Fast Algorithms for N-Body Simulations

Sridhar Sundaram

Ph.D Thesis

93-1370
August 1993

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

FAST ALGORITHMS FOR N-BODY SIMULATIONS

A Dissertation
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Sridhar Sundaram
August 1993

Fast Algorithms for N-body Simulation

Sridhar Sundaram, Ph.D.

Cornell University 1993

Many physical models require the simulation of a large number (N) of particles interacting through pair-wise inverse square law forces. N-body simulations are employed in fluid-dynamics, biochemistry, astrophysics, electrodynamics and molecular dynamics. The computational problem is intrinsically hard and these simulations are time-intensive.

Existing algorithms exploit either the spatial proximity of particles or the temporal proximity of states. In this thesis, we formally combine the two approaches and present an algorithm with sequential time complexity $O(N^{4/3})$ to integrate N uniformly distributed particles in 3D over one crossing time against the $O(N^{8/3})$ complexity of the direct method. Under reasonable assumptions, our algorithm is optimal. The core of the algorithm is the temporal multipole expansion of the field in terms of the space-time coordinates of the field-point.

We also present efficient parallel algorithms on the 2D and 3D Mesh and Hypercube which amortize communication costs through temporal multipole expansions.

The parallel algorithms offer an order of magnitude improvement over existing algorithms for even 10^4 particles.

A sequential implementation of the algorithm for two-dimensional N-body systems shows the predicted asymptotic scaling. A parallel version on a 16-processor Intel iPSC/860 machine is also in conformance with theoretical expectations.

Biographical Sketch

Born in Tirunelveli in India, Sridhar wandered all over Northern India from Ranchi to Patna to Shaktinagar to Delhi in pursuit of education and his parents, elder brother and sister. He received a Bachelor of Technology degree in Computer Science from Indian Institute of Technology, Delhi in August 1989. He received his Master's degree from Cornell in December 1991 and his Ph.D degree, also from Cornell in August 1993.

Acknowledgements

This thesis would be incomplete without a mention of all the people who worked behind the scenes to make it possible, directly or indirectly. I would like to thank the following people.

My advisers: John Hopcroft who instilled in me the vision and love of research and also encouraged me to attack this problem; David Chernoff, who nurtured my ideas gently steering me away from numerous pitfalls, encouraged me when I was down and whose patient help and guidance made this thesis possible; Keith Marzullo who lent a patient ear when I discussed my problems with him and made time for me from his busy schedule.

My friends in the department: Karl Böhringer, Suresh Chari, Prasad Jayanti, Mike Kalantar, Yanhong Liu, Dave Pearson, T.V. Raman, Pankaj Rohatgi, Aravind Srinivasan, Kjartan Stefansson, Ida Szafranska.

Juris Hartmanis, who gave me a clear perspective on research; Rich Zippel, who gave me a clear perspective on the N-body problem; Leslie Greengard, Feng Zhao, Sverre Aarseth and Vasudha Govindan who sent me various pieces of codes for testing purposes; Anne Gockel, who coaxed the Intel Hypercube to work at a crucial time and numerous others whose help and advice was invaluable.

Of course, none of the above people are responsible in any way for any mistakes or errors in the thesis. All such mistakes are the sole responsibility of Gilling, the workstation on which this thesis was written!

Table of Contents

List of Tables

List of Figures

List of Assumptions

Chapter 1

Introduction

The *N-body problem* involves studying the evolution of a large ensemble of particles interacting through a Coulombic or gravitational force. Given the state of N particles at some initial time, the objective is to find the state of the particles at a later time. This problem has applications in astrophysics, fluid mechanics, molecular dynamics and electrodynamics. The problem is chaotic, characterized by extreme sensitivity to initial conditions, and no analytical solutions are known even for the 3-body problem. Computer simulations are often used to study N-body phenomena.

1.1 Modeling N-body Problems

Modeling N-body phenomena on a computer requires first a physical model of the system. Figure 1.1 depicts one model for the gravitational N-body problem. Stars or galaxies are treated as point particles interacting through Newtonian gravity. Given the positions $\{\boldsymbol{x}_i\}$ and masses $\{m_i\}$ of N particles, the evolution of each particle is governed by Newton's second law of motion.

$$m_i \frac{d^2 \boldsymbol{x}_i}{dt^2} = \sum_{j \neq i}^{N} \boldsymbol{F}_{i,j} \qquad \text{for } i = 1 \ldots N \tag{1.1}$$

Here, $\boldsymbol{F}_{i,j}$ is the force exerted on particle i by particle j. For instance, the gravitational force is given by

$$\boldsymbol{F}_{i,j} = G m_i m_j \frac{\boldsymbol{x}_j - \boldsymbol{x}_i}{|\boldsymbol{x}_j - \boldsymbol{x}_i|^3} \tag{1.2}$$

The N coupled differential equations given by (1.1) govern the evolution of the system. The system can be evolved by repeatedly solving the equations for the

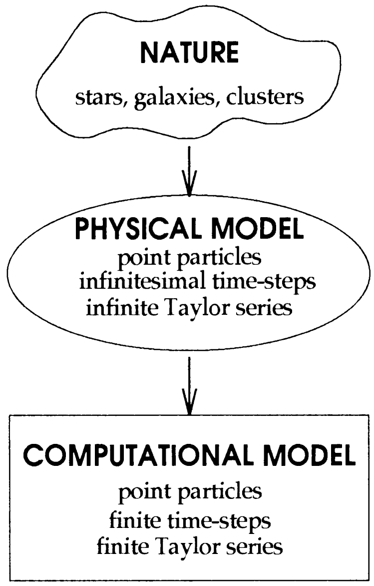


Figure 1.1: Modeling Nature through N-body Simulations

forces on each particle and advancing all particles through an infinitesimal time-step.

For a computer simulation, the solution process can be conceptualized as follows. Infinite Taylor series expansions model the force of one particle on another and integration of the expansions through infinitesimal time-steps evolves the system. To make this physical model computationally feasible, a finite Taylor series and finite time-steps are used. If all position derivatives up to the q'th are known for a particle, it can be advanced through time-step Δt by straightforward Taylor series integration.

$$\boldsymbol{x}_i(t+\Delta t) = \sum_{j=1}^{q} \frac{(\Delta t)^j}{j!} \frac{d^j \boldsymbol{x}_i(t)}{dt^j}, \qquad \text{for } i = 1 \ldots N \tag{1.3}$$

This finite treatment introduces an unavoidable error. The error will be small in a single integration step if the particles are advanced through a small distance and the potential on the particles changes by a small amount. This alone, however, is not sufficient since the chaotic nature of the N-body system causes extreme sensitivity to initial conditions. Conventionally, the measure of divergence of the two models is the difference in positions (and velocities) of the particles. Two N-body configurations initially differing by an arbitrarily small amount can diverge exponentially away from each other with increasing time. Thus, although the error in each integration step is small, the total error could be large. Reif and Tate [RT93], show that integrating N particles with $2^{-N^{O(1)}}$ accuracy through $N^{O(1)}$ time-steps is PSPACE-complete. This suggests that there is no polynomial time computable closed form representation of the trajectory of N-body systems and that the problem is intrinsically hard.

However, the statistical behavior of the system has been empirically observed to be stable. Although the exact positions and velocities of an N-body integration are unreliable, statistical properties turn out to be reliable even for a fixed precision in each time-step. Dynamical systems theory offers some theoretical justification for this in the form of "shadowing lemmas". The interested reader is referred to Guckenheimer and Holmes [GH83]. Fortunately, researchers are usually interested only in statistical properties of N-body systems. Large N (10^4–10^6) is often required for statistical validity of the results and efficient algorithms are therefore very important.

1.2 Brief History

A lot of work has been done on N-body algorithms. Most of the work is on sequential algorithms but in recent years, some work has also been done on parallel algorithms.

1.2.1 Sequential Algorithms

N-body algorithms can be classified on the basis of whether they exploit spatial location of particles or temporal proximity of successive states to reduce simulation time. See Table 1.1.

The particle-particle or direct method advocated by von Hoerner [vH60] exploits neither, Aarseth's individual time-step algorithm [Aar63, Aar85] exploits proximity in time while the Ahmad-Cohen [AC73] scheme tries to exploit both space and time proximity. All of these methods perform direct summation of forces at each particle. Particle-Mesh algorithms exploit the spatial uniformity of the particle distribution and obtain the potential by solving the Poisson equation on a mesh. They are fast but have limited resolution and are unsuitable for non-uniform particle distributions. Particle-Particle/Particle-Mesh algorithms use particle-particle methods to compute short-range interactions and particle-mesh methods to compute long-range interactions, in an attempt to get both speed and accuracy. They are very effective for nearly uniform distributions. Hockney and Eastwood [HE89] provide a good review of most of these algorithms. Tree methods organize the particles as a hierarchical tree and can be classified into three categories. Appel's algorithm [App85] which clumps particles based on nearest neighbor relationships, the Barnes-Hut [BH86] tree algorithm which groups particles by oct-trees and the fast multipole algorithm by Greengard and Rokhlin [GR87,Gre87] (and independently discovered by Zhao [Zha87]) which is also oct-tree based, all exploit only the spatial proximity of particles. While the first two have a time complexity of $O(N \log N)$ to calculate potentials on all particles, the last has a complexity of $O(N)$. Each of these tree algorithms can be generalized to exploit temporal proximity as well. Jernighan and Porter [JP89] generalize Appel's algorithm, Aarseth and McMillan[1] [MA93] generalize Barnes-Hut and this thesis generalizes the fast multipole algorithm.

1.2.2 Parallel Algorithms

Considerable effort has gone into vectorization of N-body codes for fast execution on a supercomputer. McMillan [McM86] discusses vectorization of the individual time-step method. Hernquist [Her90], Makino [Mak90], Barnes [Bar90] and Schmidt and Lee [SL91] discuss vectorization of tree codes. In another direction, GRAPE-3 (GRavity PipE-3) [OME$^+$92], a specialized computer for gravitational N-body simulations has been developed.

Some work has been done on parallel implementations also. See Table 1.2. Salmon [Sal91] implements a parallel version of the Barnes-Hut algorithm on the n-CUBE. Singh et al [SHT$^+$92] study load balancing issues in parallelization of the Barnes-Hut and Fast Multipole algorithms on the Stanford DASH multiprocessor. Parallel versions of the fast multipole algorithm have been implemented on various architectures – Zhao [Zha87] on an 8k-processor CM-2, Greengard and Gropp [GG90] on an 18-processor Encore Multimax, Zhao and Johnsson [ZJ91] on a 16K-processor CM-2 and Leathrum and Board [LB91] on a 8-processor B012 transputer torus. In all of the parallel tree code implementations, particles are advanced synchronously in lock-step.

1.3 Outline of the Thesis

1.3.1 Philosophy

In this thesis, we attempt to provide more rigorous justifications for some empirically well-known concepts. These justifications are necessary to understand the new algorithms that we introduce. However, the N-body problem is intrinsically chaotic and we are often forced to make assumptions which are valid empirically

[1]Came to the author's knowledge while this thesis was being written

but difficult to prove formally. We explicitly state such assumptions with intuitive reasons for making them whenever possible.

The new algorithms we introduce are intended for practical implementations. The presentation format, however, has been chosen for ease of theoretical justification. In the chapter on implementation, we explain how the algorithms would actually be implemented.

1.3.2 Contributions and Outline

This thesis presents the temporal multipole algorithm which exploits both spatial and temporal proximity.

In chapter 2, we briefly describe the fast multipole algorithm and the individual time-step algorithm which are building blocks in our new algorithm.

In chapter 3, we derive the individual time step algorithm rigorously and develop the new concept of temporal multipoles which expand the potential in space-time coordinates. Unlike previous spatio-temporal algorithms, which are mostly empirical, we give rigorous error bounds. Tools for manipulating temporal multipole expansions are also discussed.

In chapter 4, we present a sequential algorithm based on temporal multipoles. The notion of particle consistency across cells is introduced and blocked time-steps are discussed. The algorithm is shown to be essentially optimal for any N-body simulation.

A parallel version of the algorithm for homogeneous particle distributions is described in chapter 5. The algorithm is designed to work on coarse-grained parallel machines. It is shown that the communication costs can be effectively amortized by using temporal multipole expansions and that the algorithm remains optimal.

Chapter 6 discusses non-homogeneous distributions. A local solution to the issue of load-balancing is proposed and analyzed where processors redistribute work among neighbors based on the "center-of-work" heuristic. For systems of particles which consist of locally homogeneous subsystems, this load-balanced algorithm would be nearly optimal.

In Chapter 7, we adapt the parallel algorithms to the mesh and the hypercube. These algorithms are an order of magnitude improvement over existing algorithms both in theory and in practice and make feasible simulations which previously were inordinately expensive. For the hypercube, the algorithm is optimal while for a mesh, it is almost optimal.

Finally, chapter 8 discusses implementation results for both the sequential and parallel versions of the algorithms for two dimensional N-body systems. The sequential algorithm is tested against standard algorithms for three different input distributions. In each case, it performs substantially better. The algorithm actually shows the predicted asymptotic scaling of time-complexity with number of particles. The parallel algorithm, implemented on a 16-processor machine configured as a 2D-mesh also does substantially better than the conventional algorithms. While the performance scaling is not linear for 16 processors, it is expected to be almost linear for a larger number of processors.

Table 1.1: Time Line of Sequential N-body Algorithms

Year	Space[a]	Time[b]	Algorithm	Complexity[c]
1960:	$\times$	$\times$	von Hoerner	$O(N^{8/3})$
			$\downarrow$	
1963:	$\times$	$\surd$	Aarseth	$O(N^{7/3})$
			$\downarrow$	
1973:	$\surd$	$\surd$	Ahmad-Cohen	$O(N^{25/12})$
			$\downarrow$	
	$\surd$	$\times$	Particle-Mesh	$O(N^{5/3}\log N)$
			$\downarrow$	
	$\surd$	$\times$	Particle-Particle/Particle-Mesh	$O(N^{5/3}\log N)$
			$\downarrow$	
1985:	$\surd$	$\times$	Appel Tree	$O(N^{5/3}\log N)$
			$\downarrow$	
1986:	$\surd$	$\times$	Barnes-Hut Tree	$O(N^{5/3}\log N)$
			$\downarrow$	
1987:	$\surd$	$\times$	Greengard Tree	$O(N^{5/3})$
			$\downarrow$	
1989:	$\surd$	$\surd$	Jernighan-Porter	$O(N^{4/3}\log N)$
			$\downarrow$	
1993:	$\surd$	$\surd$	Aarseth-McMillan	$O(N^{4/3}\log N)$
			$\downarrow$	
1993 :	$\surd$	$\surd$	Temporal Multipole	$O(N^{4/3})$

[a]Exploits Space Proximity
[b]Exploits Time Proximity
[c]Time to integrate homogeneous 3D system of N particles for one crossing time

Table 1.2: Time Line of Parallel N-body Algorithms

Year	Algorithm	Architecture	Complexity[a]
1987:	Zhao	Hypercube	$O(N^{2/3}\log N)$
	↓		
1990:	Greengard-Gropp	Shared Memory	$O(N^{2/3}\log N)$
	↓		
1991:	Salmon	Hypercube	$O(N^{2/3}\log N)$
	↓		
1991:	Zhao-Johnsson	Hypercube	$O(N^{2/3}\log N)$
	↓		
1991:	Leathrum-Board	1D Torus	$O(N^{5/3})$
	↓		
1992:	Singh et al	DASH multiprocessor	$O(N^{7/6})$
	↓		
1993 :	Temporal Multipole	Hypercube	$O(N^{1/3})$
	↓		
1993 :	Temporal Multipole	3D Mesh	$O(N^{1/3}\log N)$

[a] Time to integrate a homogeneous 3D system of N particles for one crossing time with $O(N)$ processors. Estimated where unavailable.

Chapter 2

Background

The total time complexity of an N-body simulation algorithm can be expressed as a product of its spatial complexity and its temporal complexity. The first depends on how efficiently it exploits the spatial proximity of particles while the second depends on how efficiently it exploits the temporal proximity of states.

We briefly describe the fast multipole algorithm which exploits spatial proximity of particles and the individual time-step algorithm which exploits temporal proximity of states. These are used as building blocks in our new algorithm.

2.1 Physical Model

We will consider a set of N particles in a three dimensional coordinate system interacting through a gravitational or coulombic force. To keep the treatment general, we will use the following force model. If a single charge of intensity m is located at $\boldsymbol{x}'$, the field and potential at any point $\boldsymbol{x}$ is given by

$$\begin{aligned} \boldsymbol{E}(\boldsymbol{x}, \boldsymbol{x}') &= m\frac{\boldsymbol{x} - \boldsymbol{x}'}{|\boldsymbol{x} - \boldsymbol{x}'|^3} \\ \phi(|\boldsymbol{x} - \boldsymbol{x}'|) &= m\frac{1}{|\boldsymbol{x} - \boldsymbol{x}'|} \end{aligned} \tag{2.1}$$

The electrostatic and gravitational force laws can be recast in this form by suitable normalization of variables.

Some physical systems are adequately described by two-dimensional analogues of these equations where all the particles are confined to a plane. In such cases, a two dimensional simulation is conceptually simpler and computationally cheaper. In two dimensions, the field and potential at point $\boldsymbol{x}$ due to a charge of intensity

m located at $\boldsymbol{x}'$ is given by

$$\begin{aligned} \boldsymbol{E}(\boldsymbol{x}, \boldsymbol{x}') &= m \frac{\boldsymbol{x} - \boldsymbol{x}'}{|\boldsymbol{x} - \boldsymbol{x}'|^2} \\ \phi(|\boldsymbol{x} - \boldsymbol{x}'|) &= m \log(\frac{1}{|\boldsymbol{x} - \boldsymbol{x}'|}) \end{aligned} \tag{2.2}$$

Note that a point charge in two dimensions corresponds to a line charge in three.

2.1.1 Conventions

We will often explain concepts in one dimension, which although unphysical, allows a clearer exposition. As is conventional, we will denote the magnitude or norm of a vector $\boldsymbol{x}$ by x. Since the potential is a scalar function of a scalar variable, while the force is a vector function of a vector variable, we will use the potential to simplify theoretical development. This entails no loss of generality because the force can be obtained as the gradient of the potential. The physical potential will always be represented by ϕ. We will use Φ (which is somewhat confusing but by now standard) to denote the multipole expansion of the far potential due to a collection of particles. While ϕ is the actual potential, Φ is a representation of that potential as a multipole expansion.

2.2 The Direct Algorithm

The simplest and most straightforward method of integrating an N-body system is the brute-force or direct method. The disadvantage of this simple method is its time-complexity. All $\frac{N(N-1)}{2}$ pairs of particles are interacted in each time-step giving an $O(N^2)$ time-complexity per time-step. This is too high even for N as small as 1000. The algorithm is sketched below.

1. Find force on each particle due to every other particle by direct pairwise calculation.
2. Assume that the force does not change for a short time-step Δt and advance all particles through the time-step Δt.
3. Go to step 1.

2.3 Exploiting Spatial Proximity

Spatial proximity of particles can be exploited to decrease the amount of computation required. Particles which are close together experience approximately the same force from far away particles. This intuition translates into the formal concept of multipole expansions.

2.3.1 Multipole Expansions

The *multipole expansion* expresses the potential due to a particle as a Taylor series expansion in the coordinates of the point at which the potential is to be evaluated.

Example 2.3.1 Let the potential, *in one dimension*, be given by $\Phi(r) = \frac{1}{r}$ at distance r from a point charge. Let a unit charge be placed at point $\boldsymbol{x}'$. Then, the potential at some point $\boldsymbol{x}$, $x > x'$, is given by

$$\Phi(|\boldsymbol{x} - \boldsymbol{x}'|) = \frac{1}{|\boldsymbol{x} - \boldsymbol{x}'|} = \frac{1}{x}\sum_{i=1}^{\infty}\left|\frac{x'}{x}\right|^{i} \tag{2.3}$$

This series sum, expressing the potential at any point $\boldsymbol{x}$ due to the charge at point $\boldsymbol{x}'$, is called the multipole expansion of the potential due to the charge. If all terms $\left|\frac{x'}{x}\right|^{p+1}$ and higher in the series are ignored, the error in the approximation will be of order $\left|\frac{x'}{x}\right|^{p}$. This truncated series corresponds to a *p-term multipole expansion* of the potential. The multipole expansion converges only when $x > x'$. If $x \geq 2x'$, the error is very small at $O(2^{-p})$ for a p-term expansion and in this sense the expansion converges *rapidly* outside a circle of radius $2x'$. Being only interested in finite multipole expansions, we will use the term convergence to mean *rapid* convergence.

If we have n charges at points $\boldsymbol{x}'_0, \boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_n$, such that $x > 2x'_i$ for i ranging from 0 to n, then, the potential due to the charges at $\boldsymbol{x}$ can be found as summations of the potentials due to the individual charges as given by (2.3). □

For an upper bound on the error of ϵ a truncated multipole expansion with $O(\log \frac{1}{\epsilon})$ terms suffices. The field at a point due to a collection of particles can be obtained as the sum of the multipole expansions of the particles. By exploiting spatial proximity cleverly to combine multipole expansions, the forces on all N particles can be computed in $O(N)$ time (as against $O(N^2)$ for direct).

Note that the sum of the multipole expansions of a set of particles does not converge rapidly in the neighborhood of the set. In tree algorithms, the hierarchical structure induced by a tree is used to sum up the expansions fast when particles are far away. For particles close by, the potential is found by pairwise direct interaction.

2.3.2 Fast Multipole Algorithm

We describe the fast multipole algorithm ([GR87]) in one dimension. We will describe how to evaluate the potential at a point. The force can be obtained as the gradient of the potential.

Consider N particles arranged along a straight line. The fast multipole algorithm takes a cell-oriented rather than a particle-oriented view. The line is divided into equal length intervals which we will call *cells*. The cells touching a cell (including itself) are its *neighbors*.

2.3.2.1 Near and Far Potentials

The potential at a particle a located in cell i is partitioned into the *far field* and the *near field* so that

$$\Phi_a = \Phi_{a,\text{farfield}} + \Phi_{a,\text{nearfield}} \tag{2.4}$$

The near field potential, due to particles in the neighbors of cell i, is calculated by direct particle-particle interaction. The far field potential, attributed to particles outside the neighbors of cell i, is evaluated as a multipole expansion at cell i. This expansion is used to compute the far potential at each particle in the cell. The partitioning is necessary for two reasons. Firstly, the multipole expansion of particles in a cell does not converge sufficiently rapidly in its neighbor cells[1]. Secondly, the near field is often modified/softened in computer simulations for physical reasons.

To efficiently find the far potential at a cell, a tree-structured hierarchy of cells is introduced. The cells into which the line is divided are at level n, the bottom level. They are connected in a binary tree fashion as shown in Figure 2.1. Two *children* cells at level h combine to form a *parent* cell at level $(h-1)$. The cell at level 0, the top level, is identified with the 0'th cell and does not have a parent.

