesis Supervisor: Harold Abelson
Title: Associate Professor of Computer Science and Engineering

Thesis Supervisor: Gerald Jay Sussman
Title: Professor of Electrical Engineering

To my Father and Mother

Acknowledgments

I would like to thank the many people who have contributed to the preparation of this thesis.

Hal Abelson and Gerry Sussman, my thesis advisors, have provided encouragement and support throughout the course of this research. Gerry deserves thanks for getting me started on the problem and for pointing out what is obvious and what is not. Hal has offered excellent critique of my writing and has improved the clarity of many ideas in the thesis. I have benefited a great deal from inspiring discussions with Hal and Gerry which led to the development of this thesis.

Many people in the MAC research group at MIT have provided helpful comments and assistance. Professor Jacob Katzenelson from Technion—Israel Institute of Technology—read and commented on early versions of the thesis. Franklyn Turbak has been a major source of help, and has spent hours and hours helping me prepare the final draft of this thesis. Mark Sheldon has provided help with typesetting problems. Andy Berlin and Ken Yip have provided constructive criticism.

I thank Thinking Machines Corporation for providing research facilities for my experimental work. Special thanks are to Danny Hillis, Lennart Johnsson, Alan Ruttenberg, and Luis Ortiz.

I thank Vicki Ledet and Craig Withers for their constant belief in me, and Jane Poet for helping me spiritually maintain an active life.

This thesis would never have been completed without the moral support and love from my family.

This thesis describes research done at the Artificial Intelligence Laboratory and the Laboratory for Computer Science at the Massachusetts Institute of Technology, supported by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-86-K-0180.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Numerical simulations of N-body interactions are extremely important in astrophysics, plasma physics, fluid dynamics, and molecular dynamics. To accomplish such a simulation by directly evaluating the interaction potentials requires computation time on the order of $O(N^2)$, which is prohibitively expensive for large N. Consequently, there have been many efforts to develop algorithms whose complexity grows more slowly [Lec72]. These algorithms all make assumptions about the distribution of particles to approximate the interaction potentials, and thus are applicable only to certain classes of N-body interactions.

The best-known "fast" algorithm is the *tree-code* [App85], which uses a tree-like data structure to form hierarchical clusters of particles. The algorithm approximates the force exerted by a cluster on a distant particle as the force exerted by an equivalent mass located the cluster's center of mass. The overall computational complexity of this algorithm is believed to be $O(N \log N)$. However, the force approximations are effective when the particle distribution is relatively nonuniform, and there are no known *a priori* estimates of the accuracy of these approximations, although methods have been developed for use in practice [Por85] [BH86].

Rokhlin and Greengard [GR86] discovered an N-body algorithm with complexity $O(N)$. The algorithm separates the computation of the far-field potentials from that of the near-field potentials, and expands the far-field potentials into Taylor series. Significantly, given a required precision ϵ, one can determine in advance how many terms must be retained in the Taylor series expansions, and thus one can control the accuracy of the approximation.

Rokhlin and Greengard's approach relies heavily upon analytical techniques to develop the series expansions, to derive error estimates, and to shift series expansions between coordinate frames. For the N-body problem in two dimensions, this analysis is greatly simplified by modeling two-dimensional space as the complex plane and using the theory of functions of a complex variable.

This thesis applies Rokhlin and Greengard's idea to develop an $O(N)$ algorithm for the three-dimensional N-body problem. Although there is no direct analogue of complex analysis in three dimensions, we are able to exploit the harmonic nature of the gravitational potential in order to produce suitable series expansions. We implement the algorithm and experimentally verify its accuracy and its linear growth. We also implement a parallel version of the algorithm and present experimental results.

Chapter 2 of the thesis reviews Rokhlin and Greengard's algorithm and their use of complex variables for the two-dimensional problem and points out difficulties in applying the method in three dimensions.

Chapter 3 develops a three-dimensional analogue of Rokhlin and Greengard's method. The basic idea is to expand the potentials in terms of spherical harmonics. The analytical basis of the algorithm includes two theorems that derive the required expansions and give bounds on the error terms. There are also three lemmas that describe how to shift these expansions from one origin to another. Using these analytical results, we present a sequential algorithm in three-dimensions. Independently of this work, a similar extension of Rokhlin and Greengard's method has been carried out by Greengard [Gre87]. His technique differs from ours in that he uses expansions in polar coordinates and we work in Cartesian coordinates.

Chapter 4 concerns numerical verifications of the three-dimensional algorithm. We first describe a sequential implementation of the algorithm. We discuss our choice of a tree-shaped data structure and describe heuristic methods for increasing the efficiency of the implementation. We then present experimental results to demonstrate that the algorithm does have linear growth and does compute the forces and potentials to within any prespecified tolerance up to machine precision. We compare the speed of the algorithm with that of an $O(N^2)$ direct force computation. For a required average accuracy of 10^{-4} for potential fields, our algorithm will be faster than the direct force computation when there are more than 1,000 particles.

Chapter 5 presents an extremely fast implementation of the N-body

code—a parallel implementation of the algorithm, which runs in $O(\log N)$ time, implemented on the Connection Machine. We compare the experimental results with that of the sequential implementation and find that the parallel algorithm has a speed-up of 10 for $N = 1,000$ and a speed-up of 100 for $N = 10,000$. For a large parallel machine, interprocessor communication is often the bottleneck of the computation. By exploiting regularity and locality in communication patterns, by combining and delegating messages, we demonstrate how to minimize communication overhead due to message congestion.

Throughout this thesis, if not specified otherwise, N will always refer to the total number of particles in a simulation, and p will always refer to the highest degree of harmonics retained in an expansion.

Chapter 2

The N-body Algorithm in Two Dimensions

This chapter follows the presentation of the two-dimensional algorithm given by Rokhlin and Greengard in [GR86]. We review the major theorems and give a heuristic description of the algorithm. The key idea is to use complex analysis to produce a series expansion of the potentials.

2.1 The Multipole Expansion Method

In the two-dimensional N-body problem, we consider point charges and Coulombic forces. For a charge q located at the point $y = (y_1, y_2) \in R^2$, the potential at a point $x = (x_1, x_2)$ is $q \log \parallel x - y \parallel$. If we identify R^2 with the complex plane C^1 via $(x_1, x_2) \leftrightarrow x_1 + ix_2$, we can consider the potential to be the real part of the analytic function $\phi_y : C^1 \rightarrow C^1$

$$\phi_y(x) = q \log(x - y).$$

We can expand $\phi_y(x)$ in a Taylor series that converges for any x with $|x| > |y|$:

$$\phi_y(x) = q \log(x - y) = q \left[\log(x) + \log\left(1 - \frac{y}{x}\right) \right] = q \left[\log(x) - \sum_{k=1}^{\infty} \frac{1}{k} \left(\frac{y}{x}\right)^k \right]$$

If we retain only p terms of the expansion, the error is bounded:

$$\left|\phi_y(x) - q\log(x) + q\sum_{k=1}^{p}\frac{1}{k}\left(\frac{y}{x}\right)^k\right| \le \frac{q}{1-\left|\frac{y}{x}\right|}\left|\frac{y}{x}\right|^{p+1}.$$

This observation leads to the following *multipole expansion*: Given an ensemble of m charges $\{q^{(i)}, i = 1, ..., m\}$ located at points $\{y^{(i)}, i = 1, ..., m\}$, with $|y^{(i)}| \le r_0$, the potential $\phi_{y^{(i)}}(x)$ at any point $x \in C^1$ due to the charge $q^{(i)}$ at $y^{(i)}$, with $|x| > r_0$, is

$$\phi_{y^{(i)}}(x) = q^{(i)}\left[\log(x) - \sum_{k=1}^{\infty}\frac{1}{k}\frac{\left(y^{(i)}\right)^k}{x^k}\right], \qquad i = 1, ..., m. \tag{2.1}$$

The total potential at the point $x \in C^1$ due to all the charges, therefore, is

$$\phi(x) = \sum_{i=1}^{m}\phi_{y^{(i)}}(x) = Q\log(x) + \sum_{k=1}^{\infty}\frac{a_k}{x^k}, \tag{2.2}$$

where

$$Q = \sum_{i=1}^{m} q^{(i)} \qquad \text{and} \qquad a_k = \sum_{i=1}^{m}\frac{-q^{(i)}\left(y^{(i)}\right)^k}{k}.$$

Moreover, for any $p \ge 1$,

$$\left|\phi(x) - Q\log(x) - \sum_{k=1}^{p}\frac{a_k}{x^k}\right| \le \frac{A}{1-\left|\frac{r_0}{x}\right|}\left|\frac{r_0}{x}\right|^{p+1},$$

where

$$A = \sum_{i=1}^{m}|q^{(i)}|.$$

In particular, if $|r_0/x| < 1/2$, we have

$$\left|\phi(x) - Q\log(x) - \sum_{k=1}^{p}\frac{a_k}{x^k}\right| \le 2A\left|\frac{1}{2}\right|^{p+1}.$$

To obtain an approximation accurate to within a given precision ϵ, it suffices to retain only the first p terms in the multipole expansion (2.2) and to take p to be of the order $\lfloor -\log_2 \epsilon \rfloor$.

The ability to sum individual expansions of (2.1) to obtain the multipole expansion of (2.2) depends on the fact that these expansions $\{\phi_{y(i)}(x), i = 1, ..., m\}$ have a common reference point and a common region of convergence. If they don't, we need to shift the series $\{\phi_{y(i)}(x), i = 1, ..., m\}$ to a common reference point. Also, to insure that all the shifted series have a common region of convergence, we may need to "flip" a series so that its region of convergence lies inside a disk, rather than outside a disk. Rokhlin and Greengard [GR86] derive shifting and flipping lemmas that describe how to perform these operations in time independent of the total number of particles. These lemmas allow us to manipulate the multipole expansions in a manner required by the following fast algorithm.

2.2 Description of the Algorithm

Rokhlin and Greengard's $O(N)$ algorithm computes separately the potentials of close-by particles ("near-field potentials") and the potentials of far-away particles ("far-field potentials"). The near-field potentials are computed using direct evaluation. If the number of particles near any given particle is bounded by a small constant, that is, if the distribution of particles is relatively uniform, then the work required to compute the near-field potential at this particle is of order $O(1)$. The far-field potential is obtained by evaluating the p-term multipole expansion described above at the particle position, which takes constant number of operations for a prespecified p. The computation is organized so that the total amount of work for computing the potentials at all the N particles is of order $O(N)$.

More precisely, the two-dimensional space under consideration is regarded as a square (Figure 2.1). It is divided into four subsquares, and each of which is then recursively subdivided into four sub-subsquares, and so on. This decomposition is represented using a tree-like data structure in which each square is represented by a node. A square at level l of the tree has four child nodes that represent the subsquares at level $l + 1$. This recursive decomposition is continued until there are only several particles in a square at finest

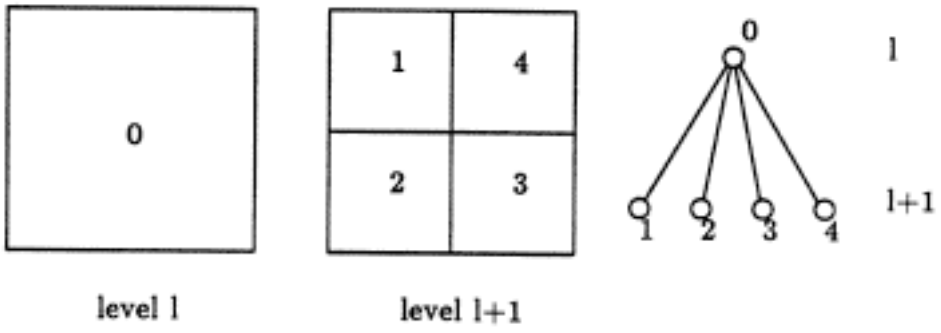


Figure 2.1: Decomposition in a two-dimensional space, with the corresponding data structure

grain. Under the assumption that the particles are uniformly distributed[1], the level of the tree n, i.e., the level of the decomposition, is usually chosen as about $\lfloor \log_4 N \rfloor$. Therefore each square at the finest level has in average one or two particle in it.

We define for a square its *near-field*, *far-field*, and *interactive-field*. The near-field of a square consists of those neighboring squares that are at the same level of decomposition as the square. In Figure 2.2, the squares labeled as B are in the near-field of square A. The far-field of a square is defined to be the exterior area of its near-field. The interactive-field of a square is the part of its far-field that is in the near-field of the square's parent. In Figure 2.2, the squares labeled as A' are in the interactive-field for the square A.

The goal of the computation is to compute the far-field potential expansions at all particle positions with $O(N)$ work. This is achieved by propagating values, first upwards through the tree, then downwards, as follows:

Initially, for each of the squares at the finest level, we compute the p-term multipole expansion, valid outside the square, for the potential due to the charges inside the square. The expansion is performed relative to the center of the square.

[1] Rokhlin [CGR87] has an $O(N)$ adaptive form of the algorithm which does not depend for its performance on the particle distributions.

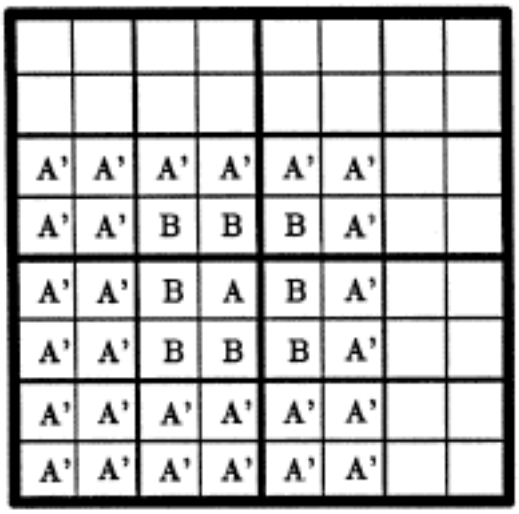


Figure 2.2: Near-field, far-field, and interactive-field

These expansions are propagated upwards through the tree to produce, for every square at every level, a multipole expansion for the potential due to all charges located inside the square. The valid region of convergence for the multipole expansion is the far-field of the square. To form the multipole expansion for a square at level l, we take each of its level-l+1 subsquares, shift their multipole expansions to the center of the level-l square and add them together. This is done recursively at each level of the tree while propagating upwards through the tree.

The downward-propagation phase of the computation produces, for every square, a local potential expansion due to its far-field interactions. The local expansion for a square is a multipole expansion that is valid inside the square. The computation proceeds recursively. Suppose we already have a local potential expansion $\phi'_{A_P}(x)$ for square A's parent A_P. The local expansion $\phi'_A(x)$ for the square A is, by the definition of an interactive-field, the sum of $\phi'_{A_P}(x)$ and those multipole expansions due to particles in A's interactive-field. However, $\phi'_{A_P}(x)$ is relative to the center of A_P. In order to combine it with other multipole expansions for the square A, we shift $\phi'_{A_P}(x)$ to the center of A. We have, from the upward-propagation phase of the computation, the multipole expansion $\phi_{A'}(x)$ for each square A' in A's

interactive-field. In order to combine $\phi_{A'}(x)$, we shift them to the center of A, and flip regions of convergence so that they have a common reference point and are valid inside the square A. The sum of the shifted $\phi'_{A_P}(x)$ and $\phi_{A'}(x)$ is the local expansion $\phi'_A(x)$ for the square A. This computation is performed recursively all the way down to the leaves of the tree.

