ABSTRACT

Title of Dissertation: FAST TRANSFORMS BASED ON STRUCTURED MATRICES WITH APPLICATIONS TO THE FAST MULTIPOLE METHOD

Zhihui Tang, Doctor of Philosophy, 2003

Dissertation directed by: Professor Ramani Duraiswami
Applied Mathematics and Scientific Computation

The solution of many problems in engineering and science is enabled by the availability of a fast algorithm, a significant example being the fast Fourier transform, which computes the matrix-vector product for a $N \times N$ Fourier matrix in $O(N \log(N))$ time. Related fast algorithms have been devised since to evaluate matrix-vector products for other structured matrices such as matrices with Toeplitz, Hankel, Vandermonde, etc. structure.

A recent fast algorithm that was developed is the fast multipole method (FMM). The original FMM evaluates all pair-wise interactions in large ensembles of N particles in $O(p^2 N)$ time, where p is the number of terms in the truncated multipole/local expansions it uses. Analytical properties of translation operators that shift the center of a multipole or local expansion to another location

and convert a multipole expansion into a local expansion are used. The original translation operators achieve the translation in $O(p^2)$ operations for a p term expansion. Translation operations are among the most important and expensive steps in an FMM algorithm. The main focus of this dissertation is on developing fast accurate algorithms for the translation operators in the FMM for Coulombic potentials in two or three dimensions.

We show that the matrices involved in the translation operators of the FMM for Coulombic potentials can be expressed as products of structured matrices. Some of these matrices have fast transform algorithms available, while for others we show how they can be constructed. A particular algorithm we develop is for fast computation of matrix vector products of the form Px, $P'x$, and $PP'x$, where P is a Pascal matrix.

When considering fast translation algorithms for the 3D FMM we decompose translations into an axial translation and a pair of rotations. We show how a fast axial translation can be performed. The bottleneck for achieving fast translation is presented by the lack of a fast rotation transform. A fast rotation algorithm is also important for many other applications, including quantum mechanics, geoscience, computer vision, etc, and fast rotation algorithms are being developed based on the properties of spherical harmonics. We follow an alternate path by showing that the rotation matrix R can be factored in two different ways into the product of structured matrices. Both factorizations allow a fast matrix-vector product. Our algorithm efficiently computes the coefficients of spherical harmonic expansions on rotation.

Numerical experiments confirm that the new $O(p \log p)$ translation operators for both the 2D and 3D FMM have the same accuracy as the original ones, achieve

their asymptotic complexity for modest p, and significantly speed up the FMM algorithms in 2D and 3D. We hope that this thesis will also lead to promising future research in establishing fast translation for the FMM for other potentials, as well as applying the transforms in other applications such as in harmonic analysis on the sphere.

FAST TRANSFORMS BASED ON STRUCTURED MATRICES WITH APPLICATIONS TO THE FAST MULTIPOLE METHOD

by

Zhihui Tang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland at College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2003

Advisory Committee:

Professor Howard Elman, Chairman
Professor Ramani Duraiswami, Advisor
Professor Larry S. Davis
Professor Nail A. Gumerov
Professor Dennis M. Healy
Professor Ricardo H. Nochetto

DEDICATION

To My Parents

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to my adviser, Ramani Duraiswami, for providing me help, support, encouragement, and allowing me to pursue my own ideas, even when they are not so close to the funded research project. I am also very grateful to my committee chairman Howard Elman, who has given me sage advice, help, and support. I would like to thank my committee members Larry Davis, Nail Gumerov, Dennis Healy, Ricardo Nochetto for their help. Jingfang Huang from the University of Carolina, and James Kelly from the department of physics at the UMCP have helped me find the right references from the endless literature. I would like to thank Professors Mark Freidlin, Daniel Rudolph, Paul Smith, Eitan Tadmor for their help and support. I would also like to thank all my friends, especially, Kexue Liu. Finally, I would like to thank my family for their love and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

The Fast Multipole Method (FMM) is an algorithm originally proposed by Rokhlin [Rokhlin83] as a fast scheme for accelerating the numerical solution of the Laplace equation in two dimensions. It was further improved by Greengard and Rokhlin when it was applied to particle simulations [Greengard87, Greengard88]. It evaluates pair wise interactions (force or potential) in large ensembles of N particles in $O(N)$ time. This is an improvement over the $O(N^2)$ time required by direct methods. Since the invention of the FMM, It has been used in a wide variety of applications , such as computational astronomy, molecular dynamics, fluid dynamics, radar scattering, etc.