2.3.2.2 Notation

Before describing the algorithm, we introduce some notation.

Parent(i) parent of cell i, i not at level 0 *(e.g.* Parent*(10) is 4 in Figure 2.1)*

Children(i) set of children of cell i, i not at level n *(e.g.* Children*(10) is* $\{21,22\}$*)*

Neighbor(i) set of neighbors of cell i (including i) *(e.g.* Neighbor*(10) is* $\{9,10,11\}$*)*

[1] In 3 dimensions, convergence is not sufficiently rapid even in cells next to neighbor cells. In this case, the near field is redefined as being due to particles in the cell's neighbors and the cells next to the neighbors. The far field is also appropriately redefined.

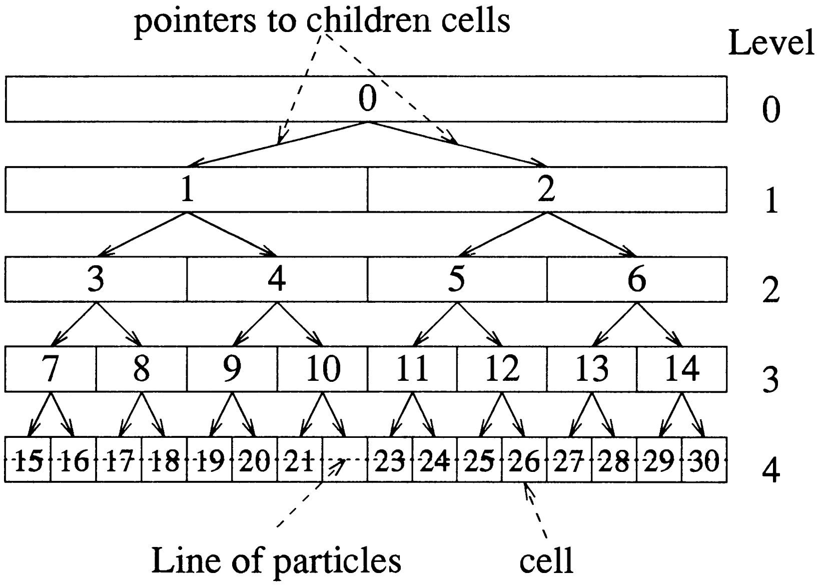


Figure 2.1: Cell hierarchy for fast multipole algorithm

Ilist(i) set of cells which are children of neighbors of i's parent, but which are not neighbors of cell i *(e.g.* IList*(10) is* $\{7,8,12\}$*)*

Φ_i the p-term multipole expansion with origin at center of cell i, describing potential *outside* Neighbor(i) due to all particles *inside* cell i

Ψ_i the p-term multipole expansion with origin at center of cell i, describing potential *inside* cell i due to all particles *outside* Neighbor(i)

The potential, Φ (Ψ), is represented as a multipole expansion about an origin point in space and converges outside (inside) a sphere about that origin. Two multipole expansions about different origins cannot be directly added. Some technique to shift the origins of multipole expansions is required that guarantees convergence. We abstractly describe some operators for this purpose following Katzenelson [Kat88] (for details see [GR87]). See Figure 2.2.

$$\begin{aligned}
\text{Translate}_{k,i} &: \Phi_k \to \widehat{\Phi}_i, \quad k \in \text{Children}(i) \\
\text{Shift}_{k,i} &: \Psi_k \to \widehat{\Psi}_i, \quad k = \text{Parent}(i) \\
\text{Convert}_{k,i} &: \Phi_k \to \widehat{\Psi}_i, \quad k \in \text{IList}(i)
\end{aligned}$$

The hatted symbols denote partial contributions to the appropriate potential at that cell. While the functions Translate and Shift introduce no error for finite

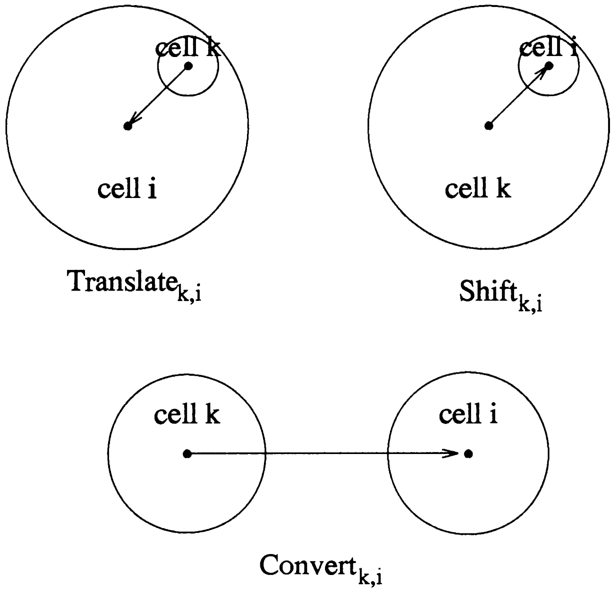


Figure 2.2: Changing the origins of multipole expansions

multipole expansions, Convert does introduce an error proportional to 2^{-p} for a p-term multipole expansion. Let '+' be an operator which adds two p-term multipole expansions term-wise to produce a new p-term multipole expansion. Using these operators, we can specify equations to form Φ and Ψ for each cell.

$$\Phi_i = \sum_{j \in \mathrm{Children}(i)} \mathrm{Translate}_{j,i}(\Phi_j) \tag{2.5}$$

$$\Psi_i = \mathrm{Shift}_{\mathrm{Parent}(i),i}(\Psi_{\mathrm{Parent}(i)}) + \sum_{k \in \mathrm{IList}(i)} \mathrm{Convert}_{k,i}(\Phi_k) \tag{2.6}$$

$$\Psi_0 = 0 \tag{2.7}$$

The recursive equations above are the driving force behind tree algorithms in general and the fast multipole algorithm in particular.

2.3.2.3 The Algorithm

In essence, the fast multipole algorithm is an efficient method of finding the far potential Ψ at each cell at the bottom level due to all particles outside the neighbors of the cell. See Figure 2.3. The far potential on a particle can be found by evaluating the Ψ of its cell at its position. The near potential due to particles in the neighbor cells is found by direct pairwise interactions. The total potential is

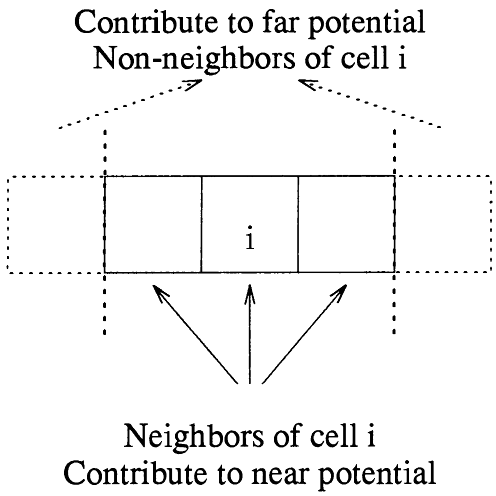


Figure 2.3: Near and far potentials at a cell

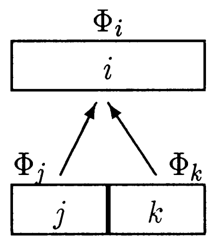


Figure 2.4: Forming Φ_i for cell i not at bottom level

just the sum of the far and near potentials. We first give an informal overview of the algorithm followed by a more formal description.

In the first pass of the algorithm, Φ, the far potential outside each cell due to particles within the cell is formed. For each bottom level cell, Φ is directly formed as the sum of multipole expansions of all particles within. By equation (2.5), Φ for a lower level cell can be formed when the Φ for each of its children is known. This is shown in Figure 2.4. Therefore, Φ can be formed for each cell in the tree moving from bottom to top.

In the second pass, Ψ, the far potential at each cell due to all particles outside the neighbors of the cell is formed. For the 0'th cell, which comprises the entire system, Ψ is zero. For higher level cells, Ψ is formed using equation (2.6) from top to bottom in the tree. This is shown in Figure 2.5.

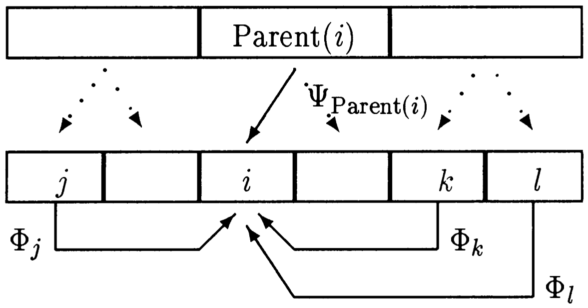


Figure 2.5: Forming Ψ_i for cell i not at top level

Once Ψ is available at each bottom level cell, the far potential can be found at each particle. For each particle in each bottom level cell, the particle's position coordinates are substituted into the multipole expansion Ψ of the cell to obtain the far potential at the particle. The near potential at each particle is found by direct interaction with all particles in the neighbors of the cell.

We give a more formal description of the fast multipole algorithm below.

0. Choose number of levels in the tree $n \approx \log_2 N$, a precision ϵ, and order of multipole expansion $p = \lceil -\log_2 \epsilon \rceil$.

1. For each cell i at level n: Compute Φ_i as the sum of the multipole expansions of the particles in the cell about the center of cell i.

2. For each cell i at levels less than n: Compute Φ_i using equation 2.5 moving up the tree of cells.

3. For each cell i at levels greater than 0: Compute Ψ_i using equation 2.6 moving down the tree of cells.

4. For each particle a in each cell i at level n:

   (a) Evaluate Ψ_i at a's position to get the far field potential at the particle.

   (b) Find the near field potential at a due to interactions with particles in Neighbor(i).

   (c) Add far and near potentials to get total potential on a.

Note that in this algorithm, accuracy is controlled entirely by the order of the multipole expansion p unlike the Barnes-Hut algorithm which has a tunable parameter θ in addition to the multipole expansion order.

Remark *Time and Space Complexity:* Shifting, translating and converting multipole expansions are all $O(1)$ (actually $O(p^2)$ in 2D and $O(p^4)$ in 3D) operations once p, the number of terms in the multipole expansion has been fixed. Each cell at the bottom level contains $O(1)$ number of particles and for a uniform distribution of particles (see next subsection), the total number of cells at all levels is $O(N)$. Hence, the amount of computation in conversions, translations and shifts which occur in the algorithm is $O(N)$ as well. The work done in the near field computation is also $O(1)$ for each cell at the bottom level. Thus, the total work done to compute forces on all particles is linear in N, the number of particles (per time-step). The space complexity is easily seen to be $O(N)$ for homogeneous distributions.

Remark *Higher Dimensions:* Generalization of the fast multipole algorithm to higher dimensions is straightforward. Figure 2.6 depicts cells in two dimensions.

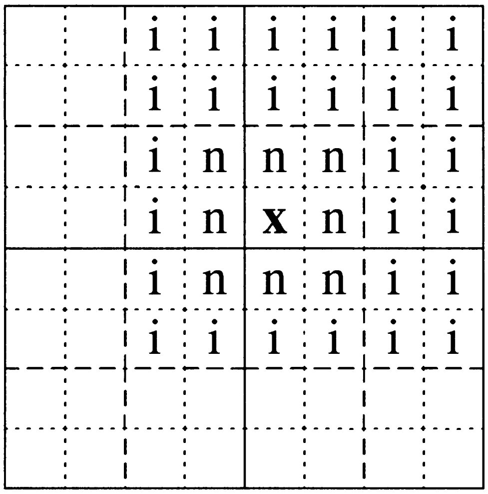


Figure 2.6: Cells for fast multipole algorithm in 2 dimensions
Cells at different levels are shown with different kinds of lines.
Cells marked **n** are in Neighbor(**x**). Cells marked **i** are in IList(**x**).

Each cell now has 4 children, 9 neighbors (including itself) and 27 ilist members. The rest of the algorithm remains unchanged.

2.3.3 Adaptive Multipole Algorithm

For non-uniform particle distributions, the algorithm described in the previous section creates a large number of empty or nearly empty cells and its time complexity per time-step is no longer linear in the number of particles. An adaptive algorithm, which starts with one large cell containing all the particles and divides it recursively into smaller cells only when the number of particles in a cell is sufficiently large, performs much better. With this algorithm, the cell hierarchy does not have a fixed number of levels, and leaf cells need not be of the same size. See Figure 2.7.

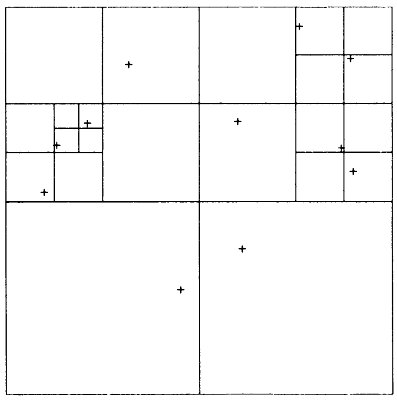


Figure 2.7: Cells for non-uniform particle distributions in the adaptive algorithm There is at most one particle in any box.

The disparity in cell sizes complicates the algorithm. The function "Convert" cannot be used to move the multipole expansion of a cell k in IList(i) to cell i, when the size of cell k is larger than the size of cell i. The converted multipole expansion does not converge within cell k. Instead, conversion has to be done on a particle by particle basis. Imagine an infinitesimal cell around each particle in cell k. "Convert" can be used to move this cell's multipole expansion to the center of cell i. Similarly, "Convert" cannot be used when cell k in IList(i) is smaller in size than cell i but not sufficiently far away. In this case, the far potential due to the multipole expansion of cell k has to be directly evaluated at each particle in cell i. For details see [GR87].

With these modifications, it can be shown that the adaptive multipole algorithm has a linear complexity in both time and space for arbitrary particle distributions per time-step.

2.4 Exploiting Temporal proximity

The time-proximity of particles can also be exploited to decrease the amount of computation required. The intuition is that the force on a particle is almost the same from one instant of time to the next. Therefore, the force can be extrapolated avoiding expensive recalculation. In simulations, where particles evolve on different time-scales in different regions, exploiting time proximity is particularly effective. We will formalize this notion in the next chapter, but algorithms based on this approach already exist [AC73,JP89,MA93].

Time-steps in the computational model are usually chosen so that each particle takes $O(\frac{1}{\epsilon})$ steps to reach its closest neighbor, with ϵ chosen suitably small. This ensures that the computational model is close to the physical model. When all particles share the same time-steps, this time-step is determined by the closest pair of particles in the entire system. With individual time-steps this restriction is not imposed and particles evolve at their own rates.

Example 2.4.1 We can estimate the savings obtained using individual time-steps in integrating a homogeneous two-dimensional distribution of particles for a constant amount of physical time. Consider a homogeneous distribution of N particles inside a unit circle. (Below, $\sim$ indicates proportionality.)

Individual Time Steps: The average inter-particle separation of $\sim \frac{1}{\sqrt{N}}$ determines the time-step for the individual time-step scheme. On average, each particle will, take $\sim \sqrt{N}$ number of time-steps to evolve through a fixed amount of physical time. The number of time-steps for the individual time-step scheme is $\sim \sqrt{N}$.

Shared Time Steps: The expected value of the minimum inter-particle separation can be calculated to be $\sim \frac{1}{N}$. Therefore, on average $\sim N$ shared time-steps will be required to evolve the system through one crossing time. The number of time-steps for the shared time-step scheme is $\sim N$.

Similar estimates can be made for power law and other distributions with more effort. In each case, the individual time-step scheme requires a much lower number of iterations. □

For homogeneous 3D distributions, Makino and Hut [MH89] estimate that individual time steps yield a gain of $O(N^{\frac{1}{3}})$ over shared time-steps for integration of all particles over one crossing time[2].

[2]The crossing time (more precisely the half-mass crossing time) is the time needed to traverse the inner part containing half the mass of a self-gravitating system of particles at a speed equal to the dispersion velocity.

2.4.1 Individual Time-step Scheme

The fast multipole algorithm imposes a common or shared time-step on all particles and all particles move forward in lock-step fashion. In individual time-step schemes, each particle has its own time-step based on how rapidly the force experienced by it changes. As we just saw, individual time-step schemes are a substantial improvement over shared time-steps. However, care has to be taken to ensure stability. We describe Aarseth's individual time-step algorithm [Aar85] briefly. Step 2 in the algorithm has been chosen to ensure stability.

Let the current time be t and suppose that particle b needs to be moved at time t_b.

1. Find particle a such that $(t_a - t)$ is minimum.
2. Project all particle positions to time t_a using derivatives of their old forces (the old Taylor series expansion).
3. Calculate force on particle a due to all other particles (by direct interaction).
4. Find new Taylor series expansion for particle a and use this expansion to predict a new time-step δt_a.
5. Set t to t_a and t_a to $t_a + \delta t_a$. Go to step 1.

2.5 Complexities

Usually, N-body simulations are done for a duration which is some multiple of a physical time. This physical time is usually the half-mass crossing time. The time complexity of an algorithm is the number of computational steps required to evolve the system from the initial conditions to the desired time. It is instructive to look at the complexities of the algorithms we have discussed so far for homogeneous 3D particle distributions. See Table 2.1.

The direct pairwise algorithm uses neither space nor time optimizations. It requires $O(N^2)$ time per iteration. Since shared time-steps are used, the number of iterations required is $O(N^{2/3})$. The total time complexity is $O(N^{8/3})$.

The fast multipole algorithm uses sophisticated space optimizations to reduce the time per iteration to $O(N)$ but still uses shared time-steps. Its total time-complexity is $O(N^{5/3})$.

The individual time-step algorithm does no space-optimization whatsoever but reduces the required number of iterations to $O(N^{1/3})$. Its total time complexity is $O(N^{7/3})$.

Table 2.1: Space-Time Complexities in 2D and 3D for integrating N particles

Complexity[a] $\rightarrow$ Algorithm $\downarrow$	2D			3D		
	Space	Time	Total	Space	Time	Total
Direct Shared	2	1	3	2	$\frac{2}{3}$	$\frac{8}{3}$
Direct Individual	2	$\frac{1}{2}$	$\frac{5}{2}$	2	$\frac{1}{3}$	$\frac{7}{3}$
Fast Multipole	1	1	2	1	$\frac{2}{3}$	$\frac{5}{3}$
Temporal Multipole	1	$\frac{1}{2}$	$\frac{3}{2}$	1	$\frac{1}{3}$	$\frac{4}{3}$

[a]Exponent q of N in $O(N^q)$ to integrate homogeneous system of N particles over 1 crossing time

The temporal multipole algorithm of this thesis combines the space optimizations of the multipole algorithm and the time optimization of the individual time-step algorithm to obtain a total time-complexity of $O(N^{4/3})$.

Chapter 3

The Temporal Multipole Expansion

Multipole expansions express the potential due to a set of particles as a series expansion in the spatial coordinates. We introduce the more general *temporal multipole expansions* which express the potential due to the set of particles as a series expansion in space-time coordinates.

The benefits are twofold. First, a temporal multipole expansion can exploit both spatial and temporal locality of particles. Within its range of validity, it can be used to calculate potentials inexpensively and accurately. This can lead to substantial speedups as discussed in the next chapter. Secondly, the time of creation of a temporal expansion is decoupled from the time of its use. Parallel algorithms based on temporal expansions can, therefore, amortize communication costs over successive N-body iterations very efficiently. This will be discussed in chapter 5.

3.1 The Temporal Expansion

We first discuss the temporal expansion in isolation. Combining the temporal expansion with the multipole expansion yields the temporal multipole expansion.

Temporal expansions are based on the following simple idea. Consider the far force at a point due to a set of particles. Changes in the far force are caused by movement of particles in the set. By characterizing this movement in terms of derivatives of the particle positions with respect to time, we can predict future values of the force.

Let a particle be integrated through one time-step, using a Taylor series expansion of the potential at the particle. The physical model requires an infinite

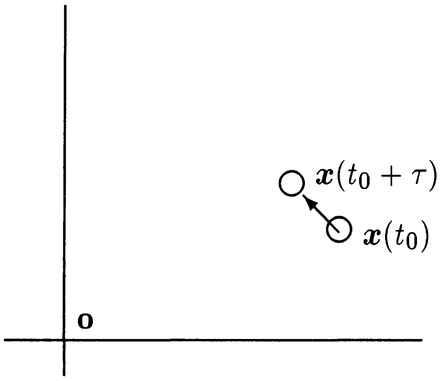


Figure 3.1: Temporal Expansion for a Particle

number of terms in the Taylor series and infinitesimal time-steps, while in any computational model, only a finite number of terms and finitely large time-steps are feasible. This can lead to divergence of the computational model from the physical model as discussed in Section 1.1. Nevertheless, we can characterize the divergence between physical and computational models provided both use finite Taylor series expansions. The following lemma examines how this error behaves with the number of terms in the series and the size of the time-step.

3.1.1 Temporal Expansion for a Particle

We will express the potential at a space-time point, $\langle \boldsymbol{r}, t \rangle$, as a time-varying function represented by $\Phi(\langle \boldsymbol{r}, t \rangle)$.