Now we have for every square at the finest decomposition level the local expansion due to its far-field valid in the square. To get the far-field potential at a given particle, we have only to evaluate the local expansion at the particle's position. The near-field potential is obtained by evaluating directly interactions with particles in the near-field. The sum of the near-field potential and the far-field potential is the desired potential at the particle.

Rokhlin and Greengard [GR86] show that the total work required is $N(ap^2 + bp + ck_n + d)$, where k_n is the upper-bound for the number of particles in a square at the finest level. The key observation in this estimate is that each shifting or flipping of a p-term multipole expansion takes $O(p^2)$ work, and that the number of squares in a given square's interactive-field is bounded by 27. The total number of squares at all levels is

$$4^0 + 4^1 + \cdots + 4^n = \frac{4^{n+1} - 1}{3} = \frac{4N - 1}{3}.$$

Therefore the upward-propagation phase and the downward-propagation phase of the algorithm accounts for the p^2 term in the estimate. The initial expansion step and the final evaluation step in the algorithm is responsible for the p term, and the direct computation on near-field potentials is for the k_n term.

The overall computational complexity of the algorithm, from the above estimate, is of order $O(N)$.

Rokhlin and Greengard [GR86] also derive that the error estimate in the computation is $C(1/2)^{p+1}$. Given a precision requirement ϵ, the number of terms p retained in the expansion is set as $\lfloor -\log_2 \epsilon \rfloor$.

2.3 Remarks

The key to the two-dimensional $O(N)$ algorithm is that, for the potential $\phi(x)$ due to charges at $\{y^{(i)}, i = 1, ..., m\}$, we evaluate the potential at each of the points $\{x^{(j)}, j = 1, ..., n\}$ by applying the multipole expansion of $\phi(x)$, rather

than by directly evaluating and summing the individual potentials $\phi_{y^{(i)}}(x)$ at each $x^{(j)}$. Our ability to apply this method relies on the fact that the Taylor series expansions of the $\phi_{y^{(i)}}(x)\{i = 1, ..., m\}$ are absolutely convergent power series whose coefficients are independent of x. We produce the expansion of $\phi(x)$ by summing corresponding coefficients of the expansions $\phi_{y^{(i)}}(x)$. In order to do so, these expansions must have a common reference point and a common region of validity, or they can be shifted so that they do so.

The applicability of complex-analytic techniques in the two-dimensional case results from the fact that the two-dimensional potential, viewed as a real-valued function on R^2, is harmonic, and thus is the real part of a complex-analytic function on C. In three dimensions, the potential function is likewise harmonic. Unfortunately, there are no corresponding complex-analytic techniques in three dimensions to simplify the analysis.

In the next chapter, we will produce expansions of the three-dimensional potential that are suitable to support the multipole expansion method.

Chapter 3

A three-dimensional N-body algorithm

We consider now the three-dimensional N-body problem, where the gravitational potential at a point $x \in R^3$ due to a point mass μ located at $y \in R^3$ is

$$\phi_y(x) = -\frac{\mu}{\| x - y \|}. \tag{3.1}$$

Coulombic potential due to a point charge q has the same formula, except that μ is replaced by $-q$ [Gol59] [Arn78].

In order to apply the multipole expansion method, we must expand this potential as an absolutely convergent series whose coefficients are independent of x, and we must produce an *a priori* bound on the error when we retain only a finite number of terms in the series. We must also be able to shift the series expansion to a new origin.

The potential function in (3.1) is harmonic in everywhere other than y [Kos64]. This suggests the possibility of expanding $\phi_y(x)$ in terms of spherical harmonics [Hob55].

In this chapter we prove two theorems that derive the required expansions and give bounds on the error terms. There are also three lemmas that describe how to shift these expansions from one origin to another. Using these analytical results, we present an analogue of Rokhlin and Greengard's algorithm in three dimensions.

At the same time the author derived the results [Zha87], Greengard also

extended the two-dimensional algorithm to three dimensions [Gre87]. His method is similar to the one given here, but the underlying theorems and expansions are different. Greengard works in terms of polar coordinates, while the formulas and derivations here are done in terms of Cartesian coordinates.

In Chapter 4, we present experimental results that demonstrate the error bound and order-of-growth estimates given here. However, there are no comparable published experimental results for the polar-coordinate form of the algorithm. It is unknown how the two different forms of the algorithm compare in practical implementations.

3.1 The Multipole Expansion in Three Dimensions

Given an ensemble of particle masses $\{\mu^{(n)}, n = 1, ..., m\}$ located at points $\{y^{(n)} \in R^3, n = 1, ..., m\}$, the gravitational potential at any given point $x \in R^3$ is

$$\Phi(x) = \sum_{n=1}^{m} \frac{-\mu^{(n)}}{\| x - y^{(n)} \|} = \sum_{n=1}^{m} \frac{-\mu^{(n)}}{r^{(n)}}. \tag{3.2}$$

Here $r^{(n)} = \| x - y^{(n)} \|$. We will show how to expand $1/r^{(n)}$ into an absolutely convergent series

$$\frac{1}{r^{(n)}} = \sum_{ijk} a_{ijk}(y^{(n)}) \Theta_{ijk}(x), \qquad n = 1, ..., m, \tag{3.3}$$

where $a_{ijk}(y^{(n)})$ is independent of x and depends only on $y^{(n)}$. This will enable us to get the multipole expansion for the potential $\Phi(x)$ due to all the masses $\{\mu^{(n)}\}$ by summing together the $a_{ijk}(y^{(n)})$'s weighted by the masses $\mu^{(1)}, \mu^{(2)}, ..., \mu^{(m)}$ respectively:

$$\Phi(x) = \sum_{ijk} \left(\sum_{n=1}^{m} \left(-\mu^{(n)} \right) a_{ijk}(y^{(n)}) \right) \Theta_{ijk}(x). \tag{3.4}$$

The force field is obtained by taking the gradient of the potential field

$$\overline{F} = -\nabla \Phi(x)$$

$$= \sum_{ijk} \left(\sum_{n=1}^{m} \left(-\mu^{(n)} \right) a_{ijk}(y^{(n)}) \right) (-\nabla \Theta_{ijk}(x))$$

$$= \sum_{ijk} \left(\sum_{n=1}^{m} \left(-\mu^{(n)} \right) a_{ijk}(y^{(n)}) \right) \Theta'_{ijk}(x). \tag{3.5}$$

The series expansion of (3.5) has the same coefficients as that of (3.4), only this time, $\Theta_{ijk}(x)$ is replaced by $\Theta'_{ijk}(x)$. This enables us to use the same expansion formulas for potentials derived in the next section to compute multipole expansions for forces.

3.2 Theorems

This section derives necessary formulas for producing and shifting multipole expansions in three dimensions, together with bounds on the error when retaining only a finite number of terms in the expansions.

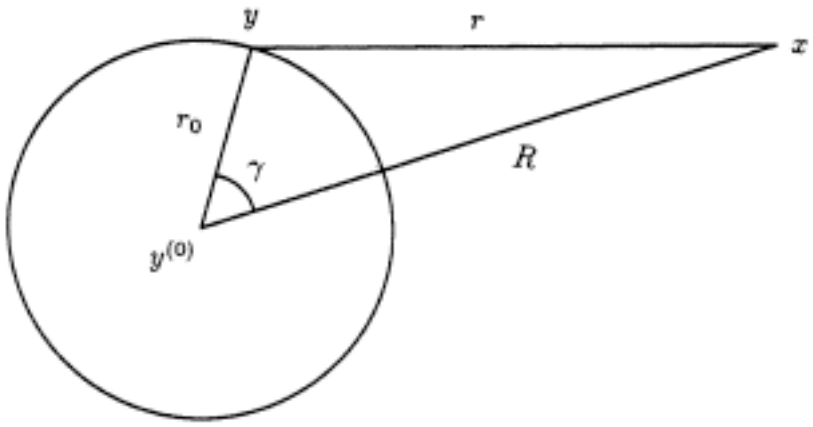


Figure 3.1: Gravitational field

Figure 3.1 illustrates the quantities needed in our derivation. The potential at point $x \in R^3$ is due to mass μ at point $y \in R^3$. $y^{(0)}$ is the reference point, and γ is the angle between vectors $\overrightarrow{y^{(0)}y}$ and $\overrightarrow{y^{(0)}x}$. Also there are

quantities $r = \| x - y \|$, $r_0 = \| y - y^{(0)} \|$, and $R = \| x - y^{(0)} \|$. We need the following relationships later in deriving the multipole expansion formula.

From the basic triangle relationship, we have

$$r = \sqrt{R^2 + r_0^2 - 2Rr_0 \cos\gamma}. \tag{3.6}$$

We derive the following theorem which produces an expansion for $1/r$ in a region of analyticity outside a sphere. This is essential in obtaining the multipole expansion (3.4).

Theorem 3.2.1 *Using the quantities as shown in Figure 3.1, if $R > r_0$, that is, if $x = (x_1, x_2, x_3)$ measured relative to $y^{(0)}$ is outside the sphere of radius r_0 centered at $y^{(0)}$, then*

$$\frac{1}{r} = \sum_{i,j,k=0}^{\infty} a_{ijk} \frac{\partial^{i+j+k}}{\partial x_1^i \partial x_2^j \partial x_3^k} \left(\frac{1}{R}\right) \tag{3.7}$$

where

$$a_{ijk} = (-1)^{i+j+k} \frac{r_0^{i+j+k}}{i!j!k!} \nu_1^i \nu_2^j \nu_3^k, \tag{3.8}$$

and ν_1, ν_2, ν_3 are the direction cosines of $\overline{y^{(0)}y}$.

If we retain only terms of $i + j + k \leq p$ then the error satisfies

$$|\epsilon| \leq C \left|\frac{r_0}{R}\right|^{p+1} \tag{3.9}$$

where C is independent of p.

Proof.

In Figure 3.1, the Cartesian coordinate distance between the points x and y is

$$r(x,y) = \| x - y \| = \sqrt{\sum_{\beta=1}^{3} \left[(x_\beta - y_\beta^{(0)}) - (y_\beta - y_\beta^{(0)})\right]^2}. \tag{3.10}$$

We expand $1/r(x,y)$ as a Taylor series in powers of $(y_\beta - y_\beta^{(0)})$ and get

$$\frac{1}{r(x,y)} = \frac{1}{r(x,y)}\bigg|_{y=y^{(0)}} + \sum_{\alpha=1}^{\infty} \frac{1}{\alpha!} \left\{ \left[\sum_{\beta=1}^{3} (y_\beta - y_\beta^{(0)}) \frac{\partial}{\partial y_\beta} \right]^\alpha \frac{1}{r(x,y)} \right\}_{y=y^{(0)}}. \tag{3.11}$$

The subscript $y = y^{(0)}$ following the brace indicates that after the differentiation is done y_1, y_2, y_3 of point y should be replaced by $y_1^{(0)}, y_2^{(0)}, y_3^{(0)}$ of point $y^{(0)}$. When x lies outside the sphere of radius r_0 centered at $y^{(0)}$, the above Taylor series converges absolutely and uniformly. we make the substitution $y = y^{(0)}$ before the differentiation is performed. Using the identities

$$\frac{\partial}{\partial y_\beta}\left(\frac{1}{r(x,y)}\right) = -\frac{\partial}{\partial x_\beta}\left(\frac{1}{r(x,y)}\right) \qquad \beta = 1,2,3$$

and

$$\frac{1}{r(x,y)}\bigg|_{y=y^{(0)}} = \frac{1}{R}$$

the Taylor series (3.11) becomes

$$\frac{1}{r(x,y)} = \frac{1}{R} + \sum_{\alpha=1}^{\infty} \frac{(-1)^\alpha}{\alpha!} r_0^\alpha \left(\sum_{\beta=1}^{3} \frac{y_\beta - y_\beta^{(0)}}{r_0} \frac{\partial}{\partial x_\beta} \right)^\alpha \left(\frac{1}{R}\right). \tag{3.12}$$

Let $\nu_\beta = \left(y_\beta - y_\beta^{(0)}\right)/r_0$. Since ν_1, ν_2, ν_3 are the direction cosines of $\overline{y^{(0)}y}$, it follows that

$$\left(\sum_{\beta=1}^{3} \nu_\beta \frac{\partial}{\partial x_\beta} \right)^\alpha = \sum_{i+j+k=\alpha} \frac{\alpha!}{i!j!k!} \nu_1^i \nu_2^j \nu_3^k \frac{\partial^\alpha}{\partial x_1^i \partial x_2^j \partial x_3^k}. \tag{3.13}$$

Substitute this into (3.12), we get

$$\frac{1}{r(x,y)} = \sum_{i,j,k=0}^{\infty} (-1)^{i+j+k} \frac{r_0^{i+j+k}}{i!j!k!} \nu_1^i \nu_2^j \nu_3^k \frac{\partial^{i+j+k}}{\partial x_1^i \partial x_2^j \partial x_3^k} \left(\frac{1}{R}\right). \tag{3.14}$$

This completes the proof for (3.7).

In order to derive the error bound of (3.9), we will use the fact that $1/r$ can be expanded in powers of R using the Legendre Polynomials [And85].

$$\frac{1}{r} = \begin{cases} \sum_{\alpha=0}^{\infty} \frac{r_0^\alpha}{R^{\alpha+1}} P_\alpha(\cos\gamma) & \text{if } \left|\frac{r_0}{R}\right| < 1 \\ \sum_{\alpha=0}^{\infty} \frac{R^\alpha}{r_0^{\alpha+1}} P_\alpha(\cos\gamma) & \text{if } \left|\frac{r_0}{R}\right| > 1 \end{cases} \tag{3.15}$$

The polynomials $P_\alpha(x)$ are called the Legendre Polynomials

$$P_0(x) = 1, \;\; P_1(x) = x, \;\; P_2(x) = \frac{1}{2}(3x^2 - 1), \;\; \cdots$$

The series of (3.15) is absolutely convergent. Notice that the Legendre Polynomials $P_\alpha(\cos\gamma)$ vary with the directions of the vectors $\overline{y^{(0)}x}$ and $\overline{y^{(0)}y}$, and thus is a function of the point x at which the potential is to be evaluated, as well as a function of the mass location y.