The FMM combines several ideas in harmonic analysis, series expansions, multiscale analysis based on hierarchical decomposition of space, and translation operators to achieve its efficiency. Usually the force or potential of the field charges are expressed as truncated multipole or local expansions with p terms, where p depends on the desired precision. It achieves linear complexity through the summation of long range interactions by using multipole and local expansions. It relies on analytical properties of translation operators that shift the center of multipole or local expansion to another location and convert a multipole expan-

sion into a local expansion. Translations of various types are the most important and expensive steps in an FMM algorithm. Straightforward translation operators achieve the translation in $O(p^2)$ operations for a p term expansion. The cost of this operation has been the main obstacle to performance of most existing FMM implementations.

Several researches have addressed this difficulty to increase the efficiency of the FMM. Petersen et al [Petersen94] have pointed out that since the convergence rate of an expansion depends on the distance of translation, the number p of terms in the expansions should not be kept constant in the process of the translation of the expansions. For example, a smaller number p of terms could be used when the translation is done for distant boxes in the interaction list. This observation leads to a procedure that they call the "very fast multipole method" (VFMM) which continues to use $O(p^2)$ translation operators. Elliot and Board have used fast Fourier transform to reduce the cost of the multipole to local translation step from $O(p^2)$ to $O(p \log p)$ [Elliott96]. However, this method incurs a big increase in memory requirement and stability problems, and requires substantial modification of a standard FMM implementations [White96]. Another approach suggested by White and Head-Gordon achieves $O(p^{3/2})$ complexity by rotating the coordinates such that the translation is always along the z axis [White96]. The implementation of this procedure is straightforward and it reduces memory usage, but is still sub-optimal. The most successful approach is by Rokhlin, Greengard, et al in [Cheng99, Greengard97, Hrycak98]. They build approximate diagonal operators to reduce the cost of translations. This method is highly effective, it reduces the cost of the largest part of the multipole-to-local translation. However, these still achieve translations with an asymptotic complexity of $O(p^{3/2})$ with a

smaller coefficient, and furthermore, they are complicated to implement.

In this dissertation, we present fast, efficient, and accurate translation operators for the potentials $\phi_{X_0}(X) = -\log(||X - X_0||)$ and $\frac{1}{||X-X_0||}$ that are governed by, respectively, the two and three dimensional Laplace equations,

$$\nabla^2\phi(x) = \frac{\partial^2\phi}{\partial x_i \partial x_i} = 0. \tag{1.1}$$

The potentials are usually referred to as Coulombic or gravitational potentials. They are closely related to a number of important problems such as celestial mechanics, plasma physics, fluid dynamics, and molecular dynamics.

1.1 Main ideas and results

For potentials governed by the Laplace equation, we present new factored forms for the translational operators as products of diagonal matrices and constant matrices. These constant matrices, which are independent of the translation distance, can further be decomposed as products of "matrices with structure" or "structured matrices". This allows fast matrix-vector multiplication. The resulting constant matrices from the translation operators can be very helpful in bringing down the cost of FMM computations.

Since our decomposition includes a constant part, we can think of a number of different strategies to exploit them. This can open some possible future research opportunities to find the best factorization. We present several different factorizations of these matrices and have implemented ones that are free of numerical instabilities. More specifically, in two dimensions, the constant matrices from the translation operators are the Pascal matrix, its transpose, or the product of the Pascal matrix and its transpose. They are further factored as the product of

diagonal matrices and Toeplitz (or Hankel) matrices. In three dimensions, it is known that the translation operators can be factored as the product of rotation matrices and coaxial translation matrices [White96, Greengard97]. We, for the first time, further factor the coaxial translation matrices into products of diagonal matrices and constant matrices. The constant matrices are factored into structured matrices such as Toeplitz matrices or Hankel matrices in a similar way to the two dimensional case. For the rotation matrices, it is known that matrices of this type can be factored into the product of constant matrices and diagonal matrices [Edmonds60]. We again find ways to factor these constant matrices arising from the rotation matrices into products of structured matrices. These representations allow multiplication with these matrices in $O(p \log p)$ operations.

This rotation matrix is extensively used in areas such as quantum mechanics, geoscience, computational biology, computer vision, etc. A new fast rotation transform, based on this factorization, could be very useful to speed up computations in these applications.