Lemma 3.1.1 (Temporal Expansion for a Particle) *Consider a unit point charge located at $\langle \boldsymbol{x}(t_0), t_0 \rangle$. The potential at the origin $\langle \mathbf{o}, t \rangle$, for $t \in [t_0 - \tau, t_0 + \tau]$, where τ is the* **time-step** *through which the charge is integrated, is given by*

$$\Phi(\langle \mathbf{o}, t \rangle) = \sum_{i=0}^{\infty} b_i t^i \text{ where } b_i = \sum_{k=i}^{\infty} \frac{1}{k!} \frac{d^k \phi(x(t_0))}{dt^k} \binom{k}{i} (-t_0)^{k-i} \tag{3.1}$$

Let η be a fixed positive constant with $\eta \in [0, \frac{\log 2}{2}]$. Let time-step τ be chosen such that

$$\tau^i \left| \frac{d^i x(t_0)}{dt^i} \right| \leq \eta^i \, |x(t_0)| \qquad \text{where } i \geq 1 \tag{3.2}$$

Then, for any integer q, $q \geq 1$ and for d-dimensional potential function ϕ,

$$\left|\Phi(\langle \mathbf{o}, t\rangle) - \sum_{i=0}^{q} \frac{(t-t_0)^i}{i!}\frac{d^i\phi(x(t_0))}{dt^i}\right| \leq |\phi(x(t_0))|^{d-2}\left(\frac{\eta}{\log 2}\right)^{q+1} \tag{3.3}$$

Proof. We first expand $\Phi(\langle \mathbf{o}, t\rangle)$ as a Taylor series with respect to t.

$$\Phi(\langle \mathbf{o}, t\rangle) = \sum_{i=0}^{\infty} \frac{(t-t_0)^i}{i!}\frac{d^i\Phi(\langle \mathbf{o}, t_0\rangle)}{dt^i} = \sum_{i=0}^{\infty} \frac{(t-t_0)^i}{i!}\frac{d^i\phi(x(t_0))}{dt^i} \tag{3.4}$$

Using MacLaurin's theorem to express the series in powers of t we obtain (3.1). To obtain the error bound (3.3), note that the error E is given by

$$E = \left|\Psi_{\langle \boldsymbol{x}, t_0\rangle}(\langle \mathbf{o}, t\rangle) - \sum_{i=0}^{q} \frac{(t-t_0)^i}{i!}\frac{d^i\phi(x(t_0))}{dt^i}\right| \leq \sum_{i=q+1}^{\infty} \frac{\tau^i}{i!}\left|\frac{d^i\phi(x(t_0))}{dt^i}\right| \tag{3.5}$$

Using an elementary theorem of calculus on differentiating implicit functions [Edw92],

$$\frac{\tau^n}{n!}\frac{d^n\phi(x(t_0))}{dt^n} = \sum_{r=1}^{n} {}^nK_r \frac{\tau^n}{r!}\frac{d^r\phi(x(t_0))}{dx^r} \tag{3.6}$$

where ${}^nK_r$ is the coefficient of τ^n in $(x(t_0+\tau) - x(t_0))^r$.
We can bound $|x(t_0+\tau) - x(t_0)|$ using (3.2) as follows.

$$|x(t_0+\tau) - x(t_0)| \leq \sum_{i=1}^{\infty} \frac{\tau^i}{i!}\left|\frac{d^i x(t_0)}{dt^i}\right| \leq \sum_{i=1}^{\infty} \frac{\eta^i}{i!} x(t_0) \leq (e^\eta - 1)x(t_0) \tag{3.7}$$

Therefore,

$$|x(t_0+\tau) - x(t_0)|^r \leq (e^\eta - 1)^r x(t_0)^r \Rightarrow |\tau^n\ {}^nK_r| \leq [(e^\eta-1)^r : \eta^n]x(t_0)^r \tag{3.8}$$

where $[Z : \eta^i] \stackrel{\text{def}}{=}$ term involving η^i in Z.
Now, consider the 3D potential function $\phi(x(t)) = \frac{1}{x(t)}$. (For the 2D potential function, $\phi(x(t)) = -\log x(t)$, the argument is similar.)

$$\left|\frac{d^r\phi(x)}{dx^r}\right| = \frac{r!\,|\phi(x)|}{x^r} \tag{3.9}$$

Combining (3.6), (3.8) and (3.9),

$$\begin{aligned}
\left|\frac{\tau^n}{n!}\frac{d^n\phi(x(t_0))}{dt^n}\right| &\le \sum_{r=1}^{n}\left|{}^nK_r\frac{\tau^n}{r!}\frac{d^r\phi(x(t_0))}{dx^r}\right| \\
&\le |\phi(x(t_0))|\sum_{r=1}^{n}[(e^\eta-1)^r:\eta^n] \\
&\le |\phi(x(t_0))|\sum_{r=1}^{\infty}[(e^\eta-1)^r:\eta^n] \\
&\le |\phi(x)|\,[\frac{1}{2-e^\eta}:\eta^n] \qquad (3.10)
\end{aligned}$$

Using this in (3.5), the error E is given by

$$E \le |\phi(x(t_0))|\sum_{i=q+1}^{\infty}\left|[\frac{1}{2-e^\eta}:\eta^i]\right| \qquad (3.11)$$

To upper bound the summand, we look at the function $[(1-\frac{e^\eta}{2})^{-1}:\eta^k]$, $k>1$.

$$[(1-\frac{e^\eta}{2})^{-1}:\eta^k] = \sum_{i=1}^{\infty}[\frac{e^{i\eta}}{2^i}:\eta^k] = \sum_{i=1}^{\infty}\frac{\eta^k i^k}{k!2^i} \le \frac{\eta^k}{k!}\int_0^\infty\frac{i^k}{2^i}di \qquad (3.12)$$

The integral above can be simplified as follows.

$$\int_0^\infty\frac{y^k}{2^y}dy = \int_0^\infty\frac{y^k}{e^{y\log 2}}dy = \frac{1}{(\log 2)^{k+1}}\int_0^\infty e^{-z}z^k dz = \frac{\Gamma(k+1)}{(\log 2)^{k+1}} = \frac{k!}{(\log 2)^{k+1}} \qquad (3.13)$$

Combining (3.11), (3.12) and (3.13),

$$E \le |\phi(x(t_0))|\sum_{i=q+1}^{\infty}\frac{\eta^i}{2(\log 2)^{i+1}} \le |\phi(x(t_0))|\frac{\eta^{q+1}}{2(\log 2)^q(\log 2-\eta)} \qquad (3.14)$$

For η less than $\frac{\log 2}{2}$, the error bound in (3.3) follows. □

Remark *Finite Number of Terms:* While (3.2) requires an infinite set of inequalities to be satisfied, note that if only q derivatives of x are maintained, the inequalities for higher derivatives are automatically satisfied, since they are all taken to be zero. Similarly, for a q-term temporal expansion, the error-bound proof requires summation only up to q terms (and not ∞) to evaluate the coefficients b_i.

Corollary 3.1.2 *In lemma 3.1.1, the error after k time-steps with parameter η ($\eta \ll \log 2$) is approximately the same as the error after 1 time-step with parameter η', where η' is $k\eta$ for worst case error and $\eta\sqrt{k}$ for expected error, provided $\eta' \in [0, \frac{\log 2}{2}]$.*

Proof. Since, η is small, the size of the time-step remains approximately the same in successive iterations. This justifies the worst case error bound. For the average case, note that errors and changes in positions in successive iterations can be thought of as normally distributed random variables. So their sum will have an expected error corresponding to $\eta\sqrt{k}$. □

Here, η plays the role of an efficiency parameter. For smaller values of η, the computational model is more faithful to the physical model, at the expense of increased computation. With a fixed value for η, the error in potential is relative to the original potential and could be large in close encounters. If η is made a function of r, the absolute error in potential can be controlled. We will discuss later how η is determined. For a desired precision, ϵ, the number of terms in the temporal expansion can be determined from (3.3).

We define the *validity period* of a particle temporal expansion as $[t_0 - \tau, t_0 + \tau]$, with t_0 and τ as in the above lemma. The temporal expansion can be used with the appropriate error bounds within its validity period. The time-step τ will be referred to as the *validity duration* of the temporal expansion.

The validity period has been chosen based on position and position derivatives. This is because, conventionally, position is thought of as the independent variable and errors are measured with respect to position and its derivatives. Force or potential derivatives can also be used to determine time-steps. Since, $\frac{d^2\boldsymbol{x}}{dt^2}$ is equal to $\nabla\phi(x)$, this is meaningful and equally effective in *the absence of close encounters.* This choice is used in Aarseth's Individual Time-step scheme discussed in section 2.4.1. Although Aarseth's scheme is widely used, this is the first rigorous justification of its fidelity to a physical model using a finite Taylor series. (See discussion in section 1.1).

With close encounters, however, time-steps based only on potential/force derivatives (i.e. ignoring the first derivative of position since only the second and higher derivatives are thus accounted for) can on occasion lead to inaccuracies. Appendix A illustrates such a situation. The main problem is that the particle velocity is not bounded and can lead to anomalous situations. Moreover, higher time derivatives of potential can be non-zero even if only q derivatives of x or ϕ are used (since the potential is an implicit function of x). This makes it difficult to enforce the inequalities (3.2).

3.1.2 Temporal Expansion for Collection of Particles

The temporal expansion due to a collection of particles is just the sum of individual particle temporal expansions. We should require the validity period of this expansion to be the minimum of the validity periods of the constituent particles.

However, this is a very stringent requirement. Although the potential due to a distant cluster of particles might change negligibly at a point, it will have to be re-evaluated frequently. Instead, we will give a statistical interpretation to the far potential due to a cluster of particles.

In other words, the temporal expansion of a collection of particles has a validity and meaning beyond that of the particles of which it is comprised. It statistically averages out random fluctuations of the particle expansions and remains an accurate representation of the potential due to the collection of particles. In practice, this interpretation has been used before in N-body codes ([AC73,MA93]) with excellent results. Considering that N-body simulation itself has only a "statistical" validity, this is a reasonable interpretation.

Thus, we make the following assumption.

Assumption 3.1.3 (Statistical Averaging) *The temporal expansion of a spatially proximate collection of particles can be used with the appropriate error bounds within an empirically determined validity period even though individual particle expansions may not be valid.*

Derivatives of the potential (rather than the positions) are used to determine the time-step in the theorem below. This is not theoretically justified as we have already discussed but has been empirically found to be very satisfactory and has been in use for almost three decades.

Theorem 3.1.4 (Temporal Expansion) *Suppose that the k'th of s charges has strength m_k, space-time coordinates $\langle \boldsymbol{x}_k, t_k\rangle$, with $x_k > R$, and temporal expansion $\Phi_k(\langle \mathbf{o}, t\rangle)$ at the space-time point $\langle \mathbf{o}, t\rangle$, with corresponding coefficients b_k (3.1) and error-bound (3.3), where $k = 1, \ldots, s$.*
Then, at time t, the potential $\Phi(\langle \mathbf{o}, t\rangle)$ induced by the charges is given by

$$\Phi(\langle \mathbf{o}, t\rangle) = \sum_{j=0}^{\infty} a_j t^j \qquad \text{where} \qquad a_j = \sum_{k=1}^{s} m_k\, b_{k,j} \tag{3.15}$$

In the above, $t \in [T_{\Phi,L}, T_{\Phi,H}]$, where $[T_{\Phi,L}, T_{\Phi,H}]$, the **validity period** *of the expansion, is defined by*

$$[T_{\Phi,L}, T_{\Phi,H}] = [\max_{k=1,\ldots,s} t_k - \tau_\Phi, \min_{k=1,\ldots,s} t_k + \tau_\Phi] \tag{3.16}$$

with τ_Φ chosen such that

$$\tau_\Phi^i \left| \frac{d^i \Phi(\langle \mathbf{o}, t\rangle)}{dt^i} \right| \le \eta^i \left| \Phi(\langle \mathbf{o}, t\rangle) \right|, \qquad i \ge 1; \qquad \eta \in [0, \frac{\log 2}{2}] \tag{3.17}$$

Furthermore, under reasonable assumptions, for the d-dimensional potential function ϕ and any $q \geq 1$,

$$\left|\Phi(\langle \mathbf{o}, t\rangle) - \sum_{j=0}^{q} a_j t^j\right| \quad \leq \quad |\phi(R)|^{d-2} \left(\frac{\eta}{\log 2}\right)^{q+1} A \qquad (3.18)$$

where $A \; = \; \sum_{k=1}^{s} m_k$ *in the worst case, and* $A \; = \; \sqrt{\sum_{k=1}^{s} m_k^2}$ *in the expected case.*

Proof. The form of the temporal expansion (3.15) is an immediate consequence of the preceding lemma and the fact that $\Phi(\mathbf{o}, t) \; = \; \sum_{i=1}^{s} \Phi_k(\langle \mathbf{o}, t\rangle)$.
For the error bound, note that since R is a lower bound for x_k, $\phi(R)$ must be an upper bound for $\phi(x_k)$. Using this fact, (3.18) is obtained as the sum of the errors due to the individual particles in the worst case. For the expected case, we simply treat individual particle errors as normally distributed random variables. □

Note that, the error bounds in the above proof are strictly valid only in the time range when individual particle temporal expansions are valid. However, we are using the expansion even when individual particle expansions themselves are no longer valid. Therefore, although theorem 3.1.4 is motivated by lemma 3.1.1, it does not follow automatically and is valid only under assumption 3.1.3. Another way to look at it is that, by corollary 3.1.2, the error is small after a few iterations. The evolution of the potential at $\langle \mathbf{o}, t\rangle$ in this time range is a good indicator of how the potential will vary at future times. Therefore, although the validity period determined by the potential derivatives is not strictly valid, for non-pathological particle distributions, the error will remain small. If we wish to be conservative, we can force the temporal expansion to have a validity period which is the intersection of the validity periods of its constituent particles.

3.2 Temporal Multipole Expansions

If we think of the potential function as being specified by the multipole expansion, the temporal expansion of this potential is the *temporal multipole expansion.* The gradient of this potential expansion is the force. All the error bounds and operations on multipole expansions are automatically valid on temporal multipole expansions because differentiation is a linear operation and the temporal expansion is just the summation of time derivatives of the spatial multipole expansion.

Theorem 3.2.1 *The functions* Translate, Shift *and* Convert *can be used to manipulate temporal multipole expansions, with error bounds identical to those for multipole expansions (as in [Gre87,Zha87]).*

The error in the actual potential is now the sum of the individual errors due to the multipole expansion and the temporal expansion. The number of terms p in the multipole expansion and q in the temporal expansion are chosen so that the two errors are of the same magnitude. The resulting (p, q) term expansion will be referred to as a temporal multipole expansion. Note that, the temporal expansion is not entirely symmetric in the four space-time coordinates since the three space coordinates of a particle depend on the time coordinate.

3.2.1 Temporal Multipole Expansions for Cells

We will use temporal multipole expansions to combine the fast multipole algorithm and the individual time-step scheme. To that end, we discuss temporal multipole expansions for cells.

For leaf cells, temporal multipole expansions are simply formed as the sum of the temporal multipole expansions for individual particles. For lower level cells, temporal multipole expansions are formed recursively as in (2.5),(2.6) and (2.7). At the same time that the expansions are formed, a validity period is also associated with them beyond which they should not be used. The validity periods $[T_{\Psi_i,L}, T_{\Psi_i,H}]$, for Ψ_i, the far potential at cell i due to outside particles, is calculated directly using (3.16). The validity period $[T_{\Phi_i,L}, T_{\Phi_i,H}]$, for Φ_i, the far potential due to particles inside cell i, is calculated in the manner of (3.16) by taking the minimum validity period computed at each cell j's center such that i is in IList(j). Specifically,

$$[T_{\Phi_i,L}, T_{\Phi_i,H}] = \min_{\{j : i \in \mathrm{IList}(j)\}} \{[T_{\Phi,L}, T_{\Phi,H}] \text{ calculated with center of } j \text{ as origin in (3.16)}\} \tag{3.19}$$

In the next chapter, we use the machinery created here, to develop the sequential temporal multipole algorithm.

Chapter 4

Sequential Algorithm

We first describe a sequential temporal multipole algorithm. The algorithm uses temporal multipoles to combine the fast multipole algorithm and the individual time-step algorithm. We show that this algorithm is the best possible under reasonable assumptions.

4.1 Informal Description of Algorithm

The algorithm combines the space optimization of the fast multipole algorithm and the time optimization of the individual time-step scheme. The framework is the same as in the individual time-step scheme. Each particle has an update time up to which it can be integrated using its old Taylor series. The particle is updated by recalculating the force on it and finding a new Taylor series expansion. Near forces are directly calculated and far forces are evaluated using temporal multipole expansions for efficiency.

The essential new idea in the algorithm is to be "lazy" or to do the minimum amount of work necessary. A multipole expansion is recomputed only when its validity period has expired. When recomputing the expansion, another expansion might need to be used whose validity period has also expired, in which case that expansion is recursively recomputed and so on. We will refer to the process of checking the validity of an expansion and recomputing it if necessary as *validation*.

The algorithm is described below in more detail. We use the same notation used in the description of the fast multipole algorithm and the individual time step scheme in chapter 2. Φ and Ψ are now, however, (p,q)-term temporal multipole expansions.

1. Particle a, with the earliest update time, is identified. Let cell i contain a and let update time for a be t_a.

2. All particles in all the neighbors of i (including itself) are integrated to time t_a using their old Taylor series expansions.

3. The near force on a is found as the sum of the forces exerted on a by all the particles in i and its neighbors.

4. The far force on a must now be calculated. If Ψ_i, the far potential at cell i, is not valid at time t_a, it has to be recalculated. This is done by recursively validating the Ψ for the parent of cell i. Then, Φ, the far potential due to particles in the cell, for each cell in the interaction list of cell i, is validated (also recursively). Finally, Ψ_i is formed using the Ψ of the parent cell and the Φ of the IList cells, and evaluated at the position of particle a to find the far force on particle a.

5. The sum of the near and far forces is the total force on particle a. This force is used to form a new Taylor series expansion. The expansion is used to determine a new update time for particle a.

6. The above procedure is repeated beginning from step 1, until all particles have been advanced through the required simulation time.

4.2 Transient Particle Consistency

The far potential at a cell could be computed from temporal potential expansions of two cells calculated at different times. A particle transiting between such cells, could contribute twice or not at all to the computed potential. This transient potential error is due to the *discreteness* of particles: a particle is either in the cell or not in the cell and no intermediate state is possible. See Figure 4.1. This error is small in magnitude since it is a "surface" effect while most of the particles are in the "volume". Simulations ignoring these effects by Aarseth-McMillan [MA93] have yielded satisfactory results.

One simple way of correcting such errors is to update a cell's temporal expansion whenever a particle crosses its boundary. But since cells at many levels share the same boundary, this is expensive and also difficult to parallelize. We discuss below a different method of correcting such errors, where the idea is to maintain a consistent interpretation of location of particles across cell boundaries. For these corrections to work, we are forced to sacrifice the complete asynchrony (i.e. never calculating an expansion unless its validity period has expired) in the algorithm described in the previous section. The algorithm is modified so that whenever Φ is computed for a cell, Φ for all its children cells is also recomputed. We need an assumption on particle movement here.

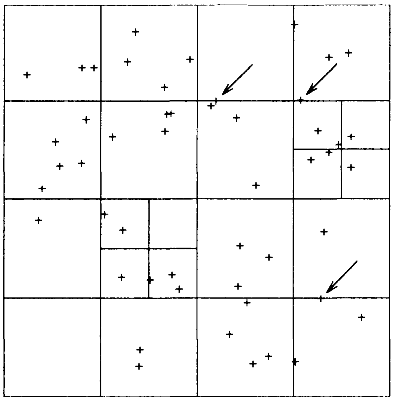


Figure 4.1: Particle Transitions across cell boundaries
Transiting particles are indicated by arrows

Assumption 4.2.1 (Particle Movements) *Within the validity period of a cell's temporal expansion, a particle can only move from that cell to some adjacent cell.*

The validity-period of a cell expansion is such that this will be true in general.

4.2.1 Particle Consistency Operator

To account for particles crossing cell boundaries, we introduce the *particle consistency operator*, χ. It makes a cell expansion formed at an earlier time consistent with another formed at a later time with respect to particles crossing the cell boundary, so that, each particle contributes exactly once to the potential at any cell. We develop the concept more formally below.

Let the temporal expansions for cell i be $\Phi_{\langle i,t_i\rangle}$ where the time-coordinate t_i denotes the time of creation of the corresponding expansion. For adjacent cells i and j, we say that temporal expansions $\Phi_{\langle i,t_i\rangle}$ and $\Phi_{\langle j,t_j\rangle}$ are *consistent*, denoted $\Phi_{\langle i,t_i\rangle} \sim \Phi_{\langle j,t_j\rangle}$, if they have the same interpretation of which particle is on which side of the boundary between i and j. In what follows, we use the convention that two temporal expansions agreeing on the time coordinate of their creation are consistent. For instance, $\Phi_{\langle i,t\rangle}$ is consistent with $\Phi_{\langle j,t\rangle}$. The particle consistency operator makes expansions created earlier in time consistent with those created later in time. More technically, particle consistency operator $\chi_{\langle j,t_j\rangle,\langle i,t_i\rangle}$ operates on $\Phi_{\langle j,t_j\rangle}$ to make it consistent with $\Phi_{\langle i,t_i\rangle}$, where t_j is assumed to be smaller than

t_i.

In practice, $\chi_{\langle j,t_j\rangle,\langle i,t_i\rangle}$ represents corrections due to particles moving from cell i to cell j in the time interval $[t_j, t_i]$. A natural choice for its representation is as a temporal expansion about the center of cell j. Then $\chi_{\langle j,t_j\rangle,\langle i,t_i\rangle}$ is simply the sum of temporal expansions about the center of cell j of particles moving from cell i to cell j in time range $[t_j, t_i]$.

The particle consistency operator has the following properties:

$$\text{Consistency:} \quad \chi_{\langle j,t_j\rangle,\langle i,t_i\rangle} + \Phi_{\langle j,t_j\rangle} \sim \Phi_{\langle i,t_i\rangle} \tag{4.1}$$

$$\text{Exclusivity:} \quad \text{Either } \chi_{\langle j,t_j\rangle,\langle i,t_i\rangle} \not\equiv 0 \text{ or } \chi_{\langle i,t_i\rangle,\langle j,t_j\rangle} \not\equiv 0 \text{ but not both.} \tag{4.2}$$

$$\text{Space Composition:} \quad \chi_{\langle j,t_j\rangle,\langle i,t_i\rangle} = \sum_{\substack{k\in\text{Children}(j)\\ l\in\text{Children}(i)}} \text{Translate}_{k,j}(\chi_{\langle k,t_j\rangle,\langle l,t_i\rangle}) \tag{4.3}$$

$$\text{Time Composition:} \quad \chi_{\langle j,t_j\rangle,\langle i,t\rangle} = \chi_{\langle j,t_j\rangle,\langle i,t_i\rangle} + \chi_{\langle j,t_i\rangle,\langle i,t\rangle} \tag{4.4}$$

The consistency property is just the definition. Exclusivity follows from the fact that the operator only makes expansions created earlier in time consistent with those created later in time and not vice versa. The space composition property is due to our representation of the consistency operator for a cell as a temporal expansion about its center. Time composition corresponds to breaking up a time interval $[t_j, t]$ into two parts, $[t_j, t_i]$ and $[t_i, t]$. Particles transiting between i and j must have done so in one of the two sub-intervals, and the corresponding consistency terms can be added up.

4.2.2 Manipulation of Consistency Operators

Consistency operator $\chi_{\langle j,t_j\rangle,\langle i,t_i\rangle}$ is needed for any two cells i and j where $\Phi_{\langle i,t_i\rangle}$ and $\Phi_{\langle j,t_j\rangle}$ combine to form another expansion. Thus, i and j could be neighbors at the same level with i and j being members of IList(k). In this case, $\Phi_{\langle i,t_i\rangle}$ and $\Phi_{\langle j,t_j\rangle}$ combine to form $\Psi_{\langle k,t_k\rangle}$. Alternatively, i and j could be cells at consecutive levels with i in IList(k) and j in IList(Parent(k)). In this case, $\Phi_{\langle i,t_i\rangle}$ combines with $\Phi_{\langle j,t_j\rangle}$ (indirectly through $\Psi_{\langle \text{Parent}(k),t_{\text{Parent}(k)}\rangle}$) to form $\Psi_{\langle k,t_k\rangle}$.

Each cell i maintains $\chi_{\langle j,t_j\rangle,\langle i,t_i\rangle}$ and $\chi_{\langle j,t_{\text{Parent}(i)}\rangle,\langle i,t_i\rangle}$ for each neighbor j of i. For bottom level cells consistency terms are directly formed from the definition. For higher level cells, they are formed from the childrens' consistency terms. The following lemma shows how particle consistency terms can be maintained as new cell temporal expansions are formed. For simplicity, we show only the lemma for two cells at same or adjacent levels. Extension to cases when their levels differ by more than one is straightforward.