Now we can complete the proof of (3.9). Using the notation

$$\frac{\partial^\alpha}{\partial r_0^\alpha} = \left(\sum_{\beta=1}^{3} \nu_\beta \frac{\partial}{\partial x_\beta}\right)^\alpha$$

the series of (3.12) becomes

$$\frac{1}{r(x,y)} = \frac{1}{R} + \sum_{\alpha=1}^{\infty} \frac{(-1)^\alpha}{\alpha!} r_0^\alpha \frac{\partial^\alpha}{\partial r_0^\alpha}\left(\frac{1}{R}\right). \tag{3.16}$$

Notice that there is a term-by-term correspondence between this series and the series of (3.15). By equating terms, we have

$$P_\alpha(\cos\gamma) = (-1)^\alpha \frac{R^{\alpha+1}}{\alpha!} \frac{\partial^\alpha}{\partial r_0^\alpha}\left(\frac{1}{R}\right). \tag{3.17}$$

Surprisingly, $\frac{R^{\alpha+1}}{\alpha!}\frac{\partial^\alpha}{\partial r_0^\alpha}(\frac{1}{R})$ only depends on γ , not R.

Using the correspondence between (3.16) and (3.15), we have

$$|\epsilon| \;\; = \;\; \left|\frac{1}{r} - \sum_{i,j,k}^{i+j+k\le p} a_{ijk} \frac{\partial^{i+j+k}}{\partial x_1^i \partial x_2^j \partial x_3^k}\left(\frac{1}{R}\right)\right|$$

$$
\begin{aligned}
&= \left| \frac{1}{r} - \sum_{\alpha \le p} \frac{(-1)^\alpha}{\alpha!} r_0^\alpha \frac{\partial^\alpha}{\partial r_0^\alpha} \left(\frac{1}{R} \right) \right| \\
&= \left| \sum_{\alpha=p+1}^{\infty} \frac{(-1)^\alpha}{\alpha!} r_0^\alpha \frac{\partial^\alpha}{\partial r_0^\alpha} \left(\frac{1}{R} \right) \right| \\
&= \left| \sum_{\alpha=p+1}^{\infty} \frac{r_0^\alpha}{R^{\alpha+1}} P_\alpha(\cos\theta) \right|
\end{aligned}
$$

Since the Legendre polynomials have the property of $|P_\alpha(\cos\gamma)| \le 1$, the error therefore is

$$
\begin{aligned}
|\epsilon| &\le \sum_{\alpha=p+1}^{\infty} \left| \frac{r_0^\alpha}{R^{\alpha+1}} \right| = \frac{1}{R - r_0} \left| \frac{r_0}{R} \right|^{p+1} \\
&= C \left| \frac{r_0}{R} \right|^{p+1}.
\end{aligned}
$$

This completes the proof.

Using the results derived in Theorem 3.2.1, we can produce the multipole expansion of (3.4) in Section 3.1. As in Section 3.1, suppose that all m particle masses are located within a spherical region with radius r_0 centered at $y^{(0)}$, that is, $\| y^{(n)} - y^{(0)} \| < r_0$, $n = 1, ..., m$, we have from Theorem 3.2.1 the series expansion of $1/r^{(n)}$ converging for any x with $\| x - y^{(0)} \| > r_0$

$$
\frac{1}{r^{(n)}} = \sum_{i,j,k=0}^{\infty} a_{ijk} \frac{\partial^{i+j+k}}{\partial x_1^i \partial x_2^j \partial x_3^k} \left(\frac{1}{\| x - y^{(0)} \|} \right),
$$

and the error estimate

$$
\left| \frac{1}{r^{(n)}} - \sum_{i,j,k}^{i+j+k \le p} a_{ijk} \frac{\partial^{i+j+k}}{\partial x_1^i \partial x_2^j \partial x_3^k} \left(\frac{1}{\| x - y^{(0)} \|} \right) \right| \le C \left| \frac{r_0}{\| x - y^{(0)} \|} \right|^{p+1},
$$

$$
n = 1, ..., m.
$$

The multipole expansion of (3.4), therefore, is

$$
\Phi(x) = \sum_{ijk} \alpha_{ijk} \frac{\partial^{i+j+k}}{\partial x_1^i \partial x_2^j \partial x_3^k} \left(\frac{1}{\| x - y^{(0)} \|} \right), \tag{3.18}
$$

where

$$\alpha_{ijk} = \sum_{n=1}^{m} \left(-\mu^{(n)} a_{ijk} \right).$$

Furthermore,

$$\left| \Phi(x) - \sum_{i,j,k}^{i+j+k \le p} \alpha_{ijk} \frac{\partial^{i+j+k}}{\partial x_1^i \partial x_2^j \partial x_3^k} \left(\frac{1}{\| x - y^{(0)} \|} \right) \right| \le C' \left| \frac{r_0}{\| x - y^{(0)} \|} \right|^{p+1}, \tag{3.19}$$

where

$$C' = \sum_{n=1}^{m} C\mu^{(n)}.$$

The following theorem produces an expansion for $1/r$ in a spherical region of analyticity, which is the basis for flipping the region of analyticity in shiftings, as required by the fast algorithm.

Theorem 3.2.2 *Using the quantities in Figure 3.1 again, if $R < r_0$, that is, if $x = (x_1, x_2, x_3)$ measured relative to $y^{(0)}$ lies inside the sphere of radius r_0 centered at $y^{(0)}$, then*

$$\frac{1}{r} = \sum_{i,j,k=0}^{\infty} b_{ijk} x_1^i x_2^j x_3^k \tag{3.20}$$

where

$$\begin{aligned} b_{ijk} &= \frac{(-1)^{i+j+k}}{r_0^{i+j+k+1}} \sum_{\gamma=0}^{\lfloor \frac{k}{2} \rfloor} \sum_{\beta=0}^{\lfloor \frac{j}{2} \rfloor} \sum_{\alpha=0}^{\lfloor \frac{i}{2} \rfloor} AB \\ A &= \binom{-\frac{1}{2}}{i+j+k-\alpha-\beta-\gamma} \frac{(i+j+k-\alpha-\beta-\gamma)!}{(i-\alpha)!(j-\beta)!(k-\gamma)!} \\ B &= \binom{i-\alpha}{\alpha} \binom{j-\beta}{\beta} \binom{k-\gamma}{\gamma} (2\nu_1)^{i-2\alpha} (2\nu_2)^{j-2\beta} (2\nu_3)^{k-2\gamma} \end{aligned} \tag{3.21}$$

here ν_1, ν_2, ν_3 are the direction cosines of $\overline{y^{(0)}y}$.

The error bound for truncating terms of $i + j + k > p$ satisfies

$$|\epsilon| \le C \left| \frac{R}{r_0} \right|^{p+1} \tag{3.22}$$

Proof.

Using the relationship in (3.6), we have

$$\begin{aligned}\frac{1}{r} &= \frac{1}{\sqrt{R^2 + r_0^2 - 2Rr_0\cos\gamma}} \\ &= \frac{1}{r_0\sqrt{\left(\frac{R}{r_0}\right)^2 + 1 - 2\frac{R}{r_0}\cos\gamma}}.\end{aligned} \tag{3.23}$$

Denote by τ_1, τ_2, τ_3 the direction cosines of $\overline{y^{(0)}x}$, and ν_1, ν_2, ν_3 the direction cosines of $\overline{y^{(0)}y}$. Then

$$\cos\gamma = \tau_1\nu_1 + \tau_2\nu_2 + \tau_3\nu_3$$

where γ is the angle between $\overline{y^{(0)}x}$ and $\overline{y^{(0)}y}$. This leads to the equality

$$\left(\frac{R}{r_0}\right)^2 - 2\frac{R}{r_0}\cos\gamma = \frac{x_1^2 + x_2^2 + x_3^2}{r_0^2} - 2\frac{x_1\nu_1 + x_2\nu_2 + x_3\nu_3}{r_0}.$$

If we introduce the variables

$$t_1 = \frac{x_1}{r_0}, t_2 = \frac{x_2}{r_0}, t_3 = \frac{x_3}{r_0}, \text{ and}$$

$$\alpha = t_1(t_1 - 2\nu_1), \beta = t_2(t_2 - 2\nu_2), \gamma = t_3(t_3 - 2\nu_3), \tag{3.24}$$

equation (3.23) becomes

$$\frac{1}{r} = \frac{1}{r_0}\frac{1}{\sqrt{1 + (\alpha + \beta + \gamma)}}.$$

We can expand this as a Taylor series

$$\frac{1}{r} = \frac{1}{r_0}\sum_{n=0}^{\infty}\binom{-\frac{1}{2}}{n}(\alpha + \beta + \gamma)^n, \tag{3.25}$$

that is valid for $|\alpha + \beta + \gamma| < 1$. But

$$(\alpha + \beta + \gamma)^n = \sum_{i+j+k=n}\frac{n!}{i!j!k!}\alpha^i\beta^j\gamma^k,$$

thus

$$\begin{aligned}\frac{1}{r} &= \frac{1}{r_0}\sum_{i,j,k=0}^{\infty}\binom{-\frac{1}{2}}{i+j+k}\frac{(i+j+k)!}{i!j!k!}\alpha^i\beta^j\gamma^k \\ &= \frac{1}{r_0}\sum_{i,j,k=0}^{\infty} b'_{ijk}\alpha^i\beta^j\gamma^k \qquad (3.26)\end{aligned}$$

where

$$b'_{ijk} = \binom{-\frac{1}{2}}{i+j+k}\frac{(i+j+k)!}{i!j!k!}.$$

Substitute α, β, γ with (3.24), the Taylor series of (3.26) becomes

$$\frac{1}{r} = \frac{1}{r_0}\sum_{i,j,k=0}^{\infty} b''_{ijk}\left(\frac{x_1}{r_0}\right)^i\left(\frac{x_2}{r_0}\right)^j\left(\frac{x_3}{r_0}\right)^k \qquad (3.27)$$

where

$$\begin{aligned}b''_{ijk} &= (-1)^{i+j+k}\sum_{\gamma=0}^{\lfloor\frac{k}{2}\rfloor}\sum_{\beta=0}^{\lfloor\frac{j}{2}\rfloor}\sum_{\alpha=0}^{\lfloor\frac{i}{2}\rfloor} b'_{(i-\alpha),(j-\beta),(k-\gamma)} \\ &\quad \binom{i-\alpha}{\alpha}\binom{j-\beta}{\beta}\binom{k-\gamma}{\gamma}(2\nu_1)^{i-2\alpha}(2\nu_2)^{j-2\beta}(2\nu_3)^{k-2\gamma}.\end{aligned}$$

Write (3.27) explicitly in powers of x_1, x_2, and x_3

$$\frac{1}{r} = \sum_{i,j,k=0}^{\infty} b_{ijk}x_1^i x_2^j x_3^k$$

where

$$b_{ijk} = \frac{b''_{ijk}}{r_0^{i+j+k+1}}.$$

As in Theorem 3.2.1, the error bound for truncating terms of $i+j+k > p$ in (3.20) satisfies

$$|\epsilon| \le C\left|\frac{R}{r_0}\right|^{p+1}.$$

In the following lemma, we produce a formula which shifts the center of a multipole expansion valid in a region of analyticity outside a sphere.

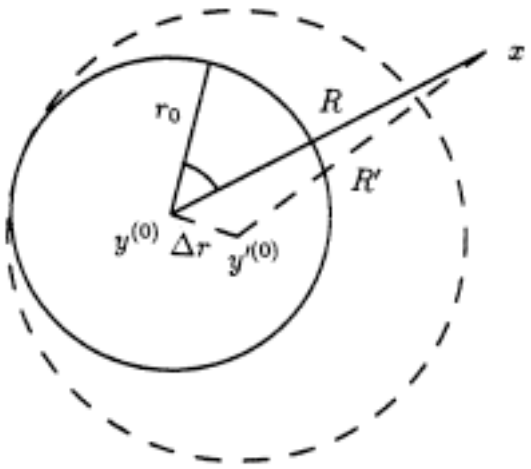


Figure 3.2: Lemma 3.2.1

Lemma 3.2.1 *Given a multipole expansion*

$$\Phi(x) = \sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} \frac{\partial^{\alpha+\beta+\gamma}}{\partial x_1^\alpha \partial x_2^\beta \partial x_3^\gamma} \left(\frac{1}{R}\right), \tag{3.28}$$

with respect to $y^{(0)}$, valid in the region outside the sphere of radius r_0 centered at $y^{(0)}$, suppose we shift the reference point $y^{(0)}$ to $y'^{(0)}$ with $\| y^{(0)} - y'^{(0)} \| = \Delta r$, as shown in Figure 3.2. Note that $R' = \| x - y'^{(0)} \|$. The new multipole expansion of (3.28), with respect to $y'^{(0)}$, in the region outside the sphere of radius $r_0 + \Delta r$ centered at $y'^{(0)}$ is

$$\Phi(X) = \sum_{i,j,k=0}^{\infty} c_{ijk} \frac{\partial^{i+j+k}}{\partial X_1^i \partial X_2^j \partial X_3^k} \left(\frac{1}{R'}\right) \tag{3.29}$$

where

$$c_{ijk} = \sum_{\alpha=0}^{i} \sum_{\beta=0}^{j} \sum_{\gamma=0}^{k} a_{\alpha\beta\gamma} a'_{i-\alpha,j-\beta,k-\gamma}. \tag{3.30}$$

Here $X = (X_1, X_2, X_3)$ are the Cartesian coordinates of the point x relative to $y'^{(0)}$, and a'_{ijk} are the coefficients in the expansion for $1/R$ with respect to $y'^{(0)}$.

The error bound for retaining terms of $i+j+k \le p$ in (3.29) satisfies

$$|\epsilon| \le C \left| \frac{r_0 + \Delta r}{R'} \right|^{p+1}. \tag{3.31}$$

Proof.