1.2 Outline of the dissertation

The dissertation is organized in the following way:

Chapter 2 introduces fast matrix-vector products for structured matrices, and briefly explains the ideas behind the fast algorithms for these matrices. It thus provides necessary background information. There exist known issues regarding numerical stability problem for some of the fast matrix-vector product algorithms. We delay the discussion of this issue and will devote a whole chapter, Chapter 9, to discuss techniques and implementation details that solve the stability issues that may arise in our algorithms.

Chapter 3 develops fast algorithms for matrix-vector products, Px,$P'x$,$(PP')x$, where P is a Pascal matrix, P' is its transpose, and x is a vector.

Chapters 4 and 5 deal with potential problems in two dimensional space. More specifically, Chapter 4 explains background material including potential fields in 2D, related multipole expansions, and translation operators and their matrix forms. It also gives a formal description of FMM algorithm and its detailed complexity analysis, motivates the need to develop fast efficient translation operators, and describes previous work that has been done on the translation operators.

Chapter 5 develops a few new factored forms for the translation operators that enable us to do fast translation. The chapter ends with a complexity analysis of these more efficient operators.

Chapter 6 deals with potential problems in three dimensional space. It presents all the background material needed by the FMM including what is a multipole expansion of a potential field, translation operators, error bound of the truncated translation operators. It describes the difference between the FMM in 3D and that in 2D, motivates the need to develop fast translation operators, and describes previous work that has been done on the translation operators in 3D, including exponential expansions based translation, and rotation based translations.

Chapter 7 introduces the rotation transform of the spherical harmonics expansion and reviews the previous work. It then presents two new different factorizations of the rotation matrix. From one of the factorization, it develops a fast algorithm for the rotation transformation. It presents a complexity analysis of the rotation transform at the end.

Chapter 8 presents a few new factored forms of the coaxial translation opera-

tors. These factorizations combined with the fast rotation transform lead to fast translation operators. The chapter ends with complexity analysis of the FMM with the new translation operators.

Chapter 9 presents implementation details of the new efficient operators in 2D and 3D to achieve accuracy, stability, and efficiency.

Chapter 10 presents numerical results to demonstrate the actual performance of the new translation operators.

Chapter 11 reaches conclusions and discusses future work.

Chapter 2

Fast Matrix-Vector Product for Structured Matrices

In the process of developing new fast algorithms for translation operators and rotation operators, we have encountered the task of doing fast matrix-vector product for Toeplitz matrices, Hankel matrices, and Vandermonde matrices, which belong to a big class of matrices, namely, the structured matrices. In this chapter we introduce some fast algorithms on the matrix-vector product for these matrices. We will give the complexity of these algorithms, make comments on the stability of Vandermonde matrices and leave the stability issues of other matrices to a later chapter, Chapter 9.

The multiplication of a matrix and a vector arises in many problems in engineering and applied mathematics. For a dense matrix A of size $n \times n$, to compute its product Ax with an arbitrary input vector x requires $O(n^2)$ work by standard matrix-vector multiplication. In many applications, n is very large, and moreover, for the same matrix, the multiplication has to be done over and over again with different input vectors, for example, in iterative methods for solving linear systems. In such cases, one seeks in various classes of applications to identify

special properties of the matrices in order to reduce the computational work. One special class of matrices are the structured matrices. They often appear in communications, control, optimization, and signal processing, etc. The multiplication of any of these matrices with any arbitrary input vector can often be done in $O(n \log^k n)$ time, where usually $0 \leq k \leq 2$, depending on the structure.

Definition 2.1. *A dense matrix of order $n \times n$ is called structured if its entries depend on only $O(n)$ parameters.*

Examples of structured matrices include Fourier matrices, Circulant matrices, Toeplitz matrices, Hankel matrices, Vandermonde matrices, etc.

In this chapter we will collect some known useful results about them. It is very interesting to observe that the main focus of this dissertation, the fast multipole method, is an efficient algorithm for a class of structured matrices, the entries of which depend on function of two set of $O(N)$ points; however, the translation operators, one of the core elements of the fast multipole method rely on algorithms for these structured matrices.

2.1 Fourier matrices

The most important class of matrices in all fast algorithms are the Fourier matrices.

Definition 2.2. *A Fourier matrix of order n is defined as the following*

$$F_n = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix}, \tag{2.1}$$

where

$$\omega_n = e^{-\frac{2\pi i}{n}}, \tag{2.2}$$

is an nth root of unity.