Lemma 4.2.2 *Consider adjacent cells i and j, at levels L_i and L_j with $|L_i - L_j| \leq 1$, with temporal expansions $\Phi_{\langle i,t_i \rangle}$ and $\Phi_{\langle j,t_j \rangle}$ and consistency terms $\chi_{\langle j,t_j \rangle, \langle i,t_i \rangle}$ and $\chi_{\langle j,t_{\mathrm{Parent}(i)} \rangle, \langle i,t_i \rangle}$. Let Φ be reformed for cell i at time t. Then the consistency terms are updated as follows:*

$$\chi_{\langle j,t_{\mathrm{Parent}(i)} \rangle, \langle i,t \rangle} = \chi_{\langle j,t_{\mathrm{Parent}(i)} \rangle, \langle i,t_i \rangle} + \chi_{\langle j,t_i \rangle, \langle i,t \rangle} \tag{4.5}$$

$$\chi_{\langle j,t_j \rangle, \langle i,t \rangle} = \chi_{\langle j,t_j \rangle, \langle i,t_i \rangle} + \chi_{\langle j,t_i \rangle, \langle i,t \rangle} \tag{4.6}$$

where $\chi_{\langle j,t_i \rangle, \langle i,t \rangle}$ is given by

$$L_i = L_j : \quad \chi_{\langle j,t_j \rangle, \langle i,t \rangle} = \sum_{\substack{k \in \mathrm{Children}(j) \\ l \in \mathrm{Children}(i)}} \mathrm{Translate}_{k,j}(\chi_{\langle k,t_j \rangle, \langle l,t \rangle}) \tag{4.7}$$

$$L_i < L_j : \quad \chi_{\langle j,t_j \rangle, \langle i,t \rangle} = \sum_{k \in \mathrm{Neighbor}(j) \cap \mathrm{Children}(i)} \chi_{\langle j,t_j \rangle, \langle k,t \rangle} \tag{4.8}$$

$$L_i > L_j : \quad \chi_{\langle j,t_j \rangle, \langle i,t \rangle} = \sum_{k \in \mathrm{Children}(j) \cap \mathrm{Neighbor}(i)} \mathrm{Translate}_{k,j}(\chi_{\langle k,t_j \rangle, \langle i,t \rangle}) \tag{4.9}$$

Proof. (4.5) and (4.6) follow from the time composition property. The other equations are simple consequences of the following observations. First, if particles move from cell i to j, they must also have moved from children of cell i to children of cell j. Secondly, the consistency term for a cell is a temporal expansion expressed about the center of the cell.

For instance, if $L_i = L_j$, consistency terms from children of i to children of j have to be translated to j's center and summed. The other cases are similar. □

4.2.3 Compensating for Transient Particle Potentials

The following lemma shows how to use consistency operators to make particle transitions consistent across a cell boundary. See Figure 4.2.

Lemma 4.2.3 (Correction) *Let cell k be a member of the interaction list of cell i. Then the contribution of cell k to the far field potential at cell i at time t_i is given by*

$$\widehat{\Psi}_{\langle i,t_k \rangle} = \mathrm{Convert}_{k,i} \left(\Phi_{\langle k,t_k \rangle} + \sum_{j \in L(k)} \chi_{\langle k,t_k \rangle, \langle j,t_j \rangle} \right) \tag{4.10}$$

where $L = L1 \cup L2 \cup L3$ *and* $L1 = \mathrm{IList}(i) \cap \mathrm{Neighbor}(k)$ *and* $L2 = \mathrm{Children}(\mathrm{Neighbor}(i) \cap \mathrm{Neighbor}(k))$ *and* $L3 = \mathrm{Parent}(\mathrm{Neighbor}(k) - (\mathrm{IList}(i) \cup \mathrm{Neighbor}(i)))$.

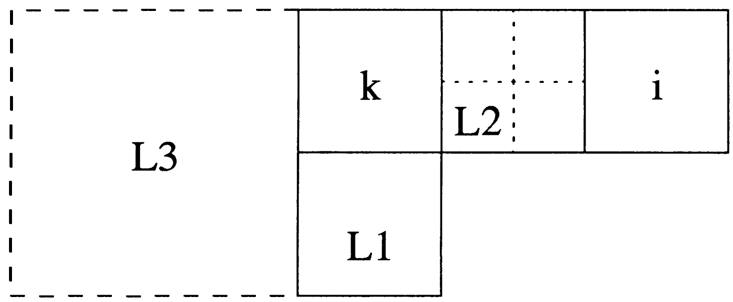


Figure 4.2: Correcting Ψ with Consistency Operators

Proof. Any particles which have transited to cell k after $\Phi_{\langle k,t_k\rangle}$ was formed could have come only from cells adjacent to k. We wish to make the expansion of cell k consistent with respect to the other expansions which will form Ψ_i. These expansions correspond exactly to cells in list L. $L1$ consists of cells of the same size as k, $L2$ consists of cells smaller than k and $L3$ consists of cells larger than k. In each case, particle consistency is achieved by adding in the appropriate particle consistency term. Since $\chi_{\langle k,t_k\rangle,\langle j,t_j\rangle}$ is about the center of cell k in each case, the terms can be added in before the "Convert" operation. □

For the adaptive algorithm, consistency corrections can be applied in a similar manner although the details are more complicated.

4.3 Temporal Multipole Algorithm with Particle Consistency

We describe the temporal algorithm more formally now and incorporate the particle consistency operators in the description. We drop the time subscripts from the expansions for simplicity. Thus, $\Phi_{\langle i,t_i\rangle}$ is denoted by Φ_i and $\chi_{\langle i,t_i\rangle,\langle j,t_j\rangle}$ by $\chi_{i,j}$.

Algorithm

Initialization

Choose refinement level n, a precision ϵ, and corresponding $p,q \approx \lceil -\log_2 \epsilon \rceil$. (Choice of q also depends on efficiency parameter η).
Form (p,q)-term temporal-multipole expansions for all cells at all levels.
Set update time t_b for each particle b.

Update Particle

Find particle a such that $t_a = \min_{b=1,\dots,N} t_b$. Let cell i contain particle a.

Find Near Force:

Integrate each particle in Neighbor(i) to time t_a using its old Taylor-series expansion.

Find near force on a by directly interacting it with all particles in Neighbor(i).

Find Far Force:

Let $U = \{$ cell j $\mid$ Ψ_j is required to form Ψ_i but $t_a \notin [T_{\Psi_j,L}, T_{\Psi_j,H}]\} \cup \{i\}$

Let $V = \{$ cell j $\mid$ Φ_j is required to form Ψ_k with $k \in U$ but $t_a \notin [T_{\Phi_j,L}, T_{\Phi_j,H}]\}$

For each j in V, recompute Φ_j and $\chi_{j,l}$ moving bottom up in the tree using (2.5); compute $[T_{\Phi_j,L}, T_{\Phi_j,H}]$ using (3.19).

For each j in U, recompute Ψ_j moving top down in the tree using (2.6) and lemma 4.2.3; compute $[T_{\Psi_j,L}, T_{\Psi_j,H}]$ using (3.16).

Evaluate Ψ_i at position of particle a to get far force on a.

Set New Time-step:

Find total force on a and find new Taylor-series representation for it.

Use this series to predict new time-step τ_a using lemma 3.1.1 and set new update time for particle a to $t_a + \tau_a$.

Go to Update Particle.

End of Algorithm

Remark *Boundary Conditions:* Dirichlet, periodic or other boundary conditions can be imposed on the above free space description of the algorithm exactly as discussed by Greengard [Gre87] but using temporal multipoles instead of just multipoles.

Remark *Convergence Issues:* Convergence problems can occur for a particle near a cell boundary, since, its spatial multipole expansion, and therefore the temporal expansion, is not guaranteed to converge once the particle crosses the boundary. In practice, this is not significant because it happens infrequently and also self-corrects. A particle near the boundary causes the temporal expansion to change more rapidly, reducing its validity period.

4.4 Complexity Analysis and Optimality

The space complexity of the algorithm is easily seen to be $O(N)$, just as in the fast multipole algorithm. The number of cells is linear in N, and $O(1)$ space is required per cell and per particle. We briefly describe how to analyze the time complexity of the algorithm and show that under reasonable assumptions, it is optimal.

The time taken by the algorithm is intimately related to the number of integration steps through which particles are advanced. Let g be the number of integration steps required to advance all the particles for a standard length of physical time T_{phys}. g is inversely proportional to the integration time-step. The number of integration steps required for particle a with time-step $\tau_a(t)$ at time t, is

$$g_a = \int_0^{T_{\text{phys}}} \frac{dt}{\tau_a(t)} \tag{4.11}$$

Then

$$g = \sum_{a=1}^{N} g_a = \sum_{a=1}^{N} \int_0^{\infty} \frac{p[\tau_a]}{\tau_a} d\tau_a \tag{4.12}$$

where $p[\tau_a]d\tau_a$ is the probability that the time-step lies in the range $[\tau_a, \tau_a + d\tau_a]$

We make the following reasonable assumption about any N-body algorithm.

Assumption 4.4.1 (Minimum Computation) *Any N-body algorithm requires a number of computational steps proportional to g to integrate an N-body system, where g is the number of time-steps required to integrate all particles in the system over the desired amount of physical time.*

The assumption is justifiable since our objective is fidelity of the computational model to the physical model. The results of Reif and Tate [RT93], indicate that there are no shortcuts for the process of integrating all particles through sufficiently small time-steps using sufficiently long Taylor series approximations.

To analyze the computational complexity of the algorithm, we make one more assumption, which is almost the same as the statistical averaging assumption 3.1.3, but more specific.

Assumption 4.4.2 (Validity period for parent cell expansion) *The temporal expansion for a parent cell is formed at most α times ($\alpha < 1$) as often as the temporal expansion is formed for its child cell(s).*

The justification for the assumption is the statistical averaging which occurs. The potential due to a larger number of particles changes less rapidly and small fluctuations due to individual particle motions cancel out. A second reason is that higher levels in the tree correspond to far potentials from particles which are further away. Therefore, the rate of change of this potential is smaller. We will explore this argument in more detail in theorem 5.3.1 in the next chapter.

We will make an additional assumption on the particle distribution, whereby, particles are assumed to be distributed in a manner such that their number density does not change drastically from a cell to its neighbor. This assumption is not necessary for the complexity analysis of the algorithm, but simplifies the proof considerably.

Assumption 4.4.3 (Non-exponential Density) *The change in particle density with distance is sub-exponential in the distance i.e. $\rho(r)$ is a sub-exponential function of r.*

The implication of the assumption is that in the adaptive algorithm, the neighbors of a leaf cell are either at the same level, or one level higher or one level lower. This will almost always be true for particle distributions in practice.

Theorem 4.4.4 (Optimality) *The temporal multipole algorithm is optimal to within a constant factor, under reasonable assumptions.*

Proof. We show that the time complexity of the temporal multipole algorithm is $O(g)$. In the next section, we will show that selection of the next particle to integrate takes only $O(1)$ time. The time-complexity is the sum of the times taken for selecting and updating particles, the time for calculating the near forces and the time to calculate the far forces. The first two are simply $O(g)$ since only $O(1)$ time is required per particle integration.

$$\text{Total Cost } = O(g) + C_\Phi + C_\Psi + C_\chi \tag{4.13}$$

Here, C_ϕ (likewise, C_Ψ and C_χ) is the cost of forming Φ (likewise, Ψ and χ) for all cells for the entire simulation. C_Φ can be broken up into $C_{\Phi,\text{leaf}}$ and $C_{\Phi,\text{non-leaf}}$ which are the respective costs for leaf cells and non-leaf cells respectively. $C_{\Phi,\text{leaf}}$ is at most $O(g)$ since Φ for a cell is formed less frequently than the particles in it are moved. Using assumption 4.4.2, Φ for a parent cell is formed at most α times as often as its children. Moreover, the number of parent cells with leaf cells as children is at most half the number of leaf cells.

$$\begin{aligned} C_\Phi &= C_{\Phi,\text{leaf}} + \alpha C_{\Phi,\text{leaf}}(1+\frac{1}{2}) + \alpha^2 C_{\Phi,\text{leaf}}(1+\frac{1}{2}+\frac{1}{4}) + \dots \\ &\le 2C_{\Phi,\text{leaf}}(1+\alpha+\alpha^2+\dots) \\ &\le \frac{2}{1-\alpha} C_{\Phi,\text{leaf}} \\ &= O(g) \end{aligned} \tag{4.14}$$

Using the non-exponential density assumption, a similar argument shows that C_Ψ is also $O(g)$. The cost C_χ can be broken up into two parts: cost for creation of χ and cost for use of χ. Now, χ is only created for a cell when Φ is and the number of χ terms is a constant for each cell. Thus the creation cost of χ is $O(C_\Phi)$. Use of χ occurs only when Ψ is created and therefore the cost of the use of χ can similarly be accounted as $O(C_\Psi)$. Therefore, C_χ is $O(C_\Psi + C_\Phi)$. Putting all this together,

$$\text{Total cost } = g + C_\Phi + C_\Psi + C_\chi \le O(g+g+g+g) = O(g) \tag{4.15}$$

□

See Table 4.1 for a comparison of the time complexities of various algorithms for integration of N particles in three dimensions over one crossing time. Complexities are given for homogeneous distributions and power law distributions. Power law distributions are spherically symmetric about the origin and have a particle density ρ given by $\rho(r) \propto r^{-\alpha}$ at radius r. Particle velocities are isotropic and given by $v(r) \propto r^{(2-\alpha)/2}$. Derivations and details are available in Makino and Hut [MH88]. Asymptotically, the temporal multipole algorithm is significantly better. For N about 10^4, gains are visible in practice despite the larger constant factor.

4.5 Blocking of Time-steps

In the optimality proof above, we ignored the time required for selecting the next particle to be integrated. If a priority queue or heap is naively used, this complexity can be as large as $O(\log N)$. We show below how this selection can be done in $O(1)$ time.

The idea is to *block* the time-steps ([McM86, MA93]). Time-steps are constrained to be multiples of a power of 2 with the characteristic smallest time-step for the system. If the predicted time-step is τ and the characteristic time-step is τ_0, then the new time-step is chosen to be $2^k\tau_0$ where $2^k\tau_0 \leq \tau \leq 2^{k+1}\tau_0$. The k'th block, consisting of particles with time-step 2^k, is advanced periodically at time intervals of $2^k\tau_0$.

4.5.1 Time-step Evolution

When a particle's predicted time-step moves it into a different block, its next update must occur before the time-step expires. The next lemma shows that the new time-step for a particle is at least half the previous time-step.

Lemma 4.5.1 (Time-step Evolution) *Consider a particle being integrated in accordance with lemma 3.1.1, with time-step τ at time t_0. At time $t_0 + \tau$, its time-step τ' is at least as large as $\frac{\tau}{2}$ if η is less than* $\log\frac{4}{3}$.

Proof. The time-step is determined from (3.2). First, we only consider the constraint on the time-step due to the first derivative of x. Note that

$$\frac{\frac{dx}{dt}}{x} = \frac{x^{(1)}}{x} = \frac{d}{dt}\log x(t) \tag{4.16}$$

where $x^{(i)}$denotes $\frac{d^ix}{dt^i}$, the i'th derivative with respect to time of x. So we can just consider the derivative of the function $\log x$. Within the time range of interest, bounds on $x(t)$ give corresponding bound for $\log x(t)$. We start with (3.7), in which the lower bound has all Taylor series terms with a negative sign while the upper bound has all terms with a positive sign.

$$
\begin{aligned}
(2 - e^{\eta(t-t_0)/\tau})x(t_0) &\leq \quad x(t) \quad \leq e^{\eta(t-t_0)/\tau}x(t_0) \\
\log(2 - e^{\eta(t-t_0)/\tau}) + \log x(t_0) &\leq \quad \log x(t) \quad \leq \log e^{\eta(t-t_0)/\tau} + \log x(t_0)
\end{aligned} \tag{4.17}
$$

If all the Taylor series terms are taken to be of the same sign in each bound on $x(t)$, as we have done, their derivatives bound the derivative of $\log x(t)$, as well.

$$
\begin{aligned}
\frac{d}{dt}\log(2 - e^{\eta(t-t_0)/\tau}) &\leq \quad \tfrac{d}{dt}\log x(t) \quad \leq \frac{d}{dt}\log e^{\eta(t-t_0)/\tau} \\
-\frac{e^{\eta(t-t_0)/\tau}}{2 - e^{\eta(t-t_0)/\tau}}\frac{\eta}{\tau} &\leq \quad \tfrac{d}{dt}\log x(t) \quad \leq \frac{\eta}{\tau}
\end{aligned} \tag{4.18}
$$

At time $t_0 + \tau$, the bound is given by

$$
-\frac{e^{\eta}}{2 - e^{\eta}}\frac{\eta}{\tau} \leq \frac{x^{(1)}(t_0+\tau)}{x(t_0+\tau)} \leq \frac{\eta}{\tau} \tag{4.19}
$$

Therefore the new time-step τ' as determined using only the first derivative of x in (3.2) is bounded by

$$
\frac{\eta}{\tau'} \leq \frac{e^{\eta}}{2 - e^{\eta}}\frac{\eta}{\tau} \Rightarrow \tau' \geq \frac{1}{2}\tau \tag{4.20}
$$

where in the last step we use the fact that $\eta < \log\frac{4}{3}$ whence $e^{\eta} < \frac{4}{3}$.

For constraints on the time-step due to higher derivatives, we can assume that the time-step is tightly constrained by every derivative, since, this would maximize the change in the time-step. But in this case, we can bound $\log(x^{(i)}/x^{(i-1)})$ as above.

$$
\begin{aligned}
\frac{x^{(i)}}{x} &= \frac{x^{(i)}}{x^{(i-1)}}\frac{x^{(i-1)}}{x^{(i-2)}}\cdots\frac{x^{(1)}}{x} \\
\Rightarrow \quad \left|\frac{x^{(i)}}{x}\right| &\leq \left|\frac{x^{(i)}}{x^{(i-1)}}\right|\left|\frac{x^{(i-1)}}{x^{(i-2)}}\right|\cdots\left|\frac{x^{(1)}}{x}\right| \\
\Rightarrow \quad \left|\frac{x^{(i)}}{x}\right| &\leq \left(\frac{e^{\eta}}{2 - e^{\eta}}\frac{\eta}{\tau}\right)^{i} \\
\Rightarrow \quad \frac{\eta}{\tau'} &\leq \frac{e^{\eta}}{2 - e^{\eta}}\frac{\eta}{\tau}
\end{aligned} \tag{4.21}
$$

which gives exactly the same bound on τ'. □

To maintain fidelity to the physical model and for stability reasons, the time-step is usually not allowed to increase by a factor of more than 2 from one time-step to the next. (Aarseth [Aar85] recommends a factor of at most $\sqrt{2}$). The implication is that after a particle is integrated, either it stays in the same block or moves one block higher or one block lower. Therefore, moving a particle to the appropriate block is an $O(1)$ operation.

4.5.2 Block Updates and Efficiency

The periodic update scheme in Sundaram [Sun92] (also see [McM86,MA93]) extends to the idea of block updates. At the jth step, particles in all blocks up to and including the r'th block are updated, where r is the maximum power of 2 which divides j. This automatically ensures the required periodicity of 2^k for the k'th block.

Blocking of time-steps also turns out to be more efficient. On the positive side, a large number of particles are advanced together. Some intermediate predictions of particle positions become unnecessary and temporal expansions need to be validated only once for an entire block of particles. Cells can also have a block time-step making recalculation of temporal expansions less frequent. On the negative side, since, the blocked time-step can at worst be half as large as the predicted time-step, the number of integration steps can double. Overall, the gains from blocking more than compensate for this increased number of integration steps.

Table 4.1: Time Complexity for integrating N particles over 1 crossing time in 3D

Distribution → Algorithm ↓	Homogeneous Complexity[b]	Power Law[a] Complexity[c]	
Number of steps (g)	$\frac{4}{3}$	$\frac{\alpha}{6-2\alpha}$	$\frac{24}{11} < \alpha < 3$
Direct Shared	$\frac{8}{3}$	$\frac{12-3\alpha}{6-2\alpha}$	$2 \leq \alpha < 3$
Direct Individual	$\frac{7}{3}$	$\frac{6-\alpha}{6-2\alpha}$	$\frac{24}{11} < \alpha < 3$
Ahmad-Cohen	$\frac{25}{12}$	$\frac{42-13\alpha}{24-8\alpha}$ $\frac{1}{\alpha-4} + \frac{6-\alpha}{6-2\alpha}$	$\frac{24}{11} < \alpha < \frac{31-\sqrt{97}}{9}$ $\frac{31-\sqrt{97}}{9} < \alpha < 3$
Fast Multipole	$\frac{5}{3}$	$\frac{6-\alpha}{6-2\alpha}$	$2 \leq \alpha < 3$
Temporal Multipole	$\frac{4}{3}$	$\frac{\alpha}{6-2\alpha}$	$\frac{24}{11} < \alpha < 3$

[a]Spherically symmetric density $\rho(r) \propto r^{-\alpha}$ at radius r; isotropic particle velocity $v(r) \propto r^{(2-\alpha)/2}$
[b]Exponent q of N in $O(N^q)$ is specified
[c]If not mentioned, for $2 \leq \alpha < \frac{24}{11}$, the power law complexity is the same as for homogeneous

Chapter 5

Parallel Algorithm: Homogeneous

Temporal multipole expansions can be used without loss of accuracy within their validity periods. This allows communication of multipoles to overlap computation. In this chapter, we examine how the temporal multipole algorithm can be parallelized for homogeneous particle distributions.

In parallelizing the algorithm, we have to address two issues: communication and load balancing. Communication between processors is a necessary evil: necessary for information exchange but inordinately expensive compared to computation. We will discuss how temporal multipoles can exploit the locality of the problem and minimize communication. For full efficiency, each processor should do approximately the same amount of work. For homogeneous distributions, this is automatically true. We address the issue of load balancing for non-homogeneous distributions in the next chapter. Since gravitational and Coulombic forces fall away rapidly with distance, they have a strong local bias. Our objective is to give local solutions which exploit this local bias.

5.1 Parallel Processing Model

We are interested only in MIMD models of computation consisting of a small network of powerful processors. Parallel computing in the next decade seems to be moving toward such machines, exemplified by the Intel PARAGON, the nCUBE, Thinking Machines CM-5, IBM Power Parallel, the Parsytec Supercluster and the new CRAY T3D.

The communications involved in the parallel algorithm correspond to the pyramid (explained later). We will therefore explain the concepts involved assuming processors interconnected in a pyramid architecture. Communication takes place by message-passing and sending or receiving a message will be assumed to take $O(1)$ time. In Chapter 7, we will show how to implement the algorithm on various parallel architectures.