Using Theorem 3.2.1, we expand $1/R$, with respect to $y'^{(0)}$, in the region outside the sphere of radius Δr centered at $y'^{(0)}$ as

$$\frac{1}{R} = \sum_{i,j,k=0}^{\infty} a'_{ijk} \frac{\partial^{i+j+k}}{\partial X_1^i \partial X_2^j \partial X_3^k} \left(\frac{1}{R'} \right), \tag{3.32}$$

where

$$a'_{ijk} = (-1)^{i+j+k} \frac{\Delta r^{i+j+k}}{i!j!k!} \tau_1^i \tau_2^j \tau_3^k.$$

τ_1, τ_2, τ_3 are direction cosines of $\overline{y'^{(0)} y^{(0)}}$. Notice that after the shifting there is

$$X_i - x_i = y^{(0)} - y'^{(0)} \qquad i = 1, 2, 3$$

Therefore

$$\frac{\partial}{\partial x_i} = \frac{\partial}{\partial X_i}.$$

Substitute (3.32) into (3.28), we have

$$\Phi(x) = \sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} \frac{\partial^{\alpha+\beta+\gamma}}{\partial x_1^\alpha \partial x_2^\beta \partial x_3^\gamma} \left[\sum_{i,j,k=0}^{\infty} a'_{ijk} \frac{\partial^{i+j+k}}{\partial X_1^i \partial X_2^j \partial X_3^k} \left(\frac{1}{R'} \right) \right],$$

that is

$$\Phi(X) = \sum_{\alpha,\beta,\gamma=0}^{\infty} \sum_{i,j,k=0}^{\infty} a_{\alpha\beta\gamma} a'_{ijk} \frac{\partial^{\alpha+\beta+\gamma+i+j+k}}{\partial X_1^{\alpha+i} \partial X_2^{\beta+j} \partial X_3^{\gamma+k}} \left(\frac{1}{R'} \right),$$

which is valid outside the sphere of radius $r_0 + \Delta r$ centered at $y'^{(0)}$. Make substitutions of $\alpha + i = i', \beta + j = j', \gamma + k = k'$, we get

$$\Phi(X) = \sum_{i',j',k'=0}^{\infty} c_{i'j'k'} \frac{\partial^{i'+j'+k'}}{\partial X_1^{i'} \partial X_2^{j'} \partial X_3^{k'}} \left(\frac{1}{R'} \right)$$

where

$$c_{i'j'k'} = \sum_{\alpha=0}^{i'} \sum_{\beta=0}^{j'} \sum_{\gamma=0}^{k'} a_{\alpha\beta\gamma} a'_{i'-\alpha,j'-\beta,k'-\gamma}.$$

So far we have proved the expansion (3.29).

Since the multipole expansion of (3.29) is unique in terms of the derivatives of $1/R'$, we could estimate the error bound for truncation in (3.29) as if we expanded it directly with respect to $y'^{(0)}$. From (3.19) we know that is

$$|\epsilon| \leq C' \left| \frac{r_0 + \Delta r}{R'} \right|^{p+1}.$$

We now show how to convert a multipole expansion into a local expansion valid within a spherical region of analyticity, by shifting the reference point. The region of analyticity is flipped.

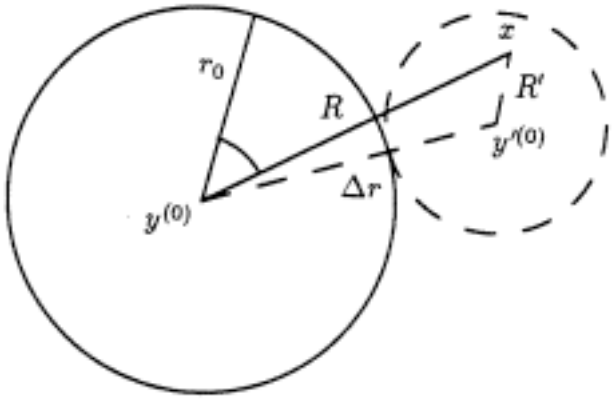


Figure 3.3: Lemma 3.2.2

Lemma 3.2.2 *Given a multipole expansion*

$$\Phi(x) = \sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} \frac{\partial^{\alpha+\beta+\gamma}}{\partial x_1^\alpha \partial x_2^\beta \partial x_3^\gamma} \left(\frac{1}{R}\right), \tag{3.33}$$

with respect to $y^{(0)}$, valid in the region outside the sphere of radius r_0 centered at $y^{(0)}$, suppose this time the reference point $y^{(0)}$ is shifted Δr to $y'^{(0)}$ with $\Delta r > r_0$, as shown in Figure 3.3. The new multipole expansion of (3.33), with respect to $y'^{(0)}$, this time inside a sphere of radius $(\Delta r - r_0)$ centered at $y'^{(0)}$ is

$$\Phi(X) = \sum_{ijk}^{\infty} c_{ijk} X_1^i X_2^j X_3^k \tag{3.34}$$

where

$$c_{ijk} = \frac{1}{i!j!k!} \sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} b_{i+\alpha,j+\beta,k+\gamma} (i+\alpha)!(j+\beta)!(k+\gamma)! \tag{3.35}$$

Again $X = (X_1, X_2, X_3)$ are the coordinates of the point x relative to $y'^{(0)}$. b_{ijk} are the coefficients in the expansions for $1/R$ with respect to $y'^{(0)}$.

The error bound for retaining terms of $i + j + k \leq p$ in (3.34) is

$$|\epsilon| \leq C \left| \frac{R'}{\Delta r - r_0} \right|^{p+1}. \tag{3.36}$$

Here $R' = \| x - y'^{(0)} \|$.

Proof.

From Theorem 3.2.2 we can expand $1/R$, with respect to $y'^{(0)}$, inside a spherical region with radius $(\Delta r - r_0)$ and center $y'^{(0)}$ as

$$\frac{1}{R} = \sum_{i,j,k=0}^{\infty} b_{ijk} X_1^i X_2^j X_3^k \tag{3.37}$$

where

$$\begin{aligned} b_{ijk} &= \frac{(-1)^{i+j+k}}{r_0^{i+j+k+1}} \sum_{\gamma=0}^{\lfloor \frac{k}{2} \rfloor} \sum_{\beta=0}^{\lfloor \frac{j}{2} \rfloor} \sum_{\alpha=0}^{\lfloor \frac{i}{2} \rfloor} AB \\ A &= \binom{-\frac{1}{2}}{i+j+k-\alpha-\beta-\gamma} \frac{(i+j+k-\alpha-\beta-\gamma)!}{(i-\alpha)!(j-\beta)!(k-\gamma)!} \\ B &= \binom{i-\alpha}{\alpha} \binom{j-\beta}{\beta} \binom{k-\gamma}{\gamma} (2\tau_1)^{i-2\alpha} (2\tau_2)^{j-2\beta} (2\tau_3)^{k-2\gamma} \end{aligned} \tag{3.38}$$

τ_1, τ_2, τ_3 are direction cosines of $\overline{y'^{(0)}y^{(0)}}$. Substitute (3.37) into (3.33)

$$\begin{aligned}\Phi(X) &= \sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} \frac{\partial^{\alpha+\beta+\gamma}}{\partial x_1^\alpha \partial x_2^\beta \partial x_3^\gamma} \left(\sum_{i,j,k=0}^{\infty} b_{ijk} X_1^i X_2^j X_3^k \right) \\ &= \sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} \sum_{i=\alpha}^{\infty} \sum_{j=\beta}^{\infty} \sum_{k=\gamma}^{\infty} b_{ijk} \\ &\quad \frac{i!j!k!}{(i-\alpha)!(j-\beta)!(k-\gamma)!} X_1^{i-\alpha} X_2^{j-\beta} X_3^{k-\gamma}. \end{aligned} \tag{3.39}$$

The region of analyticity is the inside of the sphere with radius $(\Delta r - r_0)$ and center at $y'^{(0)}$. After making substitutions of $i-\alpha = i', j-\beta = j', k-\gamma = k'$, we get

$$\Phi(X) = \sum_{i',j',k'=0}^{\infty} c_{i'j'k'} X_1^{i'} X_2^{j'} X_3^{k'} \tag{3.40}$$

where

$$c_{i'j'k'} = \frac{1}{i'!j'!k'!} \sum_{\alpha,\beta,\gamma=0}^{\infty} a_{\alpha\beta\gamma} b_{i'+\alpha,j'+\beta,k'+\gamma} (i'+\alpha)!(j'+\beta)!(k'+\gamma)!. \tag{3.41}$$

This completes the proof of (3.34).

Similar to Lemma 3.2.1, the error bound for retaining only terms of $i + j + k \leq p$ in (3.34) is

$$|\epsilon| \leq C' \left| \frac{R'}{\Delta r - r_0} \right|^{p+1}.$$

We also derive a formula for shifting an local expansion within a spherical region of analyticity.

Lemma 3.2.3 *Given a local expansion*

$$\Phi(x) = \sum_{i,j,k=0}^{\infty} c_{ijk} x_1^i x_2^j x_3^k,$$

with respect to $y^{(0)}$, valid in the region inside the sphere of radius r_0 centered at $y^{(0)}$, we expand $\Phi(x)$ inside the sphere of radius $r_0 - \Delta r$ centered at $y'^{(0)}$,

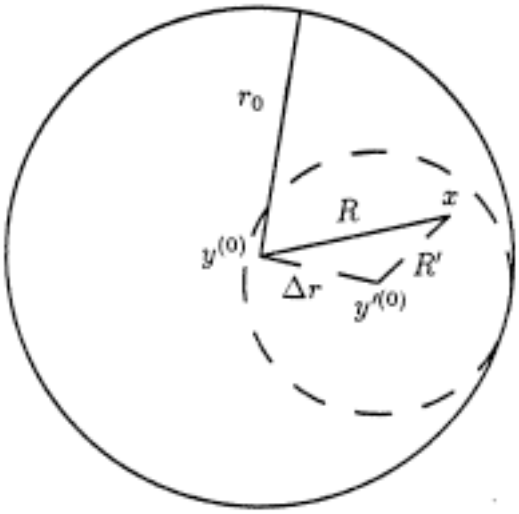


Figure 3.4: Lemma 3.2.3

after $y^{(0)}$ *is shifted* Δr *to* $y'^{(0)}$ *with* $\Delta r < r_0$ *as shown in Figure 3.4, as*

$$\Phi(X) = \sum_{i,j,k=0}^{\infty} d_{ijk} X_1^i X_2^j X_3^k$$

with $X = (X_1, X_2, X_3)$ *the new coordinates of the point* x *with respect to* $y'^{(0)}$*, and*

$$d_{ijk} = \sum_{\alpha,\beta,\gamma=0}^{\infty} c_{i+\alpha,j+\beta,k+\gamma} \binom{i+\alpha}{\alpha} \binom{j+\beta}{\beta} \binom{k+\gamma}{\gamma} \Delta x_1^\alpha \Delta x_2^\beta \Delta x_3^\gamma \qquad (3.42)$$

Here $y'^{(0)} - y^{(0)} = (\Delta x_1, \Delta x_2, \Delta x_3)$.

Proof.

This can be derived by re-expanding the expansion

$$\Phi(X) = \sum_{i,j,k=0}^{\infty} c_{ijk} (X_1 + \Delta x_1)^i (X_2 + \Delta x_2)^j (X_3 + \Delta x_3)^k$$

in powers of X_1, X_2, X_3. No truncation errors are introduced in the calculation of (3.42).

Having derived the potential in the form of the multipole expansion

$$\Phi(x) = \sum_{i,j,k=0}^{\infty} c_{ijk} x_1^i x_2^j x_3^k,$$

we recall from the equation (3.5) of Section 3.1 that the force is obtained by

$$\overline{F} = -\nabla\Phi(x),$$

where

$$\nabla = \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \frac{\partial}{\partial x_3}\right).$$

Therefore,

$$\overline{F} = \left(-\sum_{i,j,k=0}^{\infty} i c_{ijk} x_1^{i-1} x_2^j x_3^k, -\sum_{i,j,k=0}^{\infty} j c_{ijk} x_1^i x_2^{j-1} x_3^k, -\sum_{i,j,k=0}^{\infty} k c_{ijk} x_1^i x_2^j x_3^{k-1}\right).$$

3.3 A Sequential Algorithm

In this section we give a complete description of the three-dimensional N-body algorithm. This algorithm is similar to the two-dimensional algorithm of Rokhlin and Greengard described in Chapter 2, however this time, we use the analytical results obtained in the previous section. This formulation of the algorithm is for a sequential computer. We present a parallel version in Chapter 5.

The three-dimensional space under consideration is taken to be a cube (Figure 3.5). We apply the operation of subdividing a cube into eight identical subcubes repeatedly, until a subcube has only a few particles in it. When the particle distribution is relatively uniform, the level of subdivision is approximately $n = \lfloor \log_8 N \rfloor$. By convention, we say that the initial cube is at level 0, and the *atomic cubes*, i.e., cubes at the finest level of the spatial refinement, are at level n. As in the two-dimensional algorithm, we define for a cube its near-field, far-field and interactive-field. The near-field of a cube is defined as consisting of those cubes that are at the same level as the cube, and have distance to the center of the cube less than $\sqrt{3}$ of the cube edge length; the far-field of the cube is the complement of its near-field and itself; and the interactive-field of the cube is the part of its far-field contained in its parent's near-field.

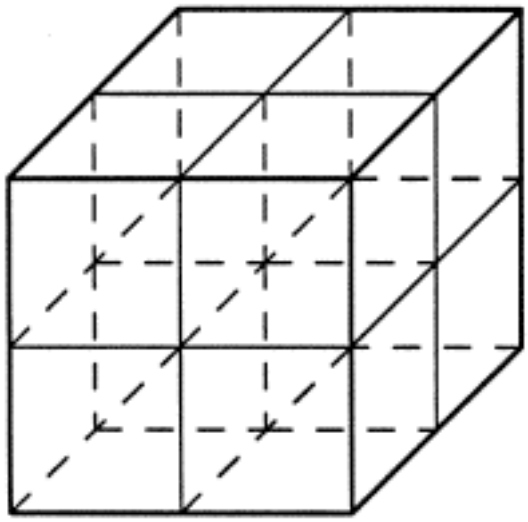


Figure 3.5: Decomposition in a three-dimensional space

For a cube i at level l, we denote by $\phi_i^l(x)$ the multipole expansion, valid in the far-field of the cube, for the potential due to particles in the cube; and by $\psi_i^l(x)$ the local expansion, valid inside the cube, for the potential due to those particles in its far-field. Both $\phi_i^l(x)$ and $\psi_i^l(x)$ are expanded relative to the center of the cube.

A description of the algorithm is given in Figure 3.6. n is the total levels of spatial refinement.

Step (1) computes initially the multipole expansion $\phi_i^n(x)$ for each atomic cube. This is obtained by adding together series expansions, relative to the center of the cube, for potentials due to individual particles in the cube.

Step (2) computes the multipole expansion $\phi(x)$ for each of the cubes at all intermediate levels in an upward manner. For a cube i at level l, the multipole expansion $\phi_i^l(x)$ is computed by summing $\phi^{l+1}(x)$ of cube i's eight subcubes that are shifted to the center of the cube i.

Step (3) computes the local expansion $\psi(x)$ for each of the cubes. For a cube i at level l, the computation consists of two parts. First it shifts $\psi^{l-1}(x)$ of cube i's parent to the center of cube i, which constitutes the far-field interaction. Then, it computes interaction with its interactive-field by summing $\phi(x)$ of the interactive-field that are shifted to the center of cube i.

The sum of these two parts is the local expansion $\psi_i^l(x)$ for the cube. This is recursively performed when walking down the refinement tree.

Finally step (4) computes the desired potential for each of the particles. The far-field potential is obtained by evaluating $\psi(x)$ of the atomic cube the particle belongs to at the particle position. The near-field potential is obtained by a direct computation.

The error estimate for the algorithm is given by (3.31) and (3.36). By the definition of the near-field and far-field for each of the cubes, we have $(r_0 + \Delta r)/R' < 1/2$ in (3.31) and $R'/(\Delta r - r_0) < 1/2$ in (3.36). For a required accuracy of ϵ, we choose $p = \lfloor -\log_2 \epsilon \rfloor$.

Let's analyze the computational complexity of the algorithm. The total number of the cubes at all levels of the subdivision is

$$8^0 + 8^1 + \cdots + 8^n = \frac{8^{n+1} - 1}{7} = \frac{8N - 1}{7}.$$

Step (1) takes $O(N)$ work to compute N series expansions for all the N particles. Summing these expansions to form $\phi(x)$ for every atomic cube takes another $O(N)$ work.

In step (2), it takes one shifting and seven summations of expansions to compute $\phi(x)$ for each of the cubes. This takes constant amount of work for a prespecified p. Since the total number of the cubes is of order N, the work to compute $\phi(x)$ for all the cubes in step (2) is of order $O(N)$.

In order to compute $\psi(x)$ for each of the cubes in step (3), we need to do one shifting on the parent $\psi(x)$ and one shifting for each of the cube in its near-field, and then sum the resulting expansions. Since the number of cubes in each cube's interactive-field is bounded by 567, it takes a constant amount of work to compute $\psi(x)$ for each of the cubes for a fixed p. Therefore step (3) takes $O(N)$ work in total to compute $\psi(x)$ for all the cubes.

We have chosen the level n of the subdivision so that the number of particles in the near-field of each atomic cube is bounded by a small constant. Thus in step (4), for each of particles the direct computation on the interaction with its near-field takes work of order $O(1)$. Evaluation of the local expansion at the particle position takes again constant amount of work for a fixed p. Thus, step (4) takes $O(N)$ work to compute potentials for all the N particles.

The overall complexity of the algorithm described in Figure 3.6, therefore, is the sum of those of step (1) through (4), that is, order $O(N)$.

(1) **Initial expansions:** for each atomic cube i, compute $\phi_i^n(x)$.

for i from 1 to 8^n **with step** 1 **do begin**
    compute for each of the particles in the cube i the series
    expansion for the potential due to the particle by using
    Theorem 3.2.1;
    sum these expansions for particles in the cube i to form
    $\phi_i^n(x)$ for the cube i.
end.

(2) **Upward-path:** for each of the cubes i at level l of the spatial refinement, compute $\phi_i^l(x)$ in an upward manner.

for l from $n-1$ to 0 **with step** -1 **do begin**
    for i from 1 to 8^l **with step** 1 **do begin**
        shift $\phi^{l+1}(x)$ of cube i's eight subcubes to the center of
        the cube i by using **Lemma 3.2.1**;
        Sum the shifted $\phi^{l+1}(x)$ of cube i's eight subcubes to form
        $\phi_i^l(x)$ for the cube i.
    end.
end.

(3) **Downward-path:** for each of the cubes i at level l, compute $\psi_i^l(x)$ in a downward manner.

for l from 1 to n **with step** 1 **do begin**
    for i from 1 to 8^l **with step** 1 **do begin**
        (3a) shift $\psi^{l-1}(x)$ of cube i's parent cube to the center
             of the cube i, by using **Lemma 3.2.3**;
        (3b) shift $\phi(x)$ of the interactive-field to the center of
             cube i, by using **Lemma 3.2.2**.
        sum the resulting expansions of (3a) and (3b) to form
        $\psi_i^l(x)$ for the cube i.
    end.
end.

Figure 3.6: A sequential algorithm

(4) **Final evaluation:** For each of particles, compute the potential at the particle.

```
for particle p from 1 to N with step 1 do begin
    (4a) evaluate ψ_i^E(x) of the atomic cube i the particle p belongs
         to at the particle position;
    (4b) compute directly interactions with particles in its near-
         field and in the cube i.
    add (4a) and (4b) as the desired potential for the particle p.
end.
```

Figure 3.7: A sequential algorithm (con't)

Chapter 4

Numerical Verifications of the Three-dimensional Algorithm

This chapter numerically verifies the three-dimensional algorithm of Chapter 3 with respect to the achieved accuracy and speed of the algorithm. We first describe a sequential implementation of the algorithm. We discuss our choice of a 3-D tree data structure, and describe heuristic methods for increasing the efficiency of the implementation based on a detailed analysis on the time complexity of the algorithm. We then present experimental results to demonstrate that the algorithm does have linear growth and does compute the forces and potentials to within any prespecified tolerance up to machine precision. We also compare the speed of the algorithm with that of a direct computation and find that for a required average accuracy of 10^{-4} for potential fields our implementation will be faster when there are more than 1,000 particles.

4.1 A Sequential Implementation

4.1.1 3-D Tree Data Structure

As described in the previous chapter, the algorithm requires the data structure used for the implementation to hierarchically decompose a three-dimensional space and to support operations such as insertion, deletion, and searching. Therefore a 3-D tree data structure is an obvious choice.

A 3-D tree [knu81] is a balanced eightfold-branching tree, in which each

level of nodes corresponds to one level of the three-dimensional spatial decomposition. The 3-D tree is an extension of the two dimensional tree data structure discussed in Section 2.2 of Chapter 2. In the 2-D tree, a node and its four children correspond to a square and its four subsquares; in the 3-D tree, a node and its eight children correspond to a cube and its eight subcubes. Note that the root of the 3-D tree corresponds to the original cube of space under consideration, while the leaves of the tree correspond to atomic cubes. Since each node of the tree corresponds to a cube in the spatial decomposition, we regard the word "node" and the word "cube" as interchangeable. From now on we will refer to a node as if we refer to the corresponding cube in three-dimensional space, and use the phrase "the center of a node" instead of "the center the cube". Each node of the tree has pointers to its children, and pointers to its near-field and its interactive-field, as defined before. The near-field and the interactive-field for each node could be computed at run time, but in practice this is too expensive. In a long-term simulation, which iterates over many time steps, these fields would have to be recomputed again and again. Therefore, we initially establish static pointers to each node's near-field and interactive-field. This approach saves time but requires extra storage space for these pointers.

Particles are initially contained in leaf nodes of the tree according to their positions. After each iteration of a simulation, information about particles, such as positions, velocities, etc. is updated. A particle is moved to a new node if it crosses boundaries of an atomic cube. Coefficients of expansions at various stages of the computation are stored in 3-D arrays held by each node. The hierarchical clustering of expansions is done by walking up and down the tree.

The level of spatial refinement is chosen approximately as $\log_8 N$. Thus, the 3-D tree is about $\log_8 N$ levels deep.

4.1.2 Implementation Details

In this section we discuss the implementation of the algorithm on a Symbolics LISP machine, and describe techniques for increasing the efficiency of the implementation.

Speed is the major concern in our implementation. One way to measure the efficiency of our implementation is to compare it with an implementation of the direct computation. For the first few implementations of our algorithm

on the LISP machine, the code ran extremely slowly. For $p = 3$, the crossover point where the running time of our method falls below the running time of the direct method was at about 10,000 particles.

Performance monitoring revealed opportunities for both machine-independent and machine-dependent optimizations.

We discuss the machine-independent optimizations first. Shifting expansions due to interactive-field potentials is very expensive. This accounts for most of the hidden constant in $O(N)$. For each node in the tree, we need to do 567 shiftings on expansions, since each node has 567 nodes in its interactive-field. We can reduce this cost by grouping together nodes in the interactive-fields. More specifically, we replace eight child nodes by their common parent node (we call it a *super-node*) if all eight nodes are in the interaction-field of a single node. Using this heuristic, each node has only 140 nodes in its interactive-field. Numerical experiments indicate that this modification speeds up the algorithm by a factor of about 8.

Another source of inefficiency in the initial implementation was the redundancy in computing coefficients in shifting formulas. The repeated calculation of factorials and permutations was removed by storing factorials and permutations in a table. There is also repeated calculation of some common factors in the formulas. We simplified this part of the computation by extracting common factors in sums, and by canceling common factors appearing in successive stages of the computation.

For the machine-dependent optimizations, we found that a major portion of the running time is spent on manipulating coefficients of expansions in shiftings and evaluations. The bottleneck of the computation is the floating-point calculation and indexing of arrays storing coefficients of various expansions. The array reference time was reduced by representing a 3-D array as a 2-D array of 1-D arrays, or as 1-D array of 2-D arrays. This minimizes the array reference overhead since 1-D and 2-D arrays are better supported on the LISP machine[1]. Because of the extensive use of array reference, the above improvement was significant. Other machine-dependent optimizations included using tight loops in frequently called procedures, and avoiding the creation of new arrays in manipulating expansions whenever possible.

The above optimizations greatly reduced both the time and storage requirements for the computation. The reduction in storage in turn reduces

[1]The idea is due to Rich Zippel.

the time due to disk paging. Overall, the machine-independent optimizations reduced running time by a factor of 8 and the machine-dependent optimizations provided an additional factor of 4. The running time crossover point with the direct computation is now at about 1,000 for $p = 3$ (See Table 4.4).

4.2 Experimental Verifications of the Algorithm

4.2.1 Accuracy of the Algorithm

In this section, we study the actual achieved accuracy of the algorithm, and compare it with the theoretical prediction given in Chapter 3. We first test our algorithm on the Pythagorean configuration of three bodies and observe how the error of the computation scales with p. Then we test the algorithm on two typical distribution models, the uniform distribution model and Plummer distribution model, each with 1,000 particles, and again determine how the error varies with p.

(a) The Pythagorean Configuration

The Pythagorean configuration of three bodies was first investigated by C. Burrau in 1913 [SP67]. It is Pythagorean not only in the geometric sense but also with respect to the masses. The sides of the triangle formed by the three bodies are 3, 4, and 5 and the masses of the three bodies are also 3, 4, and 5. The configuration is such that the body with mass x is situated at the vertex opposite to the side of length x, where x is one of 3, 4, or 5. The initial configuration of the problem and the coordinate system used are shown in Figure 4.1. Initially the three bodies are situated in the $z = 0$ plane and have speeds zero, consequently the motion is planar.

We use our algorithm to compute the accelerations of the three bodies and observe how error scales with p. The experimental results are given in Table 4.1 and plotted in Figure 4.2. We define the relative error in each of the accelerations as

$$\epsilon^{(i)}_{relative} = 2\frac{\left\|\vec{a}^{(i)}_{calculated} - \vec{a}^{(i)}_{true}\right\|}{\left\|\vec{a}^{(i)}_{calculated}\right\| + \left\|\vec{a}^{(i)}_{true}\right\|},$$

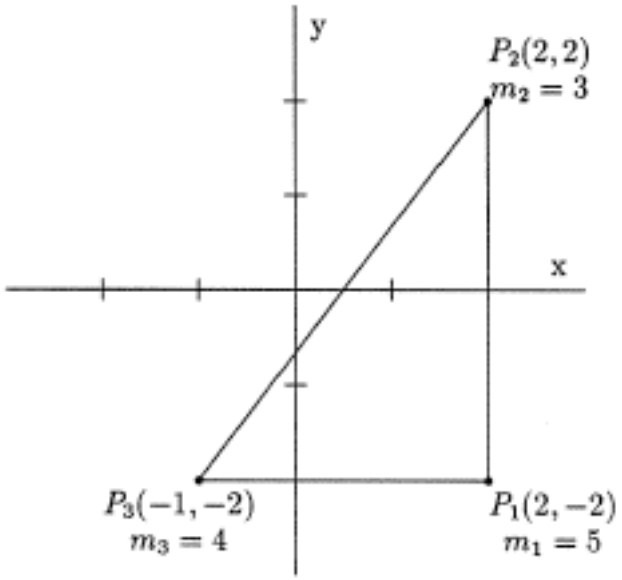


Figure 4.1: Initial configuration of the Pythagorean configuration of three bodies

where $\vec{a}^{(i)}_{calculated}$ is the calculated acceleration on body i using our method in single precision arithmetic, and $\vec{a}^{(i)}_{true}$ is the actual acceleration on that body using the direct method of force computation in double precision arithmetic and is therefore accurate to the machine round-off error of double precision arithmetic.

In Table 4.1, ϵ_{max} is the maximum error of all the relative errors in accelerations

$$\epsilon_{max} = Max\left\{\epsilon^{(1)}_{relative}, \ldots, \epsilon^{(n)}_{relative}\right\},$$

ϵ_{rms} is the root-mean-square error

$$\epsilon_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\epsilon^{(i)}_{relative}\right)^2},$$

where $n = 3$ is the number of bodies in the test, and $\epsilon_{potential-energy}$ is the relative error in total potential energy.

p	ϵ_{max}	ϵ_{rms}	$\epsilon_{potential-energy}$
1	0.58	0.46	0.0082
2	0.44	0.27	0.0020
3	0.17	0.11	4.2×10^{-4}
4	0.078	0.049	5.7×10^{-4}
5	0.040	0.025	2.7×10^{-4}
6	0.019	0.012	8.6×10^{-5}
7	0.0086	0.0055	1.3×10^{-5}
8	0.0039	0.0025	6.6×10^{-6}
9	0.0019	0.0012	6.8×10^{-6}
10	9.0×10^{-4}	5.7×10^{-4}	3.5×10^{-6}
11	4.3×10^{-4}	2.8×10^{-4}	1.1×10^{-6}
12	2.0×10^{-4}	1.3×10^{-4}	1.7×10^{-7}
13	9.3×10^{-5}	6.0×10^{-5}	1.5×10^{-7}
14	4.4×10^{-5}	2.9×10^{-5}	9.4×10^{-8}
15	2.1×10^{-5}	1.4×10^{-5}	5.5×10^{-8}
16	1.0×10^{-5}	6.6×10^{-6}	5.7×10^{-8}
17	5.0×10^{-6}	3.2×10^{-6}	5.5×10^{-8}
18	2.3×10^{-6}	1.5×10^{-6}	2.0×10^{-8}
19	1.1×10^{-6}	7.8×10^{-7}	1.7×10^{-8}
20	5.5×10^{-7}	4.0×10^{-7}	3.6×10^{-8}

Table 4.1: Accuracy test for the Pythagorean configuration of three bodies. p is the highest degree of harmonics retained in an expansion.

Table 4.1 shows that the accuracy of the algorithm is improved by approximately a factor of 2 when p is incremented by 1 for p greater than 2. Consequently, the dots in Figure 4.2 are distributed nearly on a line, since the error axis is scaled logarithmically.

Note that the near-field in this implementation has been defined so that the ratio appearing in the error bounds (3.31) and (3.36) of Chapter 3 is 1/2. The factor of 2 decrease in errors for each increase in p is thus expected from the formal analysis. The results presented in Table 4.1 and plotted in Figure 4.2 match well with the predicted error bounds.