5.2 Communication

In the sequential temporal multipole algorithm, the computation of multipoles is need-driven or demand-driven. Multipole expansions are computed only when some particle needs to use them. In the parallel setting, cells are distributed across processors and the communication time involved in making a request and receiving a response is too large. For efficiency, we make the parallel algorithm synchronous above the cell level and expansions are periodically computed and communicated.

5.3 Estimating the Validity Duration τ_Φ

The temporal expansion of one cell should be still usable when it reaches another. If the communication time from cell A to B is T, the time for which the expansion of cell A is valid at B should also be at least T. Below, we estimate $\tau_\Phi(D)$, the validity duration of a temporal expansion at distance D, for a homogeneous distribution. This estimate is subsequently used to show that the parallel temporal multipole algorithm is optimal.

Theorem 5.3.1 *Consider the potential at the origin due to particles contained inside a sphere of radius R with center at distance $(D+R)$ from the origin, where $D > R$. See Figure 5.1. Let the length of time, for which the temporal expansion of the particles inside the sphere is valid at the origin, be $\tau_\Phi(D)$. Then $\tau_\Phi(D)$ is at least $\frac{\eta D}{3v_0}$, where v_0 is the dispersion velocity*[1] *of particles inside the sphere.*

Proof. The length of time for which the temporal expansion is valid is determined using equation 3.17. Rearranging this equation,

$$\tau_\Phi \leq \eta \left| \frac{\Phi(\langle \mathbf{o}, t\rangle)}{\frac{d^i\Phi(\langle \mathbf{o}, t\rangle)}{dt^i}} \right|^{1/i} \qquad 1 \leq i \leq q \tag{5.1}$$

We wish to estimate the minimum possible value for $\tau_\Phi(D)$. The minimum value is constrained by the first derivative of the potential since we are only considering far potentials. Let us consider the potential function $\phi(x) = \frac{1}{x}$. The proof is similar

[1]Dispersion velocity is just the root mean square velocity i.e. $v_0 = \sqrt{\sum_{i=1}^{s} v_i^2/s}$ for s particles.

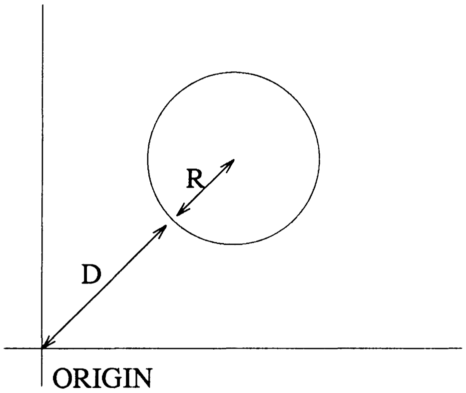


Figure 5.1: Validity duration of temporal expansion of particles inside sphere at origin

for $\phi(x) = -\log x$. Consider a unit charge in the sphere located at point $\boldsymbol{x}$. Its potential at the origin is given by $\phi(x)$ and the first derivative by:

$$\frac{d\phi(x)}{dt} = \frac{d\phi(x)}{dx}\frac{dx}{dt} = -\frac{1}{x^2}\frac{dx}{dt} \tag{5.2}$$

Let there be s particles inside the sphere with positions $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_s$ and charge intensities $m_1, m_2, \ldots, m_s$ such that $M = \sum_{i=1}^{s} m_i$. The minimum value for τ_Φ will be obtained when the numerator of (5.1) is minimized and the denominator is maximized. The numerator is

$$\eta\Phi(\langle \mathbf{o}, t\rangle) = \eta\sum_{i=1}^{s} m_i\phi(x_i) = \eta\sum_{i=1}^{s}\frac{m_i}{x_i} \geq \eta\frac{\sum_{i=1}^{s} m_i}{D+2R} = \frac{\eta M}{D+2R} \tag{5.3}$$

The denominator can be expressed as

$$\left|\frac{d\Phi(\langle \mathbf{o}, t\rangle)}{dt}\right| = \left|\sum_{i=1}^{s} m_i \frac{d\phi(x_i)}{dt}\right| = \left|\sum_{i=1}^{s}\frac{m_i}{x_i^2}\frac{dx_i}{dt}\right| \leq \frac{1}{D^2}\left|\sum_{i=1}^{s} m_i\frac{dx_i}{dt}\right| \tag{5.4}$$

We can approximate the summation by the worst case of all particles moving towards (or away from) the origin with a velocity equal to v_0, the dispersion velocity. This bulk flow is highly improbable but will give the most stringent bound on τ_Φ. Note that as the number of particles within the sphere s becomes larger, "bulk flow" becomes more and more improbable. At the other end, in the limit when s is just 1, "bulk flow" can occur with high probability.

$$\left|\frac{d\Phi(\langle \mathbf{o}, t\rangle)}{dt}\right| \leq \sum_{i=1}^{s}\frac{m_i v_0}{D^2} \leq \frac{M v_0}{D^2} \tag{5.5}$$

Using these values for the numerator and denominator of (5.1), τ_Φ is bounded by

$$\tau_\Phi(D) \geq \frac{\eta M}{D + 2R}\frac{D^2}{M v_0} \geq \frac{\eta D}{3 v_0} \tag{5.6}$$

where we used the fact that D is greater than R. □

Remark *Validity Duration for Steady-State Distributions:* For steady-state distributions like the Plummer law distribution, "bulk flow" is very highly improbable. For the more reasonable assumption that the velocities are normally distributed random variables, we get a lower bound where $\tau_\Phi(D) \propto DR^{3/2}$ when $\phi(x) = \frac{1}{x}$. This larger value of τ_Φ leads to a more efficient parallel algorithm on the mesh.

5.4 The Parallel Temporal Multipole Algorithm

For homogeneous particle distributions, all leaf level cells have the same size, contain roughly the same number of particles (say s) and these particles have time-steps of the same magnitude. For ease of exposition, we assume that each cell has one processor assigned to it, so that processors can be identified with cells. Further, we suppress details about force calculations within leaf cells.

Each cell needs to communicate with its children, its parent, its neighbors and its interaction list. Recalling that the interaction list of a cell consists of cells which are children of its parent's neighbors, it is sufficient for a cell to communicate with only its parent, children and neighbors. The communication structure is that of a pyramid of processors. Figure 5.2 shows the pyramid for a one dimensional system of particles.

The parallel algorithm just pipelines the temporal expansion communications. Communication and computation proceed concurrently. Cells internally calculate validity periods for expansions and if necessary wait for a valid expansion to arrive. However, by the validity duration estimate of theorem 5.3.1 temporal expansions almost always reach their destinations in time to be used. For simplicity, we will assume in the description below that temporal expansions always reach their destinations within their validity periods.

A formal pseudo-code description of the algorithm follows. The computations performed at a typical cell i at level h are detailed.

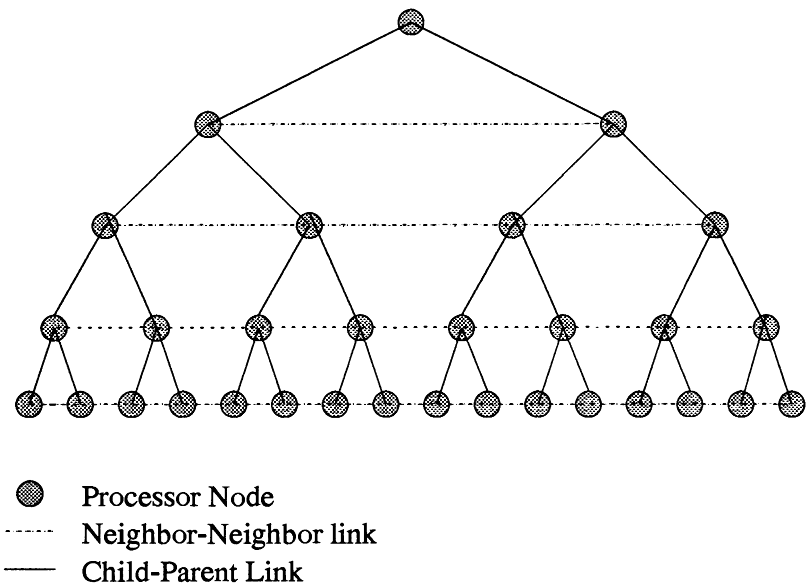


Figure 5.2: Pyramid of Processors for the 1D System of Figure 2.1

Initialization

Choose the number of levels $n \approx \log N$, desired precision ϵ, the order of the temporal expansion $p, q \approx -\log_2 \epsilon$, and number of particles per cell s.
Do one pass through the algorithm without pipelining (i.e. wait for all communication to complete) and initialize all cell values.

Choose Next Time-step at each cell at level n:

Select next time t_a such that $(t_a - t)$ is minimum and there is a particle a in the cell with update time t_a. Propose time t_a to neighbors.
Synchronize time with neighbors choosing least time proposed by all neighbors as current time t.
All particles within cell with update time t (possibly none) will be advanced.

Upward Pass

Comment [Form expansions Φ_i of potential field due to particles in cell i about the center of the cell.]

Cell i at level n:

Form (p, q)-term temporal multipole expansions $\Phi_i, \chi_{i,j}$ using some method within cell i.
Send expansions Φ_i and $\chi_{i,j}$ to parent of cell i.

Cell i at level h, $0 \leq h < n$:
Send multipole expansions Φ_i and $\chi_{i,j}$ to parent of cell i.
Receive from each child k its multipole expansions Φ_k and $\chi_{k,l}$.
Form new Φ_i and new $\chi_{i,j}$ using lemma 4.2.2.

Downward Pass

Comment [Form local expansions of potential Ψ_i, due to particles outside cell i and its neighbors, about the center of cell i]

Cell i at level 0:
$\Psi_0 = 0$
Send Ψ_0 to each child k of cell i.

Cell i at level h, $0 < h < n$:
Send to each neighbor j of cell i the temporal multipole expansions and consistency terms of children of cell i.
Receive from each neighbor j the expansions and consistency terms of j's children.
To each child k of cell i send Ψ_k formed according to lemma 4.2.3.
Receive new Ψ_i from parent of cell i.

Cell i at level n:
Receive Ψ_i from parent of cell i.

Computation of Force/Potential Comment [Use some method within each leaf cell to advance all particles with update time t. Exchange temporal multipoles and runaway particles with neighbors.]

Remark *Neighbor Synchronization:* In each cell, particles are independently integrated when their next time-step occurs. This can potentially cause neighboring cells to have different local times. Since near neighbor interactions are performed directly on particles advanced to the same time, neighbor synchronization must be maintained. This is ensured by having each cell advance its time by the minimum of the time-steps of all its neighbors. If a cell's time-step is larger than its neighbors, it waits for its neighboring cells to catch up.

Remark *Integration Method within Cell:* The method used within each cell to advance particles depends on how large the number of particles per cell, s, is. For large s, ($s \gtrsim 10^4$) the temporal multipole algorithm itself can be used for maximum efficiency. Further, within each cell, regularization of close encounters can also be done.

5.5 Complexity Analysis

As stated before, each cell is assigned a processor. We make the following assumption to analyze the complexity of the algorithm. The assumption essentially states that processors do not have to wait in an inactive state while neighbors catch up.

Assumption 5.5.1 (Large Cell) *The number of particles per cell, s, is sufficiently large for both of the following conditions to hold. At least one particle is integrated in each pass through the algorithm in each cell and the size of a cell is much larger than the average inter-particle distance.*

5.5.1 Communication Efficiency

Let the time taken for the temporal multipole expansion of a cell to reach another cell at distance D from it be $\mathrm{Comm}(D)$. We state the following self-obvious no-waiting theorem.

Theorem 5.5.2 (No Waiting) *If* $\mathrm{Comm}(D)$ *is at most* $\tau_\Phi(D)$ *for a parallel algorithm, then no processor has to wait for valid temporal expansions to reach it.*

Let the width of a cell be R_0. $\mathrm{Comm}(D)$ for the pyramid is just the time to go up and down a $\frac{D}{R_0} \times \frac{D}{R_0}$ pyramid which is $2\log\frac{D}{R_0}$ communication steps. For each communication step there is an integration step of average length τ_0. For a homogeneous distribution, τ_0 is approximately given by $\eta\frac{r_0}{v_0}$, where r_0 is the average inter-particle separation. Thus $\mathrm{Comm}(D)$ is $2\log\frac{D}{R_0}\tau_0$ which is $2\eta\log\frac{D}{R_0}\frac{r_0}{v_0}$. For theorem 5.5.2, we require

$$\begin{aligned} \mathrm{Comm}(D) &\leq \tau_\Phi(D) \\ 2\eta\frac{r_0}{v_0}\log\frac{D}{R_0} &\leq \eta\frac{D}{3v_0} \\ 2\log\frac{D}{R_0} &\leq \frac{D}{3r_0} \end{aligned} \tag{5.7}$$

Under the assumption that cells are sufficiently large (so that $R_0 > r_0$), this will be true.

5.5.2 Time and Space Complexity

We show the analysis for two dimensions. For three dimensions, the analysis is exactly analogous. There are N particles and at least $\frac{s}{4}$ particles per cell at the finest level. The number of cells at the finest level of refinement is at most $\frac{4N}{s}$ corresponding to a $\sqrt{\frac{4N}{s}} \times \sqrt{\frac{4N}{s}}$ pyramid with $O(\frac{4N}{s})$ processors.

Each communication step takes $O(1)$ time and each processor spends $O(1)$ time on communication in each pass. Under the large cell assumption 5.5.1, the total time spent on communication in all processors is $O(g)$, where g is the total number of particle integration steps. The the time spent on computation is also $O(g)$. The total time taken is $O(g)$ and this is optimal.

Each cell at the finest level stores at most s particles. Each higher level cell needs only $O(1)$ space. So the space used per processor is $O(s)$.

Blocked time-steps will reduce the constant significantly in the time complexity of the algorithm. All particles in a block get updated at the same time and the compute/communicate ratio increases, making the algorithm more efficient.

Chapter 6

Parallel Algorithm: Non-Homogeneous

Since a homogeneous distribution is symmetric, each processor does roughly the same amount of work. For other distributions, a simple assignment of cells to processors can cause severe load imbalances leading to inefficiency.

Consider the particle distribution shown in Figure 6.1. Particles are concentrated in the two corners of a square in physical space. The processors are arranged in a uniform grid in computational space. It is clear that overlaying physical space on the computational space can create load imbalances, with a few processors doing most of the work. This chapter discusses how to distribute the work fairly across all processors so that the simulation can be done more efficiently. Given

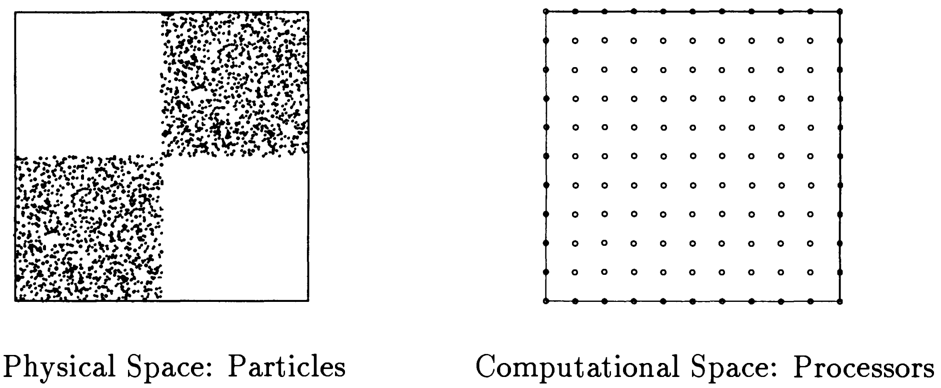


Figure 6.1: Load Balancing: Distribution of Particles and Processors

the local nature of the problem, our emphasis will be on local solutions. We will be less formal than in previous chapters and only attempt to sketch out a solution. However, it seems to the author to be a better approach to load-balancing than other cell-based approaches which do not exploit the locality inherent in the problem.

6.1 Load Balancing

Ideally, we would like to have more computing power available where there are more particles. Define the computation density at a point as $\frac{\rho}{\tau}$ where ρ is the space density of particles and τ is the average integration time-step for a particle at that location. The average computation density of a cell is a measure of the computation requirements of the cell. Informally, we wish to distort physical space-time with varying computation density on to computational space-time with uniform computation density. Sparse regions shrink so that few processors are allocated to them and dense regions expand and are allocated more processors. This will then correspond to a load-balanced computation. Since particles move as the simulation progresses and communication is expensive, the load balancing scheme should be dynamic and local.

Locality is extremely important for efficiency. Most of the communication in the algorithm occurs between processors containing neighboring cells when they compute the near force. Near force computation requires communication of detailed information on particles and cells. In contrast, far force communication costs are small since only the multipole expansions have to be communicated and when using temporal expansions, even this gets amortized. Therefore, the load-balancing scheme would be very efficient if neighbors in physical space are also neighbors in computational space. Before discussing such a scheme, we make a simple but useful observation.

Fact 6.1.1 *If N particles are arbitrarily partitioned into two sets, the force on any particle is the sum of the forces on it due to particles in the two sets.*

Thus physical space can be duplicated so long as particles are not, and a cell can exist in more than one processor simultaneously.

6.1.1 The Center of Work Heuristic

Load balancing is based on the center of work heuristic. Neighboring processors exchange particles to equalize the amount of work they do. Load balancing steps are scheduled periodically. The duration between successive load-balancing steps, T_{bal}, is fixed by mutual agreement among neighbors. The details follow.

6.1.1.1 The Center of Work

Consider a set U of particles. The i'th particle is located at $\boldsymbol{x}_i$ and has time-step t_i at a load-balancing step. Let w_i be the work done on the i'th particle in the previous T_{bal} time. The center of work, $\langle \boldsymbol{X}_U, T_U \rangle$, for the set U is defined to be the work weighted mean of the space coordinates and the time-step of all the particles in U.

$$\langle \boldsymbol{X}_U, T_U \rangle = \frac{\sum_{i \in U} \langle w_i \boldsymbol{x}_i, w_i t_i \rangle}{\sum_{i \in U} w_i} \tag{6.1}$$

6.1.1.2 Work Redistribution

Work redistribution occurs based on the "center of work" heuristic. At the junction $\boldsymbol{J}$ of any four processors, the current center of work $\langle \boldsymbol{X}_c, T_c \rangle$ for all the particles in all four processors is found. See Figure 6.2. The spatial component $\boldsymbol{X}_c$ is used

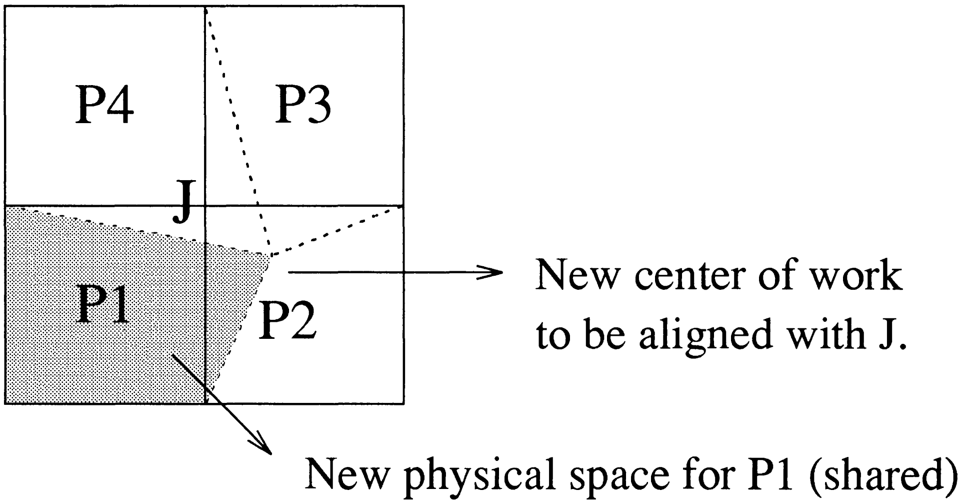


Figure 6.2: Work redistribution based on center of work heuristic

to balance the load by exchanging particles and the temporal component T_c is the duration of the next load-balancing step. Specifically, physical space is distorted by aligning $\boldsymbol{X}_c$ with the junction point $\boldsymbol{J}$ of the four processors. After this alignment, each processor is in charge of all the particles falling within its domain for the next $T_c \equiv T_{\text{bal}}$ units of time (with respect to junction $\boldsymbol{J}$).
This realignment has the following properties:

Locality: Physically adjacent regions will remain adjacent in processor space.

Linearity: The physical space allocated to a processor undergoes only a linear transformation. Since, every processor has non-zero work to begin with, the new junction point will always lie within the physical space owned by one of the four participating processors. Thus, the physical space assigned to any processor by this method will always be a convex quadrilateral.

Fairness: The load on processors after the realignment will be more equitable. Processors with more work in the previous T_{bal} units of time will have less physical space (and fewer particles) and vice versa.

Note that this realignment occurs independently at every single junction J of every 4 processors. Thus "space" assigned to a processor is not fixed but changing.

Fact 6.1.1 can be used to simplify book-keeping. Each processor finds the square bounding its quadrilateral and simulates all cells within this square. As a practical matter, to avoid unnecessary movement of particles between two processors with approximately the same load, the alignment of $\boldsymbol{X}_c$ with $\boldsymbol{J}$ is done only within some tolerance factor and not exactly. In other words, particles are transferred from one processor to another only if there is a significant difference in the amount of work done by the two processors. The load-balancing time-step may also be scaled by a multiplicative constant, so that it tracks the evolution of the system.

6.2 Communication Efficiency

Since physical space is distorted by the load balancing scheme, it is no longer clear that temporal multipoles are available where they are required and when they are required. To show this, we make the following assumption.

Assumption 6.2.1 (Evolution and Load Balancing) *The rate of evolution of the system is much slower than the rate of load-balancing so that the computation density is the same at every processor.*

The idea is that if the processors are load-balanced at the beginning of the simulation, local particle exchanges through the center of work heuristic are sufficient to ensure that they remain load-balanced throughout the simulation. This is a reasonable assumption to make since load-balancing steps are frequent and in each integration step the distance moved by a particle is very small. However, the assumption can break down for pathological situations. For non-pathological situations, if N_i is the number of particles and τ_i the average time-step at processor i, we then have

$$\frac{N_i}{\tau_i} = \text{constant for all processors } i \tag{6.2}$$

From theorem 5.3.1, we know that the validity duration is $\frac{D}{3v_0}$ at distance D from a temporal expansion of particles with dispersion velocity v_0. However, the dispersion velocity is not the same throughout the system. To simplify the discussion, we make the following assumption about the N-body system.

Assumption 6.2.2 (Composition of homogeneous subsystems) *The N-body system is composed of subsystems each of which is homogeneous.*

Being primarily interested in the mesh model of parallel computation, we will only consider nearest neighbor mode of communication. Processors pass on information to their nearest neighbors and we wish to ensure that temporal expansions are received within their validity periods at the destination processors. Therefore, Comm(D), the time for communication of temporal multipoles to distance D, is just the sum of the communication times across these subsystems. With the assumption that the system is comprised of sufficiently large load-balanced locally homogeneous subsystems, we can characterize the velocity with which temporal multipoles are communicated across such a subsystem.

Lemma 6.2.3 *Consider one homogeneous subsystem, located inside a single processor, of a load-balanced N-body system. Let the subsystem have density ρ and an average integration time-step of length τ. Then U, the velocity of communication of temporal expansions within the subsystem is given by*

$$U = \frac{1}{\rho^{1/d}\tau^{1-1/d}} = \rho^{(3-d)/(2d)} \tag{6.3}$$

where d is the number of dimensions.

Proof. Let N be the number of particles in the subsystem and R be its physical size. Since, the subsystem is homogeneous, $N \propto \rho R^d$. Once load-balanced, all subsystems have the same constant computation density. Therefore,

$$\frac{N}{\tau} = \frac{R^d\rho}{\tau} = \text{constant} \tag{6.4}$$

Now, the expansion is passed on to the next subsystem in time τ. Therefore it travels a physical distance R in time τ and its velocity of communication is given by

$$U = \frac{R}{\tau} \propto \frac{1}{\rho^{1/d}\tau^{1-1/d}} \tag{6.5}$$

ρ and τ are related by $\tau \propto \rho^{-1/2}$. Hence,

$$U \propto \frac{1}{\rho^{1/d}\rho^{-1/2+1/(2d)}} \propto \rho^{(d-3)/(2d)} \tag{6.6}$$

$\square$

By the above lemma, U is a constant in three dimensions while for two dimensions, $U \propto \rho^{-1/4}$. Now we will estimate the time taken by a temporal expansion to travel across the entire N-body system. Physically, the expansion is always despatched by each processor on its path along the shortest physical path to its destination. This physical path however is dilated by load balancing.

Theorem 6.2.4 *Consider a load-balanced N-body system consisting of s locally homogeneous subsystems. Let the i'th subsystem have density ρ_i, size D_i and average integration time-step τ_i. Then,* Comm(D)*, the time for a temporal expansion to travel a distance D across the entire system is given by*

$$\mathrm{Comm}(D) \;=\; \sum_{i=1}^{s} D_i K \rho_i^{(3-d)/(2d)} \tag{6.7}$$

where K is a constant.

Proof. The expansion is sent along the shortest path in physical space which might pass through all of the s locally homogeneous subsystems. The time for the expansion to travel a distance D is simply the sum of the times it takes to travel through each subsytem on its path.

$$\mathrm{Comm}(D) = \sum_{i=1}^{s} \mathrm{Comm}(D_i) \text{ and } D = \sum_{i=1}^{s} D_i \tag{6.8}$$

By lemma 6.2.3,

$$U_i \;\propto\; \rho_i^{(d-3)/(2d)} \tag{6.9}$$

Let the time taken to cross the i'th subsystem be Comm(D_i). Then

$$\mathrm{Comm}(D_i) = \frac{D_i}{U_i} \;=\; K D_i \rho_i^{(3-d)/(2d)} \tag{6.10}$$

Therefore, Comm(D) can be calculated as

$$\mathrm{Comm}(D) \;=\; \sum_{i=1}^{s} \mathrm{Comm}(d_i) \;=\; \sum_{i=1}^{s} D_i K \rho_i^{(3-d)/(2d)} \tag{6.11}$$

□

$\tau(D)$, the validity duration of the temporal multipole at distance D, is simply $\frac{D}{3v_j}$ where v_j is the dispersion velocity of the set of particles whose potential the expansion represents. If Comm(D) is less than $\tau(D)$, temporal multipole expansions will reach their destinations within their validity periods. To ensure this, we require

$$\sum_{i=1}^{s} D_i K \rho_i^{(3-d)/(2d)} \leq \frac{D}{3v_j} \tag{6.12}$$

This will be true if $Kv_j\rho_i^{(3-d)/(2d)}$ is less than $\frac{1}{3}$ for all i and for all j. For 3 dimensions, we just need Kv_j to be less than $\frac{1}{3}$ for all j.

If we ensure that Comm(D) is less than $\tau(D)$ for this nearest neighbor communication model, the algorithm can be implemented on a mesh architecture in a manner similar to that discussed for homogeneous systems.

Chapter 7

Parallel Algorithm on Various Architectures

In this chapter we discuss the implementation of the parallel temporal multipole algorithm on various parallel architectures. We discuss the algorithm for a two dimensional system. The extension to three dimensions is straightforward. We will refer to steps where particle integrations actually occur as *integration* steps. Those steps which only involve communication of temporal multipole expansions will be referred to as *communication* steps. By requiring that the number of communication steps per integration step be $O(1)$, we can ensure that the algorithm remains optimal.

The parallel algorithm on the hypercube is optimal and has the same asymptotic complexity as the sequential algorithm, namely $O(g)$. On the mesh, we present an algorithm which is inefficient by a multiplicative factor of $\log P$ on P processors.

7.1 Relaxed Pyramid

For conveniently mapping the algorithm on to existing processor architectures, we first define a *relaxed pyramid* on which the algorithm works as efficiently as on a pyramid. Relaxed pyramids can be easily embedded in hypercubes and meshes.

Consider an $M \times M$ pyramid. The corresponding relaxed pyramid also has an $M \times M$ base and properly contains the pyramid. We will refer to the nodes of the relaxed pyramid which are also present in the pyramid as *pyramid* nodes. Between a pyramid node i at level h and its parent pyramid node j at level $(h-1)$, the relaxed pyramid has a *child-parent chain* of $(T_h - 1)$ additional *chain* nodes: $j \equiv V_{ij}^0, V_{ij}^1, \ldots, V_{ij}^T \equiv i$ where T_h is $\lfloor \frac{\lambda M}{2^{h+1}} \rfloor$. Here, λ is a parameter to be chosen depending on the architecture. V_{ij}^{k} is connected to V_{ij}^{k-1} and is its only child. The physical interpretation is that all the nodes $V_{ij}^{k}, k \neq 0$ represent the same pyramid node but hold temporal expansions corresponding to different time-steps. The number of nodes in the relaxed pyramid is $2\lambda M^2$ ($\lambda > \frac{2}{3}$) as against $\frac{4}{3}M^2$ in the pyramid and the number of levels is $\lambda(M-1)$ as against $\log M$.

Particle consistency terms are formed as usual. However, there is one technicality, regarding the child-parent chains. Consider particle consistency terms at a pyramid node i at level h of the pyramid. A new consistency term $\chi_{i,j}^{t}$ or simply χ^t (suppressing subscripts for convenience) arrives at time t. Let the consistency terms of the previous T_h time-steps be $\chi^{t-1}, \chi^{t-2}, \ldots, \chi^{t-T_h-1}$. Then the consistency term at node i at time t can be formed using the time-composition property (4.4).

$$X^t = \sum_{j=0}^{T_h-1} \chi^{t-j} \tag{7.1}$$

This is true for each neighbor j of i. From this description, it might appear that a level h pyramid node needs to store $O(2^{n-h})$ consistency terms. If h is 0 this requires $O(M)$ space.

This requirement to store $O(M)$ terms can be avoided by the *node-chaining* technique. The essential idea is to treat nodes in a series or chain of nodes as storage space. Suppose node i receives data from nodes in chain1 and wants to receive the same data (in the same order) after a delay of $2T$. See Figure 7.1. Then, it just chooses a suitable chain of processors chain2 of length T and sends the data from chain1 into chain2. The last node in chain2 "reflects" the data so that node i receives the data at the end of time $2T$ as required.

Now, note that at time t, the consistency term X^t can also be formed as

$$X^t = X^{t-1} - \chi^{t-T_h} + \chi^t \tag{7.2}$$

Only the terms χ^t, X^{t-1} and χ^{t-T_h} are required at time t. The intermediate terms can be sent back along the child-parent chain halfway in such a way that they arrive at the node exactly in time to be used.

7.1.1 Homogeneous

The temporal multipole algorithm is the same on a relaxed pyramid except that now we allow cells with a single child. Cells in the chain (except for those also present in the original pyramid) do not participate in neighbor-neighbor interactions. A multipole expansion created at time t in some leaf cell reaches another leaf cell at distance D from it after first going up the relaxed pyramid then going down. Comm(D), the number of communication steps required to travel to a

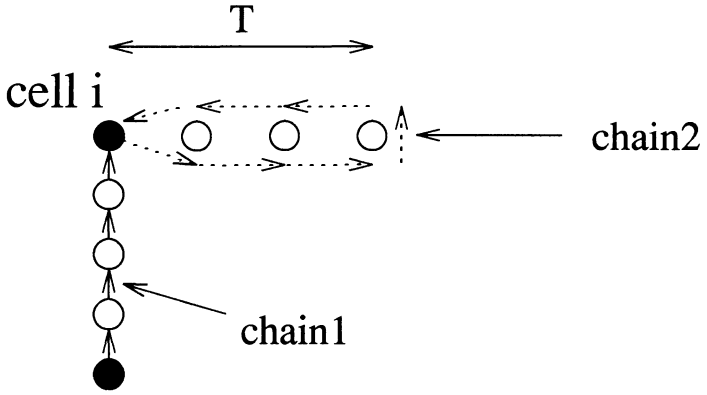


Figure 7.1: Principle of Node chaining
Node i receives data from chain1 once again after delay $2T$ by sending it along chain2 of length T.

processor at distance D, is equal to twice the height of the relaxed pyramid with base $D \times D$, which is $2\lambda(D-1)$. If we choose the number of communication steps executed per computation step to be 2λ, Comm(D) satisfies theorem 5.5.2 (under the large cell assumption).

7.2 Hypercube

To embed a pyramid on the hypercube, we proceed as in Zhao and Johnsson [ZJ91] and as described in Ho and Johnsson [HJ87,HJ90]. Successive levels of the pyramid correspond to finer grids. The finest level grid is embedded using gray codes independently for each axis. Since the number of cells is a power of two along each axis, this is easily done by partitioning the hypercube connections among the two axes. Higher level grids are embedded on to a subset of the same set of hypercube processors. Processors at level k consist of all processors with coordinates which are a multiple of 2^{n-k} along both axes. The distance between neighbors on any level is at most 4 (for non-diagonal neighbors it is 2). The parent-child distance is at most 4 as well, since the parent is also mapped onto one of the neighbors. In this embedding, communication can occur only one level at a time.

7.2.1 Homogeneous

Although the embedding is that of a pyramid, the simulation will be that of a relaxed pyramid. Conceptually, each child cell simulates its child-parent chain in a sense made more precise below.

In communication step t, level $h(t)$ of the relaxed pyramid is simulated, where

$$h(t) = \frac{M}{2^k}, \qquad k = \max_{i \geq 0}\{(t \bmod M) \bmod 2^i = 0\} \tag{7.3}$$

Level 2^h is simulated once in every $\frac{M}{2^h}$ communication steps. This being the case, particle consistency terms for intermediate steps need not be formed at all. (Note that only levels present in the actual pyramid are simulated but exactly after a delay corresponding to data movement through a relaxed pyramid.) The child-parent delay is 2^{n-h} for a child cell at level h. However, since data is not pumped continuously up the child-parent chain but only at intervals of 2^{n-h}, old data is used for the intervening 2^{n-h} steps also. Hence, the algorithm functions as it would on a relaxed pyramid with a value 4 for λ. So in this case, 2λ or 8 communication steps are required per integration step.

The number of processors required, for N particles, is $\frac{4N}{s}$ and space required per processor is $O(\frac{1}{2}\log\frac{4N}{s} + s)$ since one processor could be simulating $\frac{1}{2}\log\frac{4N}{s}$ pyramid processors.

7.3 Mesh

We will assume that the mesh neighbor structure reflects the algorithm neighbor structure. Thus, a processor has 8 neighbors in 2D and can communicate with each of them in $O(1)$ time. First, we show how to embed an $M \times M$ relaxed pyramid in an $M \times M \times \log M$ mesh with $O(1)$ slow down of the temporal multipole algorithm. Since, an $M \times M$ mesh can simulate an $M \times M \times \log M$ mesh with $O(\log M)$ slow down, the temporal multipole algorithm can be implemented on an $M \times M$ mesh with $O(\log M)$ slowdown.

The embedding of the $M \times M$ pyramid in an $M \times M \times \log M$ mesh is done in the obvious way. Let both be placed symmetrically about the origin of some coordinate reference frame. Each processor on the pyramid is mapped to the nearest processor in the mesh. The bottom layer is the same in both and the h'th level of the pyramid is mapped onto the h'th level of the mesh. The relaxed pyramid has the same embedding except that each child-parent chain is embedded on the parent layer and corresponds to the shortest path between child and parent. In this embedding, child-parent communication (on the relaxed pyramid) takes $O(1)$ time. However, on the h'th layer, neighbor-neighbor communications take $O(1)$ time on the relaxed pyramid but $O(\frac{M}{2^h})$ time on the mesh. We will postpone a discussion of this delay and its effects.

When the data from its children arrives at a node, it stores this data and forwards a copy to each of its neighbors. When the data from the neighbors arrives, the far potential is computed for each child and dispatched to the child. Since all the neighbors are equidistant, their data arrives simultaneously. So corrections for intervening time-steps are accumulated and applied individually, after computation, to each child's potential.

The data received from children is part of the data sent to a cell's neighbor and also later used in computations. This data should therefore be stored, until the neighbor data arrives. But the storage required for this, over successive time-steps, could be as large as $O(M)$ at one processor. We can alleviate this problem by node-chaining. In Figure 7.1, the child-parent chain is chain1 and half the neighbor-neighbor chain is chain2. The modified child-parent chain passes twice through the parent. Think of data being pumped pipeline fashion on this chain. The first time the children's data passes through the parent, a message to the neighbor can be sent. (Due to symmetry, data from all children arrives at the same time). The second time, the children's data arrives simultaneously with the neighbors' data. Required results can be computed and sent out immediately and no storage is required.

7.3.1 Homogeneous

With node-chaining of the child-parent chain, the delay for a child-parent communication at level h is $3 * 2^{n-h-1}$, which corresponds to a relaxed pyramid with λ equal to 3. To satisfy theorem 5.5.2, 2λ or 6 communication steps are needed per integration step.

Coming back to neighbor-neighbor communication delay, this delay is 2^{n-h} on level h. The distance traveled by a particle in one integration step is proportional to the inter-particle distance r_0, while the distance travelled by the temporal expansion in one communication step is proportional to the inter-cell distance R_0. By the assumption 5.5.1, $R_0 \gg r_0$. It is easy to see that the distance that a particle can travel in 2^{n-h} communication steps is less than the size of a cell at level $h+1$. Therefore, particle consistency terms remain valid.

In summary, the number of processors required is $O(\frac{4N}{s})$, the time per N-body iteration is $O(\frac{1}{2}\log\frac{4N}{s} + s)$ and space per processor is $O(\frac{1}{2}\log\frac{4N}{s} + s)$.

For the sake of completeness, we mention that an optimal algorithm on the mesh seems possible. Preliminary results indicate that if the embedding of the relaxed pyramid is not static but dynamic, the logarithmic multiplicative factor can be removed from the time-complexity.

7.4 3D Algorithm on 2D Mesh

Conventional parallel machines have a two-dimensional mesh topology while the N-body algorithm is usually studied in three dimensions. We, therefore, require a way of mapping the computations of the 3D mesh on to a 2D mesh of processors.

Specifically, we wish to map computations on an $M \times M \times M$ mesh on to an $M \times M$ mesh. This is a well-studied problem [Lei92]. Moreover, since each processor now simulates M other processors, there is more computation involved per communication step. This can be used to hide communication costs and it is straightforward to change the algorithm so that the $\log M$ inefficiency is removed making the algorithm optimal.

7.5 A Distributed System

It would be desirable to be able run N-body codes on conveniently available workstation networks. There are two problems with such networks. They have a high communication latency and they are typically ethernets with broadcast or global communication primitives. The rough analysis below, under extremely optimistic assumptions, shows that the main limitation in implementing the temporal multipole algorithm on such networks comes from the latter. With current technology, processors will spend most of their time waiting for communication to occur.

For simplicity, let us look at a homogeneous distribution of particles. Let there be s particles per cell and C cells per processor. Let the information in bytes communicated to a neighbor be P_0 bits per particle and C_0 bits per cell. Only cells on the common boundary and their ancestors need be communicated to a neighbor. Then, the amount of information communicated to each of 8 neighbors is approximately $\sqrt{C}(2C_0 + sP_0)$ (in 2 dimensions). So the total data sent out by a processor per integration step is $8\sqrt{C}(2C_0 + sP_0)$. Assuming a message transfer rate of B bits per second,

$$\text{Time Taken for message send} = \frac{8\sqrt{C}(2C_0 + sP_0)}{B} \tag{7.4}$$

Optimistically, let us assume that receiving messages from neighbors also requires only this much time. Let the communication time be overlapped with computing the near force. This is to avoid processors spending most of their time waiting for messages. With C cells, this will involve approximately $O(8Cs^2)$ force calculations. If a force calculation takes time T_f, computation time is $8Cs^2T_f$. For maximum efficiency, the computation time and the communication time should be roughly equal so that the processor spends no time waiting. Therefore, we want

$$\begin{aligned} 8Cs^2T_f &= \frac{2 \times 8\sqrt{C}(2C_0 + sP_0)}{B} \\ C &= 4\left(\frac{2C_0 + sP_0}{Bs^2T_f}\right)^2 \end{aligned} \tag{7.5}$$

For optimistic values: 500 bits for P_0, 5×10^4 bits for C_0, 10×10^6 bits per second for B, 100 for s, 1×10^{-6} second for T_f, the obtained value for C is 10 which corresponds to a 1000 particles per processor roughly. The required latency is $0.1s$ which also is not too stringent.

The problem is that the ethernet is intrinsically a broadcast network. If there are p processors on the network, all working on the N-body problem, the latency will be p times the intrinsic $0.1s$ latency since each processor has to wait for every other processor to finish. Then the number of particles per processor has to increase enormously to hide this latency of $0.1p$ and in turn causes the latency to go up since more particles per processor requires more information to be communicated.

The essential problem is one of trying to communicate local information globally. The N-body problem is more suited to networks which do not suffer from this problem.

Chapter 8

Implementation

The sequential algorithm in two dimensions was implemented on a Sun sparcstation. The parallel algorithm, also in two dimensions and without any load balancing, was implemented on a 16-node Intel iPSC/860 hypercube. The code written in C consisted of around 4000 lines for the sequential implementation and around 5200 lines for the parallel. Since each processor itself employed the temporal multipole algorithm, the parallel code was a superset of the sequential code.

8.1 Implementation Details

Below we discuss some of our implementation decisions which might be useful for future implementations.

8.1.1 Scaling

Particle masses (and charges) were scaled so that the total mass and charge were each 1. Thus, for N particles, each particle had a charge and a mass of $\frac{1}{N}$. All particles were placed in a unit square centered at the origin.

8.1.2 Tree of Cells

A tree of cells with each leaf cell containing a list of particles within it was maintained. The tree was incrementally updated. When a particle moved from cell i to cell j, it was deleted from cell i's list and inserted into cell j's list. When the number of particles in a leaf cell increased beyond s, it was divided into children cells. Likewise, if the number of particles in all the children of a cell decreased below s, that cell was made a leaf and its children were deleted. Temporal expansions were suitably updated in each case. The number of particles per cell was fixed at 40 in all the tests reported here.

8.1.3 Taylor Series Integration

Particles were integrated using a Taylor series. Time steps for particles as well as cell temporal expansions were determined by the criterion used by Aarseth [Aar85]. Blocked time-steps were used for efficiency.

8.1.4 Temporal Multipole Expansions

Instead of explicitly forming temporal multipole expansions and time derivatives, divided differences as described by Ahmad-Cohen [AC73] and Aarseth [Aar85] were used. Particle forces in the previous four time-steps were used to form the differences. The same divided-difference technique was extended to form temporal expansions for cells also. For each coefficient in the multipole expansion, divided differences were individually formed up to the 4th order. These differences were used to extrapolate the expansions and also to predict the next update-time. Fourth order divided differences need values in 4 previous time-steps to work and leapfrog integration with shared time-steps was used for initialization.

Divided differences have the advantage of simplicity and speed over forming the complete temporal expansion. The disadvantage is that they are not sufficiently sensitive to close encounters and tend to have a time-lag effect.

Particle consistency terms only at the leaf level were formed and used. At lower levels, particle consistency was ignored.

8.1.5 Optimizations of Multipole Manipulations

Some optimizations, originally suggested and implemented by Greengard in his code were used in modified form in the multipole manipulation procedures. The optimizations take advantage of the symmetry of the IList cells to "Convert" 4 expansions at the same time with cost $O(4p + p^2)$ instead of $O(4p^2)$. Also, fewer number of terms were converted for IList cells which were further away. Multipole expansions for the forces, instead of potentials, were directly formed. Powers of the complex number vector from cells to their ilist members and children were pre-computed and stored. This allowed expansions to be evaluated rapidly.

8.1.6 Parallel Optimizations

Only information about cells on the shared boundary with a neighbor and multipole expansions for the ancestors and siblings of these cells were communicated to the neighbor. For each processor, a processor tree was formed. This tree refined the entire system simulation space top-down although only that processor's particles were present in this space. Processor level cells corresponding to neighbor processors and interaction list processors were also formed. This allowed temporal multipoles to be updated conveniently.

An effort was made to hide communication latency in the parallel version. After sending out messages to all neighbors, a processor instead of idly waiting for responses, calculated the near forces in between particles within the processor itself.

8.2 Verification

Some tests were done to verify the fidelity of the algorithm. The first concern was whether the accuracy of force calculation using multipoles is sufficient. The second was whether the accuracy of integration improved when the temporal multipole algorithm was used and all parameters were tuned for increased accuracy.