The experimental results exhibit two phenomena that warrant further discussion. First, the error $\epsilon^{(i)}_{relative}$ in individual accelerations does not de-

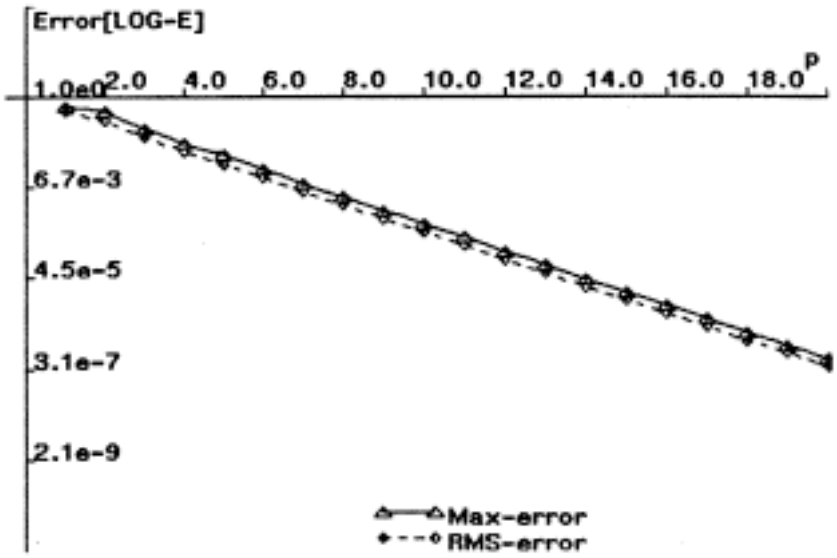


Figure 4.2: Plot of the accuracy test for the Pythagorean configuration of three bodies. p is the highest degree of harmonics retained in an expansion.

crease monotonically as p increases. Second, the calculated accelerations have nonzero components in the z-direction, which is perpendicular to the plane of the three-body configuration. Below, we discuss how these phenomena arise.

Nonmonotonic decrease of $\epsilon^{(i)}_{relative}$ with increasing p

The nonmonotonic decrease of $\epsilon^{(i)}_{relative}$ for individual accelerations with increasing p is due to the fact that a multipole expansion is a mixed-sign Taylor series. The error due to truncating a multipole expansion is also a mixed-sign series. Consequently, the error from truncating the multipole expansion does not necessarily decrease monotonically when retaining more terms in the multipole expansion. Nevertheless, the error in individual accelerations is always bounded by ϵ_{max} in Table 4.1, which, like the predicted error bounds, decreases monotonically as p increases.

Nonzero z component for accelerations

The nonzero z component in our computation results from two kinds of

truncation: the *primary truncation* and the *secondary truncation*. We note that each term of the multipole expansion in Lemma 3.2.2 of Chapter 3 is approximated by a convergent Taylor series, which we will call *term series*. In computing the multipole expansion using Lemma 3.2.2, the primary truncation truncates the multipole expansion and the secondary truncation truncates each term series .

In the multipole expansion computed from Lemma 3.2.2, the coefficients in each dimension are determined by parameters in all three dimensions. If an infinite number of terms were retained in computing both the term series and the multipole expansion, then the z component of the multipole expansion would be zero due to cancellation among nonzero z terms. Since only a finite number of terms are retained, the z component of the multipole expansion is not completely canceled.

Intuitively, we might expect that retaining more and more terms in the term series would reduce in general the secondary truncation error in computing the multipole expansion and in particular the error in z-direction. But the experimental results show that the intuition is unfounded. The reason is that the multipole expansion and the term series are mixed-sign series. The fact that each of the terms in the multipole expansion is better approximated by the term series does not necessarily guarantee that the resulting truncated multipole expansion is more accurate.

(b) Uniform Distribution of 1,000 Particles

We verify the accuracy claim of our algorithm using a configuration in which particles are distributed uniformly. In this test, the system has 1,000 particles, each of which has mass 1/1000. The results are shown in Table 4.2. The errors ϵ_{max}, ϵ_{max}, and $\epsilon_{potential-energy}$ are defined as before. The results match well with the prediction.

(c) Plummer Distribution of 1,000 Particles

We also test our algorithm to determine if the accuracy of the algorithm is sensitive to the distribution of particles. The Plummer distribution is a nonuniform distribution with the density profile [Her86]

$$\rho(r) = \frac{3M}{4\pi} \frac{r_0^2}{(r^2 + r_0^2)^{5/2}},$$

p	ϵ_{max}	ϵ_{rms}	$\epsilon_{potential-energy}$
1	0.56	0.064	5.0×10^{-4}
2	0.11	0.016	1.6×10^{-5}
3	0.095	0.0065	4.5×10^{-6}
4	0.024	0.0028	1.1×10^{-7}
5	0.016	0.0014	1.9×10^{-7}
6	0.0083	7.0×10^{-4}	2.9×10^{-7}
7	0.0026	3.3×10^{-4}	2.7×10^{-7}
8	0.0017	1.9×10^{-4}	1.3×10^{-7}
9	6.9×10^{-4}	9.7×10^{-5}	3.2×10^{-8}

Table 4.2: Accuracy test for the uniform model. p is the highest degree of harmonics retained in an expansion.

which is spherical and isotropic. M is the mass of the system and r_0 is the scale-length.

In the test the initial configuration of the system has two clusters, each of which is a Plummer system with 500 particles. Two clusters have the same M and r_0 and are separated by $5r_0$. A two-dimensional projection of this configuration is shown in Figure 4.3.

The experimental results are given in Table 4.3. Again, they match very well with the theoretical prediction.

In summary, the accuracy of the algorithm is demonstrated in Table 4.1, Table 4.2, and Table 4.3. These indicate that our method can compute forces and potentials to within any prespecified tolerance up to the machine round-off precision, which is about 7 decimal digits in single precision arithmetic [Ref85]. As we retain more and more terms in the multipole expansions, the accuracy of our method improves, and when $p = 20$ the error of the computation is at the level of the machine round-off errors.

4.2.2 Running Time of the Algorithm

In this section, we test the speed of our algorithm and experimentally determine how the running time grows with the number of particles. We also compare the results with those of a direct computation and determine the

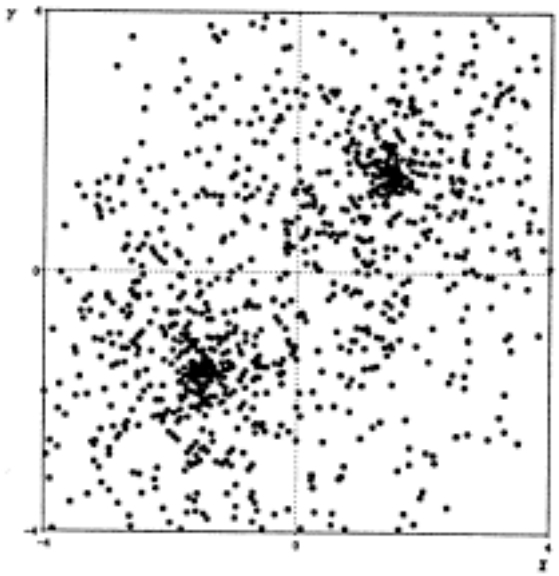


Figure 4.3: Initial configuration of two Plummer systems

p	ϵ_{max}	ϵ_{rms}	$\epsilon_{potential-energy}$
1	0.56	0.076	3.3×10^{-4}
2	0.13	0.019	1.8×10^{-5}
3	0.032	0.0053	7.7×10^{-6}
4	0.014	0.0018	1.2×10^{-6}
5	0.0066	7.3×10^{-4}	7.5×10^{-7}
6	0.0030	3.3×10^{-4}	2.6×10^{-7}
7	0.0013	1.6×10^{-4}	6.4×10^{-8}
8	5.9×10^{-4}	7.9×10^{-5}	2.1×10^{-8}
9	2.8×10^{-4}	4.0×10^{-5}	2.1×10^{-8}

Table 4.3: Accuracy test for the Plummer model. p is the highest degree of harmonics retained in an expansion.

Number of Particles	Running Time (sec.)		Speed-up
	Multipole-expansion	Direct	
64	12.60	2.30	0.18
128	22.73	9.08	0.40
256	51.38	39.04	0.76
512	409.95	156.34	0.38
1024	620.92	616.62	0.99
2048	1050.65	2468.72	2.3
4096	5221.29	9858.48	1.9
8192	7203.37	39433.92	5.5
16384	11411.68	157735.69	13.8

Table 4.4: Speed test for the Multipole-expansion algorithm with p=3. Results of a direct computation are also presented for the purpose of comparison. The running time excludes paging time.

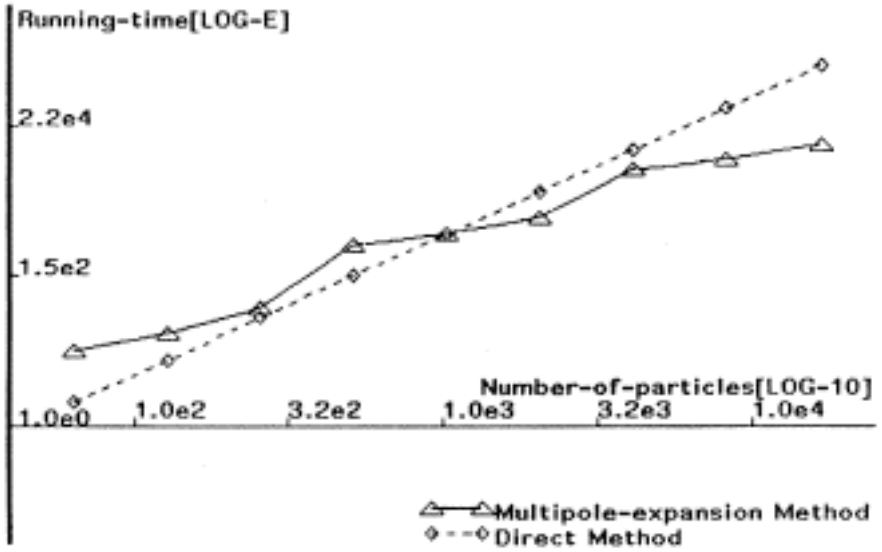


Figure 4.4: Running time growth rate of the Multipole-expansion algorithm, plotted against that of the direct computation.

running time crossover point. In the test, particles are initially distributed uniformly within a spherical region. The space under consideration is the cube of space containing the spherical region.

The results of the test on the speed of the algorithm are given in Table 4.4, and are plotted in Figure 4.4. The running time growth rate of the direct computation is also plotted for comparison. In the plot, both the running-time axis and the number-of-particles axis are scaled logarithmically. The parameter p of the algorithm is chosen as 3 and the calculated potentials are accurate in average to about 10^{-4}.

Figure 4.4 shows that the running time of our sequential implementation of the algorithm grows linearly with the number of particles. The jumps in the curve when the number of particles are 500 and 4,000 are due to the overhead of maintaining the tree structure when the tree grows one level deeper. The height of the tree grows logarithmically with the number of particles to enforce the constraint that the number of particles in each atomic cube is bounded by a predetermined constant.

The running time crossover point of our algorithm with the direct computation is at about 1,000 particles. This means that for a required average accuracy of 10^{-4} for the potentials, our sequential algorithm is faster than direct computation when there are more than 1,000 particles.

4.3 Discussion

We have numerically verified the three-dimensional algorithm with respect to the achieved accuracy and speed of the algorithm in the previous sections. In this section, we discuss some additional issues concerning the use of our algorithm.

(a) Tree Overhead

We note the overhead of maintaining the 3-D tree data structure in Section 4.2.2. The tree has a complicated pattern of links for the near-fields and the interactive-fields at each of the nodes. Much of the computation time is spent on tracing these pointers. When the tree is not fully loaded, this part of the overhead will dominate. This accounts for the jumps in the curve in Figure 4.4.

(b) Distribution of Particles

We have assumed in Chapter 3 that the distribution of particles is relatively uniform within the region under consideration. For very clumpy distributions, the performance of the algorithm will degenerate. If we still construct the decomposition tree with height of $\lfloor \log_8 N \rfloor$, step (4) of the algorithm would take more than $O(N)$ time to compute all the near-field interactions. Otherwise, to maintain the condition that the number of particles in each atomic cube is bounded by a predetermined constant, we would have to construct the tree with height of $O(N)$ in the worst case. Then step (2) and step (3) of the algorithm in Figure 3.7 of Chapter 3 would take time exponential in N to compute all the multipole expansions. One way to improve the efficiency is to dynamically prune empty tree branches, that is, if no particles are founded in a branch, the branch will be pruned from the tree.

(c) Choice of p

We have demonstrated in Section 4.2.1 that the accuracy of our method improves as we retain more and more terms in the multipole expansions. How big should the parameter p be in practice? This question depends on the specific nature of a problem. The goal of a numerical simulation is not always "accuracy" in a strictly mathematical sense, but rather "fidelity" to the underlying physics in a sense that is looser and more pragmatic [Pe86]. Often, as in galaxy simulations, the conserved system quantities such as total energy and total momentum are of more interest, so monitoring on these quantities will better reflect how good an algorithm is. Sometimes we only want to look at the statistical behavior of a system. In such context, some types of errors are much more tolerable than others. Another reason is that the error introduced by the discrete integration time step of a simulation is itself comparable with that of truncation [App85]. Therefore choosing small p often suffices. This will greatly reduce the constant factor in our algorithm.

Chapter 5

A Parallel Algorithm

5.1 Introduction

It is clear that the N-body method of Chapter 3 can take significant advantage of parallelism. With a massively parallel computer, where we can allocate a separate processing element for each particle, we can compute from expansions the potentials for all of the particles in one parallel step. We can also propagate values up and down the spatial-decomposition tree, as in steps (2) and (3) of the algorithm in Figure 3.6, in parallel, generating expansions for all the nodes at a given level in one parallel step. The depth of the tree grows as $\log N$, so a complete propagation will require time at least on the order of $\log N$.

In this chapter, we present an extremely fast implementation of the N-body code—a parallel implementation of the algorithm, whose measured running time does indeed grow as $O(\log N)$. The code runs on the Connection Machine, which is a massively parallel SIMD computer, consisting of up to 65,536 processors, each with its own local memory, connected in a communications network. On the Connection Machine model CM-2 that we have been using, each processor is a one-bit ALU with floating-point unit and 64K bits of local memory.

In designing a practical algorithm for a large parallel machine, there are two issues that must be addressed in addition to the issue of deciding which steps should be performed concurrently. For many large parallel computations, local computations at each processing element are relatively cheap, while communications among processing elements are expensive. The bot-

tleneck of the computation is the interprocessor communication. Thus, an efficient program must be designed to minimize both the need for interprocessor communication and the contention generated by whatever communication is required. For the parallel N-body code, we exploit the locality of the algorithm to reduce the need for communication. In addition we exploit the regularity in the communication patterns of the algorithm in order to substantially reduce contention for the underlying communication resource.

5.2 Why a Parallel Computer?

In the tree walking of our sequential algorithm, computations on expansions could be done concurrently within each level of the tree. Tremendous speed-up can be expected if we exploit this parallelism. The question is how to map our problem onto a parallel machine, either a machine specially designed for this problem or a commercially available one, such as the Connection Machine.