8.2.1 Accuracy of Multipoles

Table 8.1: Multipole Order and Force Error

Order (p)	Precision (2^{-p})	Force Error[a]
4	6.25×10^{-2}	3.0814×10^{-3}
7	7.81×10^{-3}	1.3525×10^{-4}
10	9.76×10^{-4}	2.1064×10^{-05}
14	6.10×10^{-5}	4.2115×10^{-08}
17	7.63×10^{-6}	4.1295×10^{-09}

[a]Maximum Absolute Relative Error = $\max_{i=1}^{N} \left| \frac{\Delta F_i}{F_i} \right|$

Accuracy of multipole calculations were tested using a simple static 2-body problem. Force calculation should show exponential convergence to the exact force

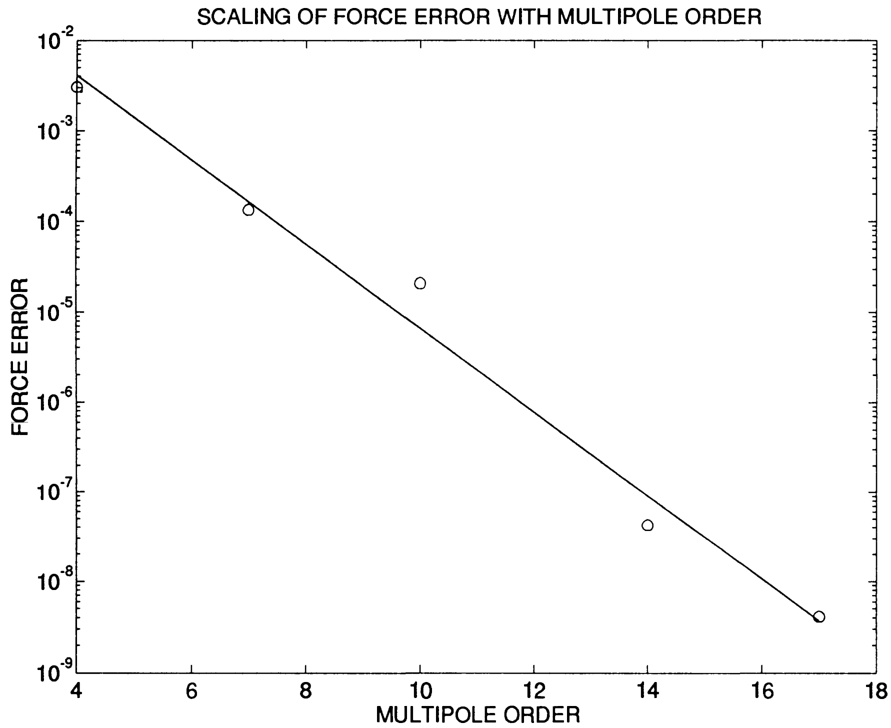


Figure 8.1: Error in force computed using multipoles

as the order of multipoles is increased. Table 8.1 and Figure 8.1 show the results. The force error used was the maximum relative absolute error i.e.

$$\text{Error} = \max_{\text{all particles}} \left| \frac{\text{Actual force} - \text{Multipole force}}{\text{Actual force}} \right| \tag{8.1}$$

The semilog plot of multipole order against the force error is almost a straight line as expected. The convergence is not exactly linear because the multipole expansion is a mixed-sign Taylor series as discussed by Zhao [Zha87].

8.2.2 Accuracy of Integration

The fidelity of integration was tested using a uniformly distributed set of 100 particles. The temporal multipole algorithm was compared against the individual time-step scheme for position and velocity errors. Relative error in energy conservation was also observed. Increased accuracy in integration involved changing three parameters. The initialization error (leapfrog scheme was used in initialization) was made smaller by decreasing the leapfrog time-step (T_L). The force computation error was made smaller by increasing the order p of the multipoles used. (q was fixed at 4 since we used only 4th order divided differences). Finally, the efficiency parameter η was made smaller to decrease the integration error. Since leapfrog integration is second order, these parameters have to be scaled in the proportionality $T_L^2 \sim \eta^6 \sim 2^{-p}\eta^2$ since that is the extent of error in position due to initialization, integration and force calculation respectively.

Table 8.2 and Figure 8.2 show the results. Again convergence is observed on the log-log plot of integration precision against the errors. Note that at the highest accuracy of integration, there is no improvement in energy error. This seems to be due to round-off error since the direct scheme also has this level of energy error.

Table 8.2: Accuracy of Integration and Errors

Order	Accuracy	Error		
p	2^{-p}	Position[a]	Velocity[b]	Energy[c]
4	6.25×10^{-2}	8.62×10^{-11}	1.75×10^{-6}	6.67×10^{-9}
7	7.81×10^{-3}	2.94×10^{-12}	8.80×10^{-8}	3.53×10^{-11}
10	9.76×10^{-4}	2.30×10^{-13}	4.48×10^{-9}	4.69×10^{-12}
14	6.10×10^{-5}	2.97×10^{-15}	9.53×10^{-11}	1.24×10^{-13}
17	7.63×10^{-6}	3.64×10^{-16}	1.06×10^{-11}	1.37×10^{-13}

[a]Average Absolute Error = $\sum_{i=1}^{N} |\Delta x_i| / N$
[b]Maximum Absolute Error = $\max_{i=1}^{N} |\Delta v_i|$
[c]Absolute Relative Error = $|\frac{\Delta E}{E}|$

8.3 Sequential Algorithm: Results

In the sequential case, three different distributions were used to test the algorithm against other standard algorithms. One was a homogeneous distribution ("RANDOM") with a uniform distribution of particles. The other two were power law distributions with density $\rho(r) \propto r^{-\alpha}$ and velocity $v(r) \propto r^{(1-\alpha)/2}$. "POWER

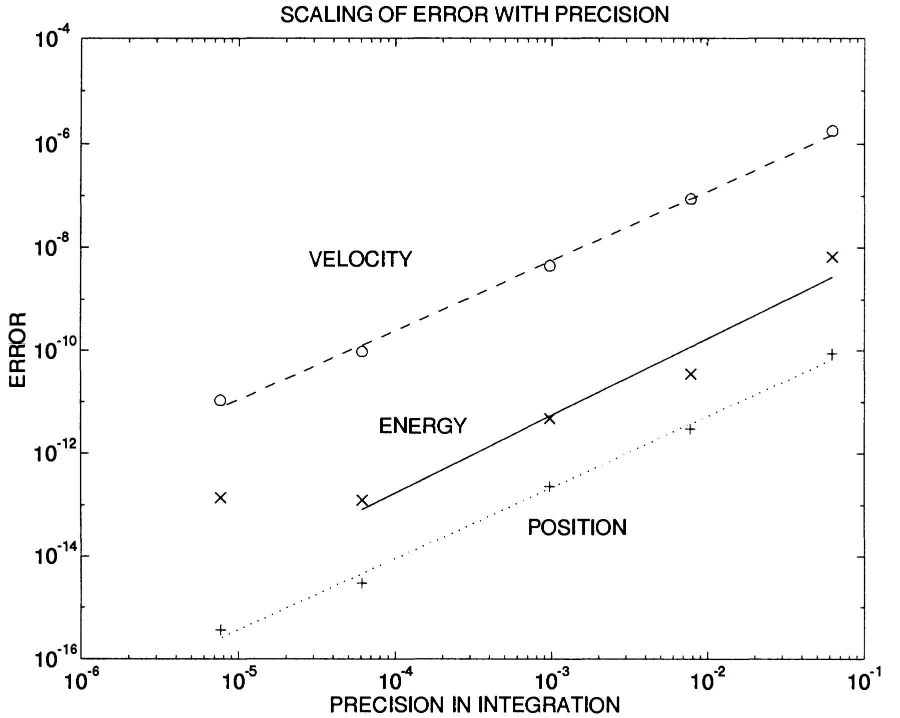


Figure 8.2: Integration precision against energy, position and velocity errors

LAW 1" corresponded to a value 1 for α and "POWER LAW 2" corresponded to a value 1.5 for α. Figure 8.3 shows a sample distribution of 500 particles for each of the three cases. (The '+' shaped empty region in POWER LAW 2 at the origin seems to be an artifact of the pseudo-random number generator used to generate the distribution.) Finally, the asymptotic scaling of the sequential temporal multipole scheme was tested.

All results are with a multipole expansion of order 7 (accuracy of $\approx 10^{-2}$ in force calculations). The time-steps for the shared time-step schemes were chosen to be $\frac{0.01}{\sqrt{N}}$ for RANDOM, $\frac{0.0025}{\sqrt{N}}$ for POWER-LAW 1 and $\frac{0.002}{\sqrt{N}}$ for POWER-LAW 2. These were approximately the minimum time-step chosen by the direct individual time-step scheme for that distribution.

A simple Barnes-Hut algorithm was also implemented for comparison purposes. (The opening parameter in the algorithm, θ was set to 0.5.) So we had three algorithms: Direct, Barnes-Hut and Multipole. In addition, we had two integration methods: shared time-step with leapfrog and individual time-steps with divided

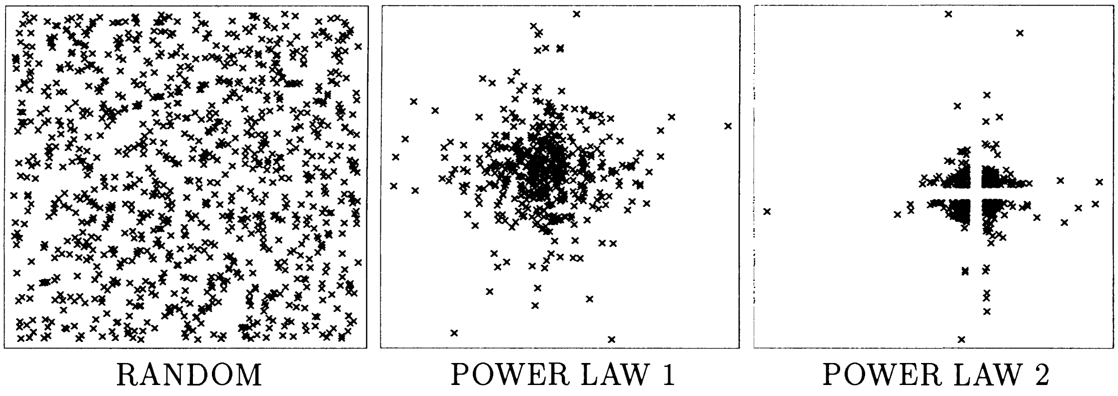


Figure 8.3: Test Distributions of 500 Particles

differences. (Note that multipole with shared time-steps is the same as the fast multipole algorithm while multipole with individual time-steps is the same as the temporal multipole algorithm.) These six schemes were all tested for a varying number of particles for each of the three distributions mentioned above. In each case, energy and momentum were monitored for conservation. To make computations for differing numbers of particles comparable, each distribution was integrated for a time equivalent to 50 shared time-steps.

8.3.1 Homogeneous Distribution

Table 8.3 and Figure 8.4 summarize the timing results. Cases in which time taken was excessive were omitted. The energy and momentum errors are also shown. In comparing the algorithms, the time taken is not the sole criterion. The errors in conservation of energy and momentum can be equally or more important depending on the application. In each algorithm, these errors can be controlled by changing parameters, in which case the time taken will change.

Nevertheless, we are most interested in the algorithmic complexity and the time taken. Both of the direct algorithms have complexities proportional to $O(N^2)$ as is visible from the graph. The individual time-step scheme performs considerably better than the shared time-step scheme for all three methods. The Barnes-Hut algorithm has an almost linear performance but the slope is much steeper than for the multipole algorithms. Nevertheless, it catches up in performance with the direct shared time-step method at a 1000 particles. The sudden increase in slope between 1000 and 2000 particles for the multipole algorithm is apparently because the asymptotic regime has not yet been reached, and an increase in the number of

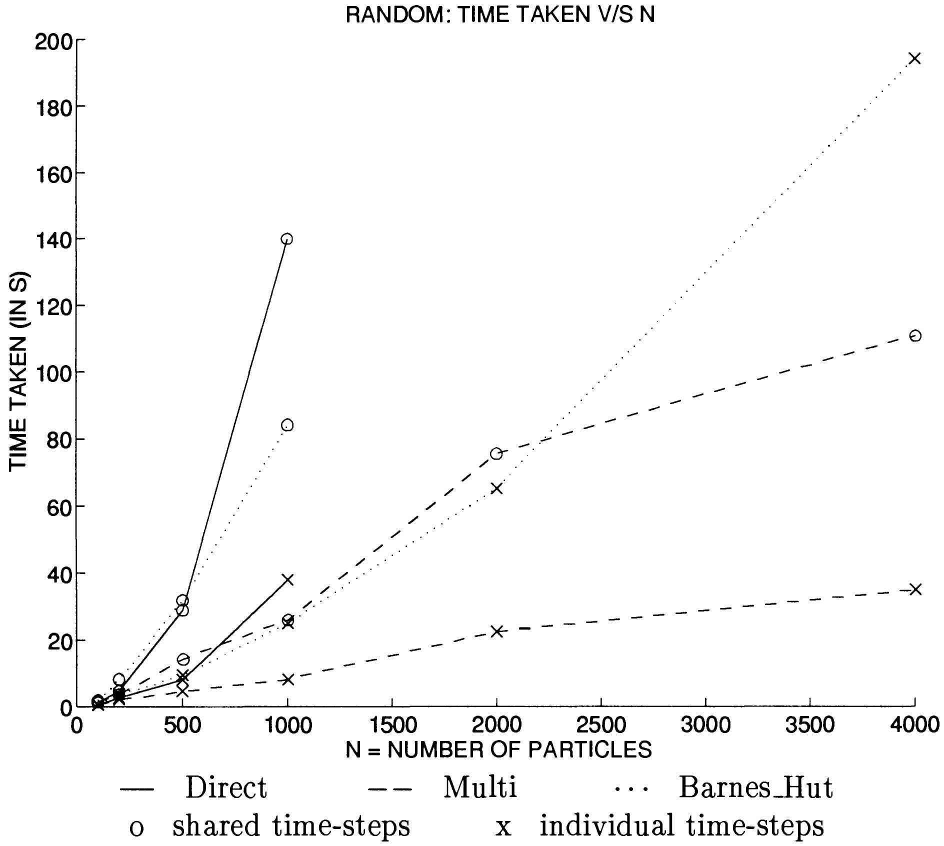


Figure 8.4: Time taken by various algorithms for homogeneous distributions

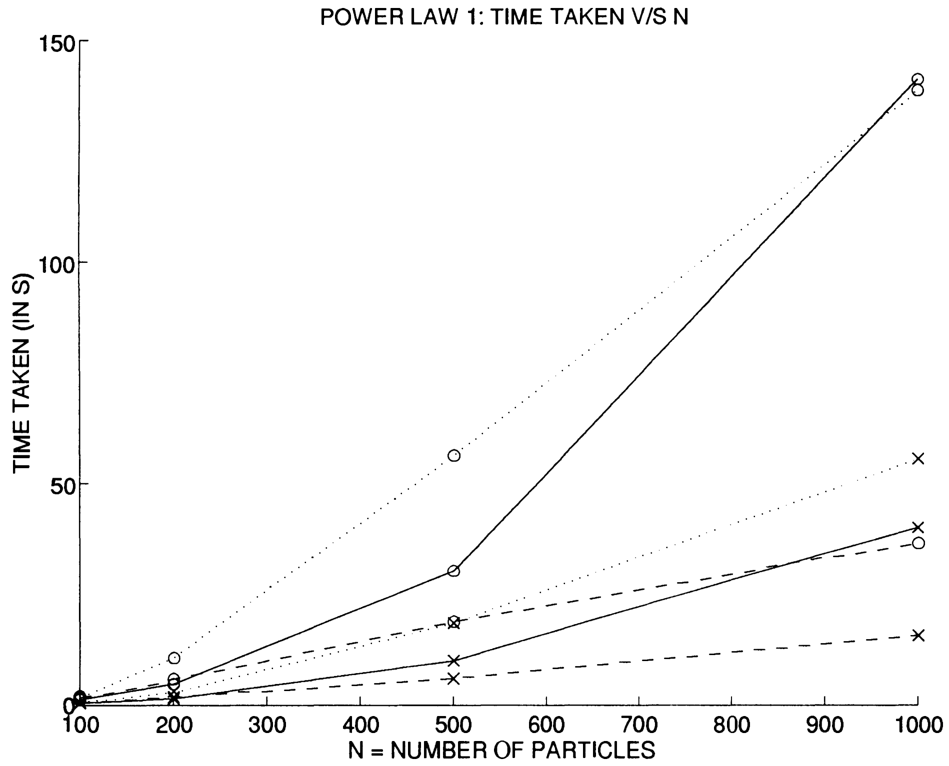
levels of the tree affects the performance.

8.3.2 Power-Law Distributions

Tables 8.4, 8.5 and Figures 8.5, 8.6 show the obtained results.

The direct schemes show no change in performance. However, all multipole schemes perform slightly worse since the number of tree levels is larger. In particular, Barnes-Hut performs significantly worse. Since there are 9 levels in the tree for POWER LAW 2 for a 1000 particles, this is to be expected. The direct scheme with individual time steps performs as well as or better than all the shared time-step schemes even for a 1000 particles, for POWER LAW 1, demonstrating the importance of temporal optimization.

Since, we implemented particle consistency corrections only at the bottom level,



— Direct   – – Multi   ··· Barnes_Hut

o shared time-steps   x individual time-steps

Figure 8.5: Time taken by various algorithms for POWER LAW 1 distribution

the tree algorithms with individual time-steps did not conserve momentum as well as the other schemes for POWER LAW 2. For a 1000 particles, the tree individual schemes conserved momentum to only 1 part in 10^4, while the direct individual scheme conserved momentum to 1 part in 10^5. In general, it also seems to be true that the shared time-step schemes conserve momentum better while the individual time-step schemes conserve energy better. But this could be an artifact of the different integration methods used.

8.3.3 Asymptotic Scaling

To test the asymptotic scaling, the temporal multipole algorithm was run on different numbers of uniformly distributed particles, for a fixed physical time. A distribution with more particles is treated as a higher resolution sampling of a

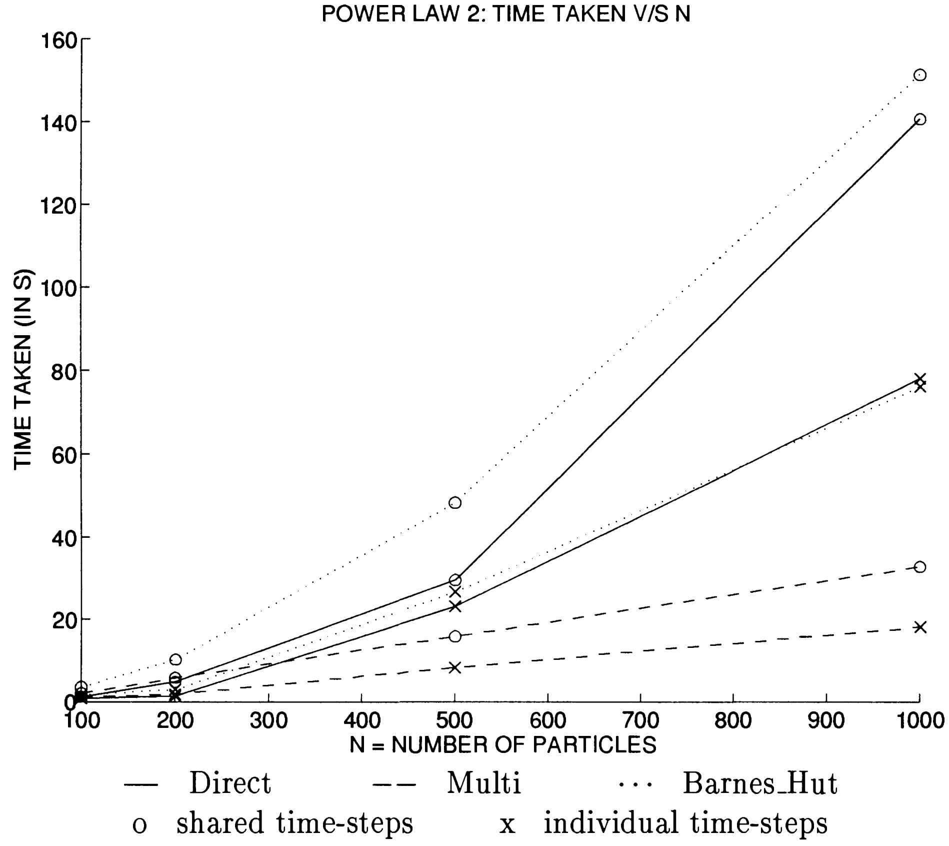


Figure 8.6: Time taken by various algorithms for POWER LAW 2 distribution

homogeneous distribution.

Table 8.6 and Figure 8.7 depict the results. As expected, the log-log plot of time taken against number of particles is a straight line. The slope of the line is 1.535 in close agreement with the value of 1.5 predicted theoretically. We use only N above 4000 since the asymptotic regime seems to be reached only beyond this number of particles.

8.4 Parallel Algorithm: Results

In the parallel case, only the direct algorithm with shared time-steps and the multipole algorithms with shared and individual time-steps were tested for homogeneous distributions (no load balancing). Most of the powerful parallel machines being

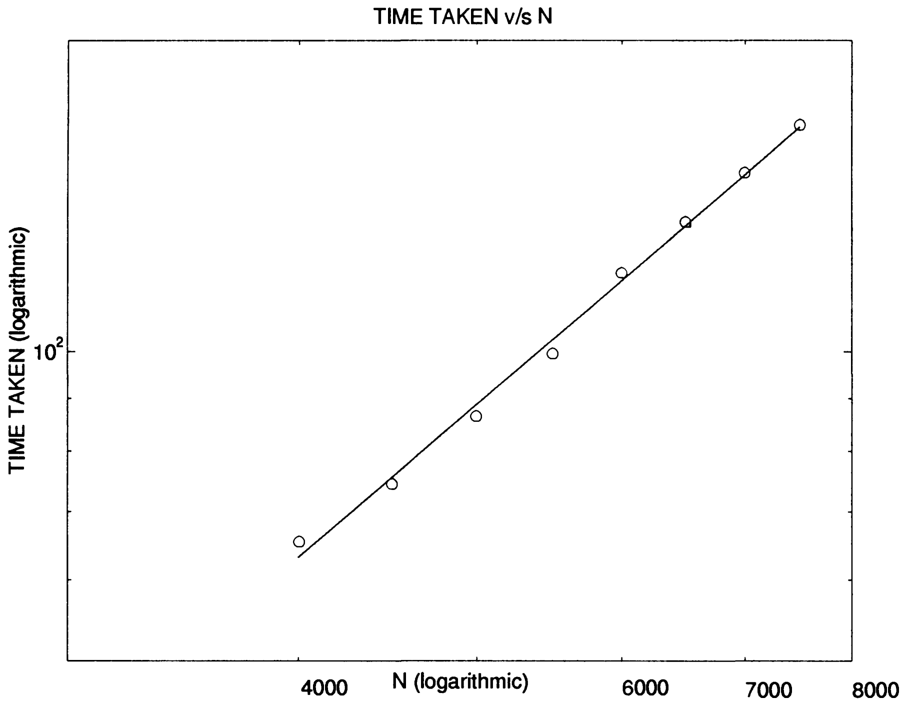


Figure 8.7: Asymptotic scaling with N of Temporal Multipole algorithm

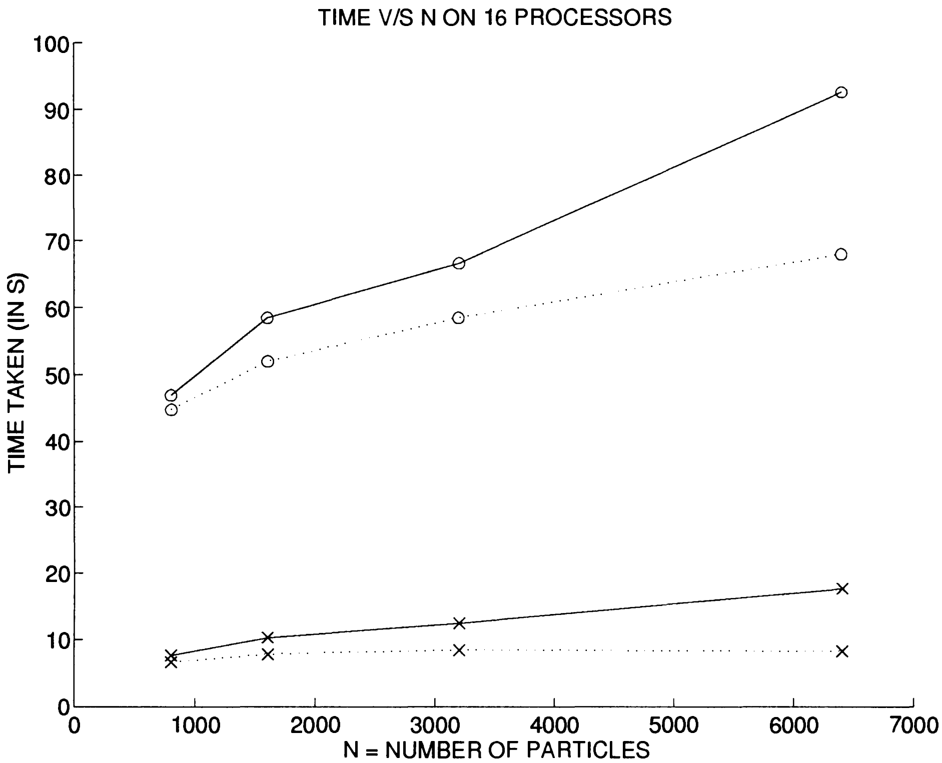
built have 2D mesh architectures. Therefore, even though the implementation was on a hypercube, only 2D mesh connections were actually used. A hypercube implementation was not done since it would not have given any new insights for as small a number of processors as 16 or 32.