Let us take a close look at the sequential algorithm we developed in Chapter 3. First, the tree computation has regularity over all nodes at each level of the tree. A SIMD machine suffices for computation of this pattern. Second, the tree computation has concurrency. Computation at each of the nodes within a single level of the tree can proceed at the same time. Third, the tree computation has spatial locality. In the upward-path of tree walking, each of the nodes talks only to its parent node. In the downward-path of tree walking, each of the nodes talks not only vertically to its parent, but also laterally to those in its interactive-field. The lateral communication with a node's interactive-field is local to a subtree rooted the node's predecessor four levels up in the tree. Depending on how we construct the tree on a parallel machine, we might be able to exploit this locality. Lastly, the tree computation has dependencies among levels of the tree. More specifically, computation at one level of the tree cannot proceed unless the computation one level up or down is finished. This determines that the optimal performance we can achieve from the tree computation is proportional to the height of the tree, that is, $O(\log N)$, where N is the number of leaves in the tree.

We choose the Connection Machine to implement a parallel version of the algorithm developed in Chapter 3.

5.3 3-D Tree on the Parallel Computer

We use a 3-D tree as our data structure, as in the sequential implementation. The goal of the algorithm design is to distribute the components of the 3-D tree structure among many processors so that the concurrency in the tree computation can be maximumly exploited.

We allocate a processor to each node of the tree. The upward-path and downward-path of the tree walking of the algorithm require that each node in the tree has links to its parent as well as to its children. Explicitly storing these links would be expensive, since each node has one parent and eight children. However, since the tree has a fixed branching factor, a regular allocation of processors for the tree will impose an implicit relationship among the processor addresses, as long as the tree is allocated within a well-defined region of processors. This scheme is called the address-induced representation of the tree structure [Hil85] [Chr84].

More precisely, we allocate consecutive processors to nodes, starting at the root of the tree. We first make a sequential breadth-first scan of the tree to be constructed, and label each node as we encounter it, starting from 0 at the root. Then we assign to every node a processor (or a fixed number of processors) whose address is the breadth-first scan label of the node. We therefore have a formula to induce the parent-child relationship for every processor in the tree. Given the address i of a processor, we know that its parent processor has address of $\lfloor (i-1)/8 \rfloor$, and its child processors have addresses from $8i+1$ to $8i+8$. By representing the tree in this way, we avoid having to store pointers at every tree node. However, the saving in storage is achieved at the cost of computing the tree structure every time we trace pointers.

We also allocate a processor to each of the particles in a test [1]. Each leaf node of the tree remembers addresses of those particles that belong to the node spatially, so that those particles can be accessed simply by referring to the leaf node during the computation of its near-field. Similarly those particles also remember the leaf node for the initial expansion stage of the computation. This representation of particle-leaf-node links has an additional advantage. After every time step of the simulation, the tree may be updated for particles that move to new cells. This process is easy, since only particle-

[1] It is possible that the allocation of the tree overlaps with that of the particles.

leaf-node links need to be updated.

5.4 A Parallel Algorithm

In this section, we will describe a parallel implementation of the algorithm we developed in Chapter 3. The basic patterns of communication include passing and combining data among processors vertically and laterally with respect to the embedded tree structure. The description of the algorithm is summarized in Figure 5.1. Notations are used in the same way as that in the sequential algorithm of Figure 3.6. The number of levels in the tree is n.

In Step (1) of the algorithm, each node of the tree is mapped to a processor. Since the tree only needs to be built once, the sequential construction is not a serious drawback as long as we amortize its cost over many time frames in a simulation.

Step (2) finds for every node of the tree its near-field and interactive-field nodes. This could be done at run time, since the addresses for these nodes take a fair amount of memory spaces. But this means that we have to find these nodes every time we want to access them. We know that there are 81 nodes in a node's near-field, and 567 nodes (using heuristics this can be reduced to 140 nodes) in each interactive-field (we ignore the boundary conditions). Experiments have shown that the calculation of these fields is very expensive. So we precompute these fields and store them. Step (3) inserts all the particles into the tree.

Step (4) initially expands potentials for all particles and forms the expansions Φ for leaf nodes of the tree. An expansion for the potential of a particle is valid outside the leaf node the particle belongs to, and has the center of the node as the reference point. All expansions of a single leaf node are added together. This is done all in parallel. (Φ's and Ψ's are expansions as defined in Section 3.3, and are represented as arrays of parallel variables.)

Step (5) implements the upward path of the tree walking, which computes Φ for every node of the tree. The computation proceeds in this way: nodes at one level of the tree in parallel shift the Φ's to the centers of their respective parents, combine the shifted Φ's according to whether they have the same parents, and send the results to the parents.

Step (6) walks down the tree and computes Ψ's for every node of the tree. Every node in parallel fetches Ψ at its parent node, and shifts it to its center.

(1) **Tree embedding:** allocate one processor for every tree node, starting at the root.

```
allocate processor 0 for the root;
for i from 1 to n do begin
    for j from 1 to 8^i do begin
        allocate processor 8(8^(i-1) - 1)/7 + j for the jth to the
        leftmost node at level i of the tree.
    end.
end.
```

(2) **Find** for each of nodes its near-field and interactive-field nodes:

```
for all nodes of the tree in parallel do begin
    each node finds its near-field and interactive-field nodes
    and stores the addresses.
end.
```

(3) **Insert** N particles:

```
for i from 1 to N do begin
    insert particle i into the tree.
end.
```

(4) **Initial expansion** at leaf nodes:

```
for all particles in parallel do begin
    compute expansion Φ for the potential due to each of particles
    relative to center of the leaf node the particle belongs to;
    add together those Φ's due to particles of the same leaf node
    and form Φ for that leaf node.
end.
```

Figure 5.1: A parallel algorithm

(5) **Upward-path** of tree walking:

```
for i from n down to 1 do begin
    for all nodes at level i in parallel do begin
        at each node
           shift the Φ to the center of its parent node;
           sum the resulted expansions of those who have the
           same parent node and form Φ for that parent node.
    end.
end.
```

(6) **Downward-path** of tree walking:

```
for i from 1 to n do begin
    for all nodes at level i in parallel do begin
        at each node d1
           (6a) shift Ψ at d1's parent node to d1's center;
           (6b) for node d2 in d1's interactive-field do begin
                    shift Φ at d2 to d1's center;
                end.
                add the shifted Φ's together.
           add (6a) and (6b) to form Ψ for node d1.
    end.
end.
```

(7) **Final evaluation:** compute near-field and far-field interactions for each of particles.

```
for all particles in parallel do begin
    at each particle
       (7a) evaluate at the particle position the Ψ of the leaf
            node the particle belongs to;
       (7b) compute directly potentials due to its near-field.
       add (7a) and (7b) to form the desired potential.
end.
```

Figure 5.2: A parallel algorithm (con't)

This is done level by level, as in Step (5).

Finally, step (7) finishes the computation. Every particle evaluates in parallel the corresponding Ψ's at the particle position. This part of the potential is due to the interaction with its far-field. Then every particle computes in parallel the near-field interactions directly. The sum of the far-field and near-field interactions is the desired potential at this particle position.

We have seen from the above description that computation at a given level of the tree in steps (4)–(7) takes one parallel step. Thus, step (4) and step (7) each take constant time. Since the tree has depth of $\log N$, and computations at different levels of the tree have dependencies among them, a complete propagation in step (5) and step (6) takes $O(\log N)$ time. Therefore, the time complexity of the parallel algorithm, excluding the initial set-up time for the tree and communication costs, is $O(\log N)$.

Let us now look at the memory usage in the computation. When we retain in an expansion only the terms of spherical harmonics with degree less than four, the total number of terms in an expansion is 20. As verified by the experimental results from the sequential implementation, this guarantees in average a four decimal-digit accuracy in potentials. Each of the coefficients in an expansion is a 32-bit single precision floating-point number. We have three different arrays of coefficients for each node of the tree in the computation. Thus expansions alone take about 2K bits of memory at each of nodes. Storing the interactive-field and near-field takes another 4K bits of memory. We also need some stack space for intermediate computation. Therefore, about 10K bits of local memory at each node suffice for our purpose. In case a machine has insufficient amount of local memory per processor, we can remedy this by simulating a node of the tree with several processors. In such a case, however, the cost of the communication overhead would be high.

5.5 Experimental Results

We have implemented the parallel algorithm on the Connection Machine. The experimental results are summarized in Table 5.1. The running time growth rate of the implementation is plotted in Figure 5.3, along with that of the sequential implementation. The results experimentally verify that the parallel implementation of the algorithm on the Connection Machine scales

Number of Particles	Running Time (sec.)		Speed-up
	Lisp Machine	Connection Machine	
64	12.60	9.06	1.4
128	22.73	14.99	1.5
256	51.38	23.72	2.2
512	409.95	33.86	12
1024	620.92	51.78	12
2048	1050.65	60.65	17
4096	5221.29	71.83	73
8192	7203.37	79.63	90
16384	11411.68	94.17	121

Table 5.1: Experimental results on the Connection Machine. The running time excludes paging time.

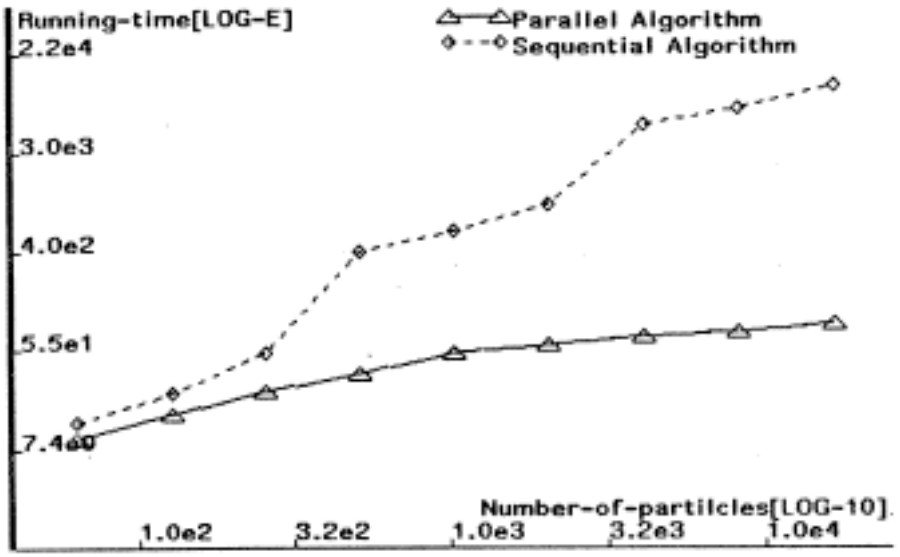


Figure 5.3: Running time growth rate of the parallel algorithm, with that of the sequential implementation.

logarithmically in the total number of particles in the simulation, even when we take the communication into the consideration. We find that the parallel algorithm runs faster than the sequential algorithm, even on small examples. Compared with the sequential implementation, the parallel algorithm exhibits a speed-up factor of about 10 for N=1,000 and a speed-up factor of about 100 for N=10,000[2].

The complexity issue of the parallel algorithm, which is complicated by the need for communication, will be discussed in more detail in the following section.

5.6 Communication Patterns

We have discussed the complexity issue due to the floating-point computation in Section 5.4. Our experiments have shown that the floating-point computation takes about 20% of the total running time and the interprocessor communication consumes a major portion of the rest[3].

A careful study at the communication patterns reveals that there are two major kinds of communication going on in the computation. The first kind of communication occurs between adjacent levels of the tree. We call this *vertical communication* . The second kind of communication occurs between nodes and their interactive-fields and near-fields, and takes place within a single level of the tree. We call this *lateral communication*[4]. The experimental results show that, when the tree is averagely loaded, the lateral communication constitutes about 20% of the total running time, and the vertical communication accounts for another 25%. In the following sections, we describe various optimizations we have made to reduce the communication overhead.

[2]The parallel computation has about 200 million floating-point calculations in a test of 1,000 particles, and has about 1,600 million floating-point calculations in a test of 10,000 particles.

[3]The results are obtained on the Connection Machine using the metering program provided by Shawn Mclean.

[4]Recall from Section 4.1.2 that a super-node is a node formed by grouping on interactive-field nodes. Communicating with a super-node in the lateral communication involves two adjacent levels of the tree. Although this situation is similar to the vertical communication, for the sake of simplicity we will consider it to be the lateral communication.

5.6.1 Reducing Communication Bottlenecks

The lateral communication required for computing interactions with near-fields and interactive-fields is one of the most expensive parts of the parallel computation. The cost of the lateral communication is due largely to *message collisions* in which many processors try to access a single processor at the same time. Since processor access on the Connection Machine is handled in an exclusive reading and exclusive writing way, the time of a successful message routing is proportional to the maximum number of message collisions.

We tried to ease the communication bottleneck by reducing the number of collisions in the lateral communication. We know that each node has 140 nodes in its interactive-field, and those nodes are represented as a list of addresses. A close look shows that many processors often have the same node as their interactive-field node at the same time, due to the way the interactive-field list is constructed. Our algorithm, therefore, randomizes the order in which one's interactive-field nodes nodes are accessed, in the hope that two processors are less likely to access a single processor at the same time. As a result, the randomization improves the performance of the algorithm.

5.6.2 Localizing Communication

In computing the interaction with a node's interactive-field, we need to use coefficients of expansions stored in an interactive-field node again and again. Instead of fetching these every time we use them, we fetch them once and store them on a temporary local stack. This can reduce the traffic by a factor of 5, when computing expansions whose highest degree is three.

5.6.3 Combining and Delegating Messages

The Connection Machine router has fairly good performance when the network is averagely loaded, but the routing time scales up as the number of collisions in the routing process. Collisions are mostly due to concurrent read or concurrent write to a single processor. To reduce the collisions, we can use the *scan* operation, which does the segmented parallel prefix computation on

the Connection Machine[5]. To resolve collision due to the concurrent read, we first do an exclusive read, and then do a *scan-copy* operation. In case of the concurrent write combined with addition, we do a *scan*-+ and then an exclusive write instead. If elements of a segment to be scanned are not in consecutive processors, in order to apply the *scan* operator we have to first project the elements to a new segment so that the new segment has elements in consecutive processors. This extra projection pays off when there are more than thirty processors in a segment.

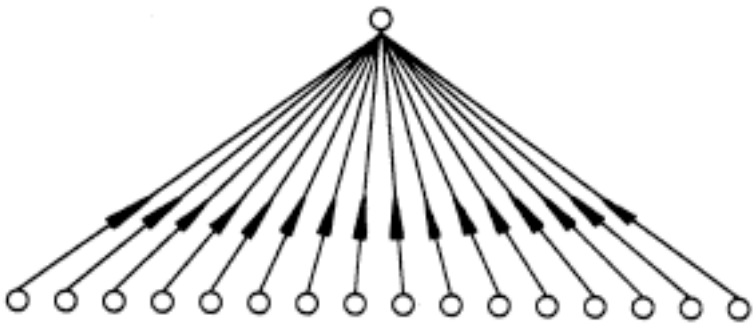


Figure 5.4: Concurrent write

In our implementation of the algorithm, the final direct interactions with a node's near-field often gives rise to 10–20 collisions. Therefore we use a different method to reduce the collisions. The idea is to structure the message routing through a tree that distributes the message collisions over many intermediate nodes. This tree-like routing allows many of the collisions to be handled in parallel, thereby reducing the total time delay due to the collisions.

Consider an example in which 16 processors access a single processor for concurrent write (Figure 5.4). Using the tree routing scheme outlined above,

[5]Given a binary associative operator $*$ and a segment of sequence of elements $x_1, x_2, ..., x_m$, the parallel prefix operation computes x_1, $x_1 * x_2$, ..., $x_1 * x_2 * \cdots x_m$ in $O(\log(m))$ time [HJ86]. In particular, there are *scan*-+ and *scan-copy* operations. The *scan* operation is very efficiently implemented on the Connection Machine. For simple binary operations such as + and *copy*, it takes the same order of time as a routing cycle does, which is the fundamental unit for time measurement, and is therefore considered to take constant amount of time.

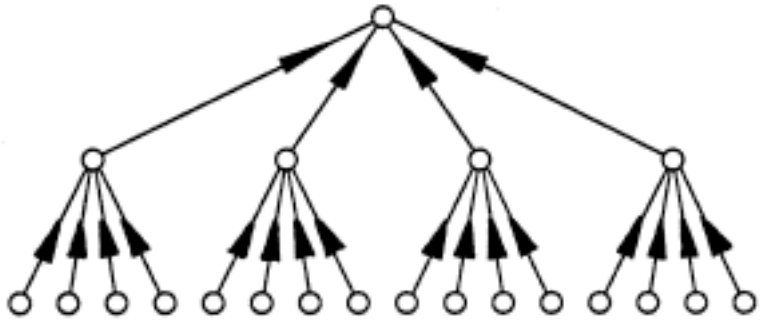


Figure 5.5: Combining messages

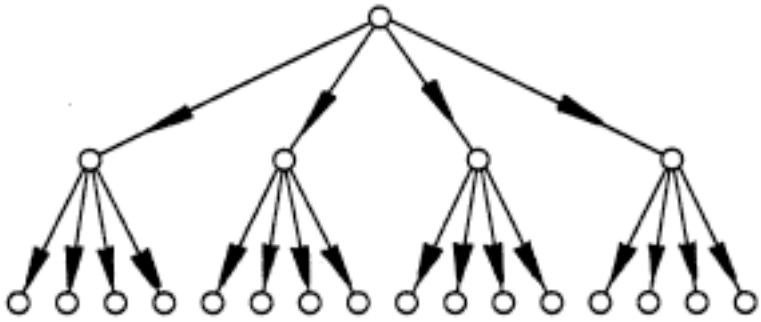


Figure 5.6: Delegating messages

we can use a two-level tree to combine every four of the writes together, and then combine every four of the resulting writes (Figure 5.5). Even though there are 20 total collisions in this tree, the time delay is only proportional to 8 collisions because the nodes at the intermediate level of the tree handle their collisions in parallel. This process is called *message-combining*. The dual process, *message-delegating*, reduces collisions due to concurrent read, as illustrated in Figure 5.6.

In general, suppose there are N collisions in total. Each stage of combining or delegating reduces the number of messages by a factor of m. Therefore the number of stages is $\log_m N$. What is the optimal value of m such that the function

$$\textsf{Routing-time}(m, N) = m \log_m(N)$$

is minimal? Using elementary calculus, we find that the optimal $m = e$, where e is the base of the natural logarithm 2.718... Of course, we must actually choose an integral value for m (2 or 3).

Message-combining and message-delegating have been very effective, and in practice have given a factor of 2 improvement in speed.

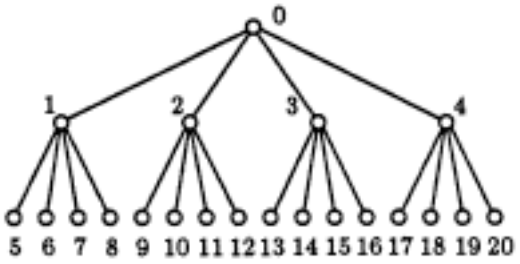
5.7 Discussion

In this section, we discuss trade-offs we made in our parallel implementation, and suggest alternatives to this implementation.

5.7.1 Grid Representation

We have used an address-induced scheme to construct the tree on the Connection Machine. This mapping scheme has the merit of simplicity. It also saves memory space, by avoiding explicitly storing pointers in the tree. As a result of this consecutive mapping scheme, only a subset of processors are active at a time, since the computation proceeds level by level in the tree.

The alternative to the address-induced scheme is to use the Connection Machine grid-like communication network called the NEWS grid. This structure provides fast communication for local or highly structured communication patterns [Hil85]. Every processor is assigned a grid coordinate and can be addressed by specifying this coordinate. The tree can be embedded in the



0,1,5	8	2,9	12
6	7	10	11
3,13	16	4,17	20
14	15	18	19

Figure 5.7: Superimposed mapping. Each square in the grid represents a processor, and each number represents a node in the tree.

1	4		
2	3	0	
5	8	9	12
6	7	10	11
13	16	17	20
14	15	18	19

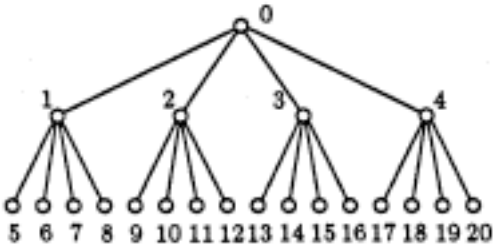


Figure 5.8: Non-superimposed mapping. Each square in the grid represents a processor, and each number represents a node in the tree.

network in two different ways. One way is to superimpose levels of the tree so that nodes at one level may share processors with those at other levels, as illustrated in Figure 5.7. As a result, some processors require more memory than others do. This scheme makes the processor utilization at most 100%. However, since all processors on the Connection Machine must have the same amount of local memory, the nonuniform memory requirement across processors wastes memory in some processors. The alternative is to unfold levels of the tree and to assign to every node a processor, as in Figure 5.8. This makes the memory utilization at most 100% but wastes some processors.

By using the NEWS grid, we can take advantage of the fast grid routing for highly structured communication patterns, such as the communication with interactive-fields discussed below.

5.7.2 Exploiting Regularities in Communication Patterns

To further reduce the number of collisions discussed in Section 5.6.1, Alan Ruttenberg has suggested a scheme to exploit the regularities in the communication patterns. We have not implemented this, so we do not know how much it decreases the running time of the algorithm. To keep the discussion simple, we present the scheme in two dimensions. The extension to three dimensions is straightforward.

We classify squares of the spatial decomposition into four classes. Suppose all the squares of smallest size of Figure 5.9 are at level l. We first group every four squares that share a common parent at level $l-1$. Then we classify the four squares of a group as class 1, 2 ,3, and 4 respectively according to the relative position of a square in the group, as labeled in the Figure 5.9.

We find that for every group there is spatial symmetry among the interactive-fields of its four squares. In Figure 5.10, square a' is an interactive-field square of square a. A clockwise rotation of 90 degrees of the square a' with respect to the point O results in another square b', which is an interactive-field square of square b, and so on. As a result of the spatial symmetry, the four squares a', b', c', and d' are in different classes, as shown in Figure 5.10. This symmetry property can be used to completely eliminate message collisions in the communication with interactive-field squares. The idea is as follows: each of four squares a, b, c, and d interacts with squares a', b', c', and d' respectively at the same time so that the communication

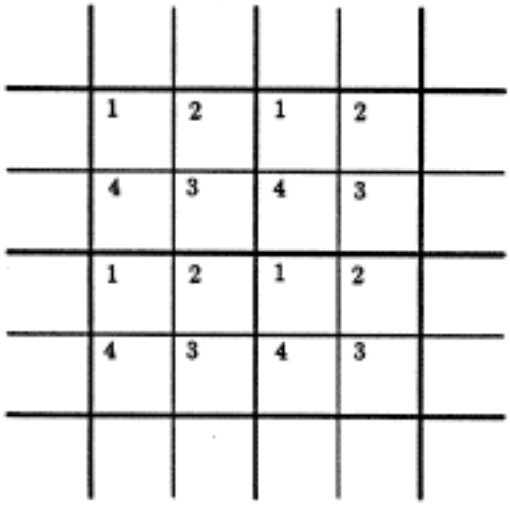


Figure 5.9: Classification of squares

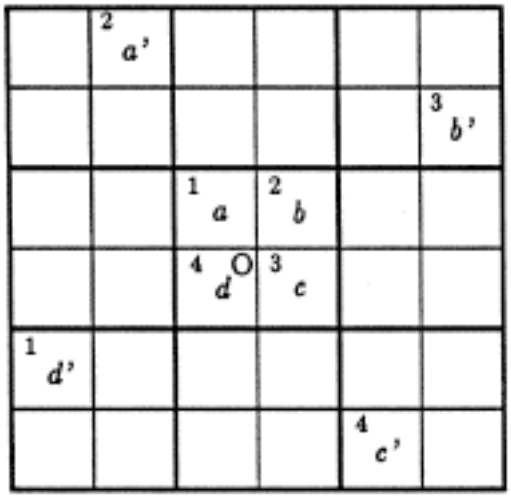


Figure 5.10: Symmetric communication pattern

pattern is symmetric with respect to four squares; the same is true for all other groups. Since the four squares a', b', c', and d' are all in different classes and communication patterns of two different groups are identical except that they are shifted from each other by some distance, none of the four squares a', b', c', and d' will be fetched by more than one square at a time. Thus, lateral communication can take place without collisions.

In Section 4.1.2, we discussed super nodes in the interactive-fields. Since the super nodes are one level higher in the decomposition tree than those ordinary nodes in the interactive-field, interactions with a super node using the above scheme will give rise to collisions. However, the maximum number of interactions to a super node will not exceed 4 at a time. Therefore the collisions can be avoided by spreading the data to be fetched at a super node to its four children, and afterwards, access to the super node will be directed to the four children.

In the above discussion we also ignored the boundary case. We can introduce dummy squares outside the boundaries so that boundary squares still preserve the symmetry property of interactive-fields.

5.8 Conclusion

We have described the parallel implementation of the three-dimensional N-body algorithm that runs on the Connection Machine in order $O(\log N)$ time in this chapter. Compared with the previous N-body methods, our method has a demonstrated advantage in simulating large number of particles. The parallel implementation achieves tremendous speedup in running time and computes the forces and potentials to within any prespecified tolerance up to machine precision.

Combining the results of this chapter with the analytical results derived in Chapter 3 and the experimental results of the sequential implementation in Chapter 4, we have presented a complete analysis of the three-dimensional N-body algorithm both theoretically and experimentally. Because of the superior speed and accuracy of our algorithm, we expect that it will find applications in astrophysics, plasma physics, fluid dynamics, and molecular dynamics. Nevertheless, there are additional improvements that we believe will enhance the practicality of the algorithm and, therefore, are worth further research effort.

We have observed in Section 5.6 that local computation—mainly floating-point calculation at each processing element of the parallel implementation—takes only about 20% of total running time. The rest of the running time is mostly spent on communication among processing elements. The results indicate that the bottleneck is the interprocessor communication. Therefore, an efficient implementation of the algorithm should explore the communication requirements of the algorithm.

In Section 5.7 we described alternatives to the current parallel implementation to improve the performance of the algorithm. By using the NEWS grid, we can exploit the localities of the algorithm and take advantage of the fast grid routing. The use of the NEWS grid will also enable us to exploit the regularities in the lateral communication of the algorithm. We expect that the resulting implementation will ease the bottleneck of the interprocessor communication.

Bibliography

[And85] L. Andrews. *Special Functions for Engineers and Applied Mathematicians.* Macmillan Publishing Company, New York, 1985.

[App85] A. Appel. An efficient program for many-body simulation. *SIAM. J. Sci. Stat. Comput.*, 6(1), Jan. 1985.

[Arn78] V. Arnold. *Mathematical Methods of Classical Mechanics.* Springer-Verlag, New York, 1978.

[BH86] J. Barnes and Piet Hut. *A Hierarchical $O(N \log N)$ Force Calculation Algorithm.* Technical Report, The Institute for Advanced Study, Princeton, NJ 08540, 1986.

[CGR87] J. Carrier, L. Greengard, and V. Rokhlin. *A Fast Adaptive Multipole Algorithm for Particle Simulations.* Research Report YALEU/DCS/RR-496, Yale University, January 1987.

[Chr84] D. Christman. *Programming the Connection Machine.* Master's thesis, MIT, Dept. of Electrical Engineering and Computer Science, Jan. 1984.

[Gol59] H. Goldstein. *Classical Mechanics.* Addison-Wesley, Reading, Massachusetts, 1959.

[GR86] L. Greengard and V. Rokhlin. *A Fast Algorithm for Particle Simulations.* Research Report YALEU/DCS/RR-495, Yale University, April 1986.

[Gre87] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems.* PhD thesis, Yale University, April 1987.

[Her86] L. Hernquist. *Performance Characteristics of Tree Codes.* Technical Report, Dept. of Astronomy, University of California, Berkeley; and The Institute of Geophysics and Planetary Physics, LLNL, Livermore, 1986.

[Hil85] D. Hillis. *The Connection Machine.* MIT Press, 1985.

[HJ86] D. Hillis and G. Steele Jr. Data parallel algorithms. *Comm. of the ACM*, 29:1170–1183, 1986.

[Hob55] E. W. Hobson. *The Theory of Spherical and Ellipsoidal Harmonics.* Chelsea Publishing Company, New York, 1955.

[knu81] D. knuth. *The Art of Computer Programming.* Volume 2, Addison-Wesley, 2nd edition, 1981.

[Kos64] N. S. Koshlyakov. *Differential Equations of Mathematical Physics.* North-Holland Publishing Company, Amsterdam, 1964.

[Lec72] M. Lecar. *Gravitational N-body Problem.* D. Reidel Publishing Company, Nordrecht-Holland, 1972.

[Pe86] W. Press and etc. *Numerical Recipes.* Cambridge University Press, 1986.

[Por85] D. Porter. *A Study of Hierarchical Clustering of Galaxies in an Expanding Universe.* PhD thesis, University of California, Berkeley, 1985.

[Ref85] *Reference Guide to Symbolics-Lisp.* Symbolics, Inc., Cambridge, Massachusetts, June 1985.

[SP67] V. Szebehely and C. Peters. Complete solution of a general problem of three bodies. *The Astronomical Journal*, 72(7), Sept. 1967.

[Zha87] F. Zhao. *An $O(N)$ algorithm for three-dimensional N-body simulations.* Master's thesis proposal, MIT, Dept. of Electrical Engineering and Computer Science, Jan. 1987.