Since the multipole algorithm requires a square domain whose side is a power of 2, the number of processors is constrained to be a power of 4. The algorithm was tested on 1, 4 and 16 processors. The iPSC/860 has a relatively high latency of $\approx$ 75 microseconds, while one floating point operation takes $\approx$ 0.3 microseconds. That is, communication is very expensive compared to computation. We break up the total time taken into into three parts: T_{comp}, the time spent on computation; T_{wait}, the time spent waiting for messages; and T_{OH} the time spent in preparing, sending and receiving messages[1]. The communication cost T_{comm} is the sum of T_{wait} and T_{OH}.

[1]Due to measurement problems, the assignment of costs to T_{wait} and T_{OH} is not very accurate. T_{OH} also includes some time spent in busy-wait loops.

8.4.1 Computation and Communication

Table 8.7 and Figure 8.8 show the computation and communication times for 16 processors for the different algorithms for different N. The direct algorithm is not shown in the figure since communication time is negligible and computation time is excessive. The temporal multipole algorithm is more efficient in both communication and computation than the fast multipole algorithm. Note also that the communication time for the temporal algorithm remains almost unchanged with the number of particles.



— Total time ··· Communication time
o shared time-steps x individual time-steps

Figure 8.8: Time taken by multipole algorithms on 16 processors

Figure 8.9 depicts the percentages of times each algorithm spends in computing versus communicating. The direct algorithm spends almost all its time computing while the shared time-step multipole algorithm spends more than half its time communicating. The temporal multipole algorithm also spends spends a large per-

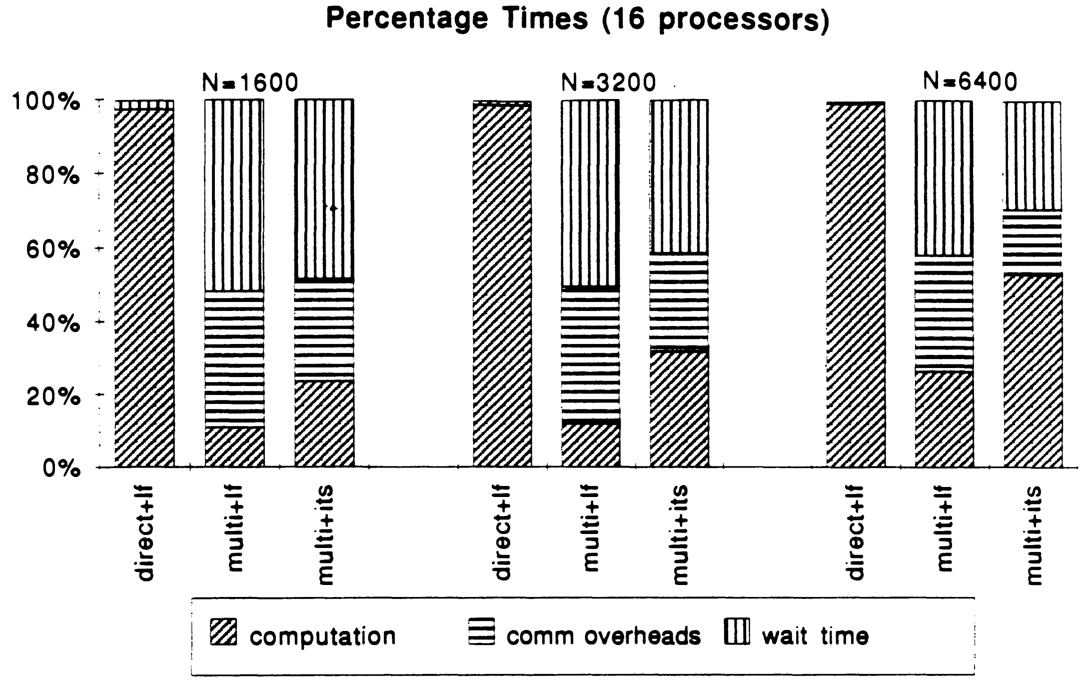


Figure 8.9: T_{comp}, T_{wait} and T_{OH} for various configurations on 16 processors

centage of its time communicating but this percentage goes down as the number of particles goes up. Contrast this with the corresponding times taken by 4 processors as listed in Table 8.8.

8.4.2 Scaling with Number of Processors

Figure 8.10 compares the compute/communicate ratio for 16 processors with that for 4 processors. The 4-processor version spends more than 90% of its time computing. Part of the reason for this is that during initialization the shared time-step algorithm is used which is very communication intensive. With 4 processors, all processors being adjacent to each other, this cost does not show up. This initial communication cost can be estimated using the communication costs of the shared time-step multipole algorithm in the table as ≈ 4.48 for 800 particles. Subtracting this out, we obtain a much smaller value of 2.09 as the cost of communication for 800 particles on 16 processors. The second reason is that with 4 processors, each

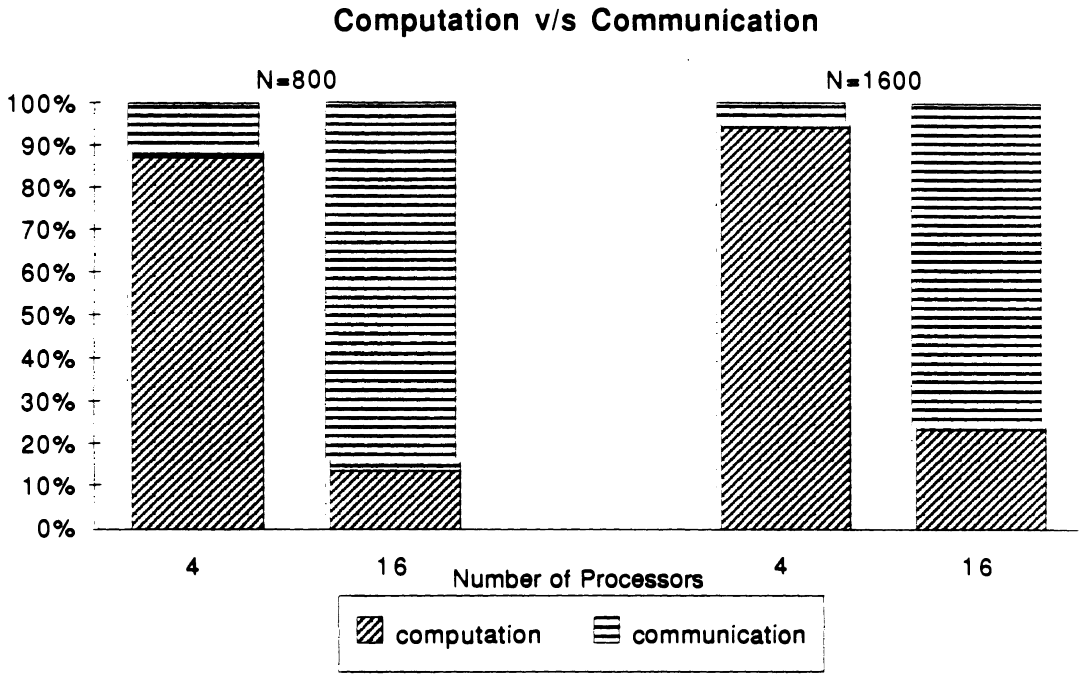


Figure 8.10: Computation v/s Communication on 4 and 16 processors

processor has at most 3 neighbors while with 16, some have as many as 8 neighbors. Figure 8.11 shows how the time taken scales with the number of processors. The cost of communication seems to be increasing with the number of processors. However, the local nature of the algorithm guarantees that the communication cost will become constant when most processors have 8 neighbors and the integration is performed long enough to amortize initialization costs. Beyond this point, performance will increase almost linearly (remember the logarithmic factor) with the number of processors. For the sake of completeness, computation times for 1 processor are given in Table 8.9.

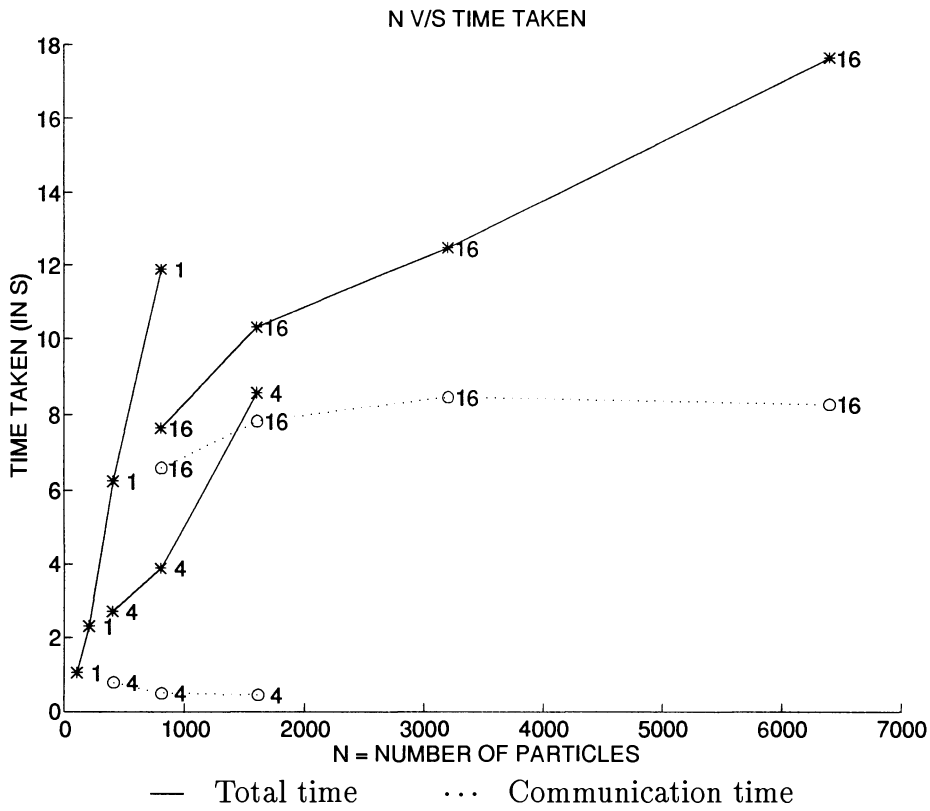


Figure 8.11: Time taken by temporal multipole algorithm on 1,4 and 16 processors

Table 8.3: Time taken (in s) for different algorithms: RANDOM

N[a]		Direct		Multi		Barnes_Hut	
		Shared	Individual	Shared	Individual	Shared	Individual
100	T[b]	1.29	0.58	1.49	0.48	1.96	0.66
	E[c]	4.45e-5	3.55e-6	4.45e-5	3.55e-6	4.45e-5	3.55e-6
	M[d]	1.81e-18	4.10e-18	2.71e-18	1.34e-18	1.1e-8	1.15e-8
200	T	4.74	2.68	3.68	2.24	8.33	3.78
	E	1.78e-5	4.00e-6	1.78e-5	3.93e-6	1.78e-5	3.93e-6
	M	5.42e-19	3.53e-4	2.53e-8	4.00e-3	7.1254e-8	4.00e-3
500	T	29.01	8.18	14.54	4.69	32.03	9.51
	E	1.04e-5	8.34e-7	1.04e-5	8.36e-7	1.04e-5	8.34e-7
	M	7.58e-19	6.89e-8	5.40e-9	6.35e-8	3.04e-8	6.40e-8
1000	T	140.02	38.04	26.34	8.44	84.21	25.33
	E	1.03e-5	8.23e-7	1.03e-5	8.24e-7	1.03e-5	8.23e-7
	M	9.21e-19	3.82e-9	3.76e-9	9.91e-9	1.16e-8	1.78e-8
2000	T	-	-	75.74	22.74	-	65.46
	E	-	-	2.56e-6	2.05e-7	-	2.05e-7
	M	-	-	3.07e-9	6.21e-9	-	6.38e-9
4000	T	-	-	111.01	35.09	-	194.29
	E	-	-	2.54e-6	2.03e-7	-	2.03e-7
	M	-	-	1.53e-9	1.39e-9	-	3.83e-9

[a]Number of particles
[b]Time taken (in s)
[c]Energy conservation: $|\frac{\Delta E}{E}|$
[d]Momentum conservation: $|\Delta \sum_i m_i v_i|$

Table 8.8: Time taken (in s) for different algorithms on 4 Processors

Number of Particles	Direct Shared			Multi Shared			Multi Individual		
	T_{comp}	T_{wait}	T_{OH}	T_{comp}	T_{wait}	T_{OH}	T_{comp}	T_{wait}	T_{OH}
400	16.78	0.33	0.02	5.01	2.94	0.87	1.93	0.50	0.28
800	68.82	0.64	0.02	7.36	2.73	0.87	3.40	0.23	0.25
1600	276.00	1.23	0.02	22.41	1.56	1.25	8.13	0.14	0.30

Table 8.9: Time taken (in s) for different algorithms on 1 Processor

Number of Particles	Direct Shared	Multi Shared	Multi Individual
	T_{comp}	T_{comp}	T_{comp}
100	8.04	2.49	1.05
200	33.73	5.65	2.30
400	136.75	17.41	6.23

Appendix A

Force-determined Time-steps

A.1 Pathology with Force-determined Time-steps

Consider the system of 4 particles shown in figure A.1. The velocity of m is much larger than that of $m1$ or $m2$. The Y-component of the forces exerted by $m1$ and $m2$ on m cancel each other out. Further, in traveling distance r, the force on m changes negligibly since M and R are both very large. In short, the force on m is large and the derivative of the force on m is small. The time-step determined for m using force derivatives is very large and m "whizzes" past $m1$ and $m2$ disregarding a possible close encounter.

Now, the positions of $m1$ and $m2$ can be adjusted in such a way that the Y-component of the force exerted by $m1$ on m does not exactly cancel the force exerted by $m2$ on m. The difference is insignificant at distance r. When m is much closer, however, the difference in their forces on m becomes appreciable and can change the trajectory of m. With force-determined time-steps, this effect will not be discovered by the algorithm. The position oriented scheme, on the other hand, will correctly discover this problem. In a normal N-body simulation such situations will occur with very low probability.

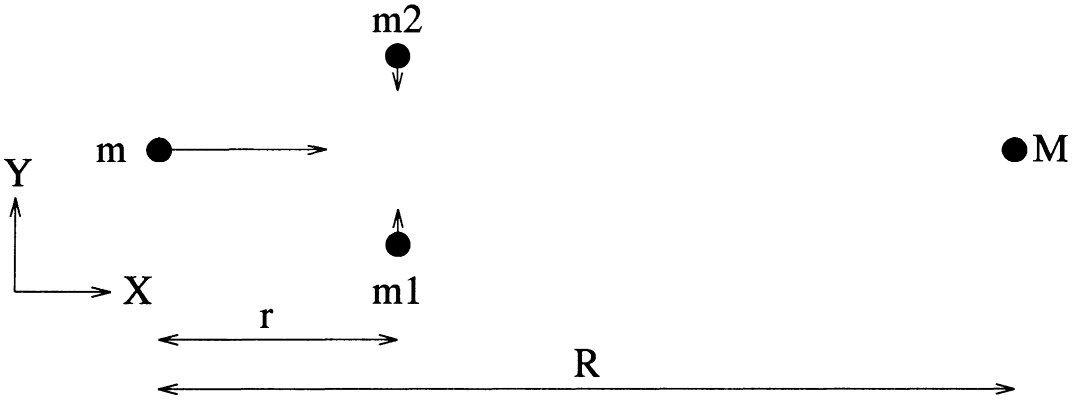


Figure A.1: Pathology with Force-determined Time-steps
$m \ll M$ *and* $r \ll R$
Arrow at each particle indicates velocity (magnitude and direction).

Bibliography

[Aar63] Sverre J. Aarseth. Dynamical evolution of clusters of galaxies, I. *Monthly Notices of the Royal Astronomical Society*, 126:223, 1963.

[Aar85] Sverre J. Aarseth. Direct methods for N-body simulations. In Jeremiah U. Brackbill and Bruce I. Cohen, editors, *Multiple Time Scales*, pages 377–418. Orlando: Academic Press, 1985.

[AC73] A. Ahmad and L. Cohen. N-body gravitational problem. *Journal of Computational Physics*, 12:389–402, 1973.

[App85] A. Appel. An efficient program for many-body simulation. *Siam Journal on Scientific and Statistical Computing*, 6(1):85–103, 1985.

[Bar90] Joshua E. Barnes. A modified tree code - dont laugh - it runs. *Journal of Computational Physics*, 87(1):161–170, 1990.

[BH86] Joshua Barnes and Piet Hut. A hierarchical $O(N \log N)$ force calculation algorithm. *Nature*, 324:446–449, 1986.

[Edw92] Joseph Edwards. *An Elementary Treatise on the Differential Calculus.* New York: MacMillan and Co., 1892.

[GG90] L. Greengard and W. D. Gropp. A parallel version of the fast multipole algorithm. *Computers and Mathematics with Applications*, 20(7):63–71, 1990.

[GH83] J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcation of Vector Fields.* New York: Springer-Verlag, 1983.

[GR87] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, 1987.

[Gre87] Leslie Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems.* Ph.D. dissertation, Yale University, 1987.

[HE89] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles.* New York: A. Hilger, 1989.

[Her90] Lars Hernquist. Vectorization of tree traversals. *Journal of Computational Physics*, 87(1):137–147, 1990.

[HJ87] C. T. Ho and S. L. Johnsson. On the embedding of arbitrary meshes in boolean cubes with expansion two dilation two. In *International Conference on Parallel Processing*, pages 188–191, 1987.

[HJ90] C. T. Ho and S. L. Johnsson. Embedding meshes in boolean cubes by graph decomposition. *Journal of Parallel and Distributed Computing*, 8:325–339, 1990.

[JP89] J. G. Jernighan and D. H. Porter. A tree code with logarithmic reduction of force terms, hierarchical regularization of all variables and explicit accuracy controls. *Astrophysical Journal Supplement Series*, 71(817), 1989.

[Kat88] J. Katzenelson. Computational structure of the N-body problem. Technical Report AI Memo 1042, AI Lab, MIT, 1988.

[LB91] J. F. Leathrum, Jr. and J. A. Board, Jr. Parallelization of the fast multipole algorithm using the b012 transputer network. *Transputing 91. Proceedings of the World Transputer User Group (WOTUG) Conference*, pages 296–310, 1991.

[Lei92] F. Thomson Leighton. *Introdution to Parallel Algorithms and Architectures*, volume 1, pages 234–235. California: Morgan Kauffman, 1992.

[MA93] Stephen L. W. McMillan and Sverre J. Aarseth. An $O(N \log N)$ integration scheme for collisional stellar systems. *Preprint*, 1993.

[Mak90] Junichiro Makino. Vectorization of a treecode. *Journal of Computational Physics*, 87(1):148–160, 1990.

[McM86] Stephen L. W. McMillan. The vectorization of small-N integrators. In P. Hut and S. L. W. McMillan, editors, *The Use of Supercomputers in Stellar Dynamics*, page 156. New York: Springer-Verlag, 1986.

[MH88] Junichiro Makino and Piet Hut. Performance analysis of direct N-body calculations. *The Astrophysical Journal Supplement Series*, 68:833–856, 1988.

[MH89] Junichiro Makino and Piet Hut. Gravitational N-body algorithms - a comparison between supercomputers and a highly parallel computer. *Computer Physics Reports*, 9(4):199–246, 1989.

[OME+92] S.K. Okumura, J. Makino, T. Ebisuzaki, T. Ito, T. Fukushige, D. Sugimoto, E. Hashimoto, K. Tomida, and N. Miyakawa. Grape-3: Highly parallelized special-purpose computer for gravitational many-body simulations. In V. Milutinovic, B.D. Shriver, J.F. Nunamaker, and R.H. Sprague, editors, *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, volume 25, pages 151–160, 1992.

[RT93] John H. Reif and Stephen R. Tate. The complexity of N-body simulation. In *International Colloquium on Automata, Languages and Programming*, 1993.

[Sal91] John K. Salmon. *Parallel Hierarchical N-Body Methods.* Ph.D. dissertation, California Institute of Technology, 1991.

[SHT+92] Jaswinder Pal Singh, Chris Holt, Takashi Totsuka, Anoop Gupta, and John L. Hennessy. Load balancing and data locality in hierarchical N-body methods. Technical Report CSL-TR-92-505, Computer Systems Laboratory, Stanford University, 1992.

[SL91] K. E. Schmidt and M. A. Lee. Implementing the fast multipole algorithm in 3 dimensions. *Journal of Statistical Physics*, 63(5-6):1223–1235, 1991.

[Sun92] Sridhar Sundaram. Periodic updates in processor networks. Technical Report TR 92-1295, Computer Science Department, Cornell University, 1992.

[vH60] S. von Hoerner. *Z. Astrophys.*, 50:184–214, 1960.

[Zha87] Feng Zhao. An $O(N)$ algorithm for three-dimensional N-body simulations. Masters dissertation, MIT, 1987.

[ZJ91] Feng Zhao and S. L. Johnsson. The parallel multipole method on the connection machine. *Siam Journal on Scientific and Statistical Computing*, 12(6):1420–37, 1991.