It is well known that the product of this matrix with any vector is the so-called discrete Fourier transform, which can be done efficiently using the so-called fast Fourier transform (FFT) algorithm [Cooley65]. Notice that Fourier matrix is a unitary matrix, that is, $F_n F_n^* = I$, therefore, the conjugate transpose F_n^* is also a unitary matrix. The corresponding efficient matrix-vector product is the inverse fast Fourier transform (IFFT) [Van92, Golub96].

Theorem 2.3. *The FFT and IFFT can be done in $O(n \log n)$ time.*

A proof can be found in [Cooley65] or [Van92].

This theorem is the basis for a number of other efficient algorithms, for example, the product of a circulant matrix and a vector.

2.2 Circulant matrices

Definition 2.4. *A matrix of the form*

$$C_n = C(x_1, ..., x_n) = \begin{bmatrix} x_1 & x_n & x_{n-1} & \cdots & x_2 \\ x_2 & x_1 & x_n & \cdots & x_3 \\ x_3 & x_2 & x_1 & \cdots & x_4 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_n & x_{n-1} & x_{n-2} & \cdots & x_1 \end{bmatrix} \tag{2.3}$$

is called a circulant matrix.

It is easy to see that a circulant matrix is completely determined by the entries in the first column. All other columns are a shift of the previous column. It has the following important property.

Theorem 2.5. *Circulant matrices $C_n(x)$ can be diagonalized by the Fourier matrix,*

$$C_n(x) = F_n^* \cdot diag(F_n x) \cdot F_n, \tag{2.4}$$

where $x = (x_1, ..., x_n)'$.

A proof can be found in [Bai2000]. Given this theorem, we can easily have the following fast algorithm.

Given a circulant matrix C_n, and a vector y, the product

$$C_n y \tag{2.5}$$

can be computed efficiently in the following four steps:

1. compute f =FFT(y),

2. compute g =FFT(x),

3. compute the element wise vector-vector product $h = f. * g$,

4. compute z =IFFT(h) to obtain $C_n y$

Since the FFT and the IFFT can be done in $O(n \log n)$ time, $C_n y$ can be obtained in $O(n \log n)$ time [Bai2000, Lu98].

2.3 Toeplitz matrices

After we know how to do the fast matrix-vector product for circulant matrices, it is easy to see the algorithm for the Toeplitz matrix, since a Toeplitz matrix can be embedded into a circulant matrix. As we mentioned in the beginning of the chapter, we will need to compute the product of a Toeplitz matrix and a vector fast in later chapters 3, 5, 7, 8 to develop new fast algorithms on fast translation operators and rotation operators.

Definition 2.6. *A matrix of the form*

$$T_n = T(x_{-n+1}, \cdots, x_0, ..., x_{n-1}) = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_{n-1} \\ x_{-1} & x_0 & x_1 & \cdots & x_{n-2} \\ x_{-2} & x_{-1} & x_0 & \cdots & x_{n-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_{-n+1} & x_{-n+2} & x_{-n+3} & \cdots & x_0 \end{bmatrix} \quad (2.6)$$

is called a Toeplitz matrix.

A Toeplitz matrix is completely determined by its first column and first row. The entries of T_n are constant down the diagonals parallel to the main diagonal. It arises naturally in problems involving trigonometric moments. Sometimes we

denote a Toeplitz matrix with first column vector

$$c = \begin{bmatrix} c_0 & c_1 & c_2 & ... & c_{p-1} \end{bmatrix}' \tag{2.7}$$

and first row vector

$$r = \begin{bmatrix} c_0 & r_1 & r_2 & ... & r_{p-1} \end{bmatrix} \tag{2.8}$$

by

$$Toep(c, r') = Toep \begin{bmatrix} c_0 & c_0 \\ c_1 & r_1 \\ c_2 & , \quad r_2 \\ \vdots & \vdots \\ c_{p-1} & r_{p-1} \end{bmatrix} \tag{2.9}$$

One important property of Toeplitz matrices is described in the following theorem [Bai2000, Kailath99].

Theorem 2.7. *The product of any Toeplitz matrix and any vector can be done in $O(n \log n)$ time.*

Proof. Given a Toeplitz matrix T_n and a vector y, to compute the product $T_n y$, a Toeplitz matrix can first be embedded into a $2n \times 2n$ circulant matrix C_{2n} as follows

$$C_{2n} = \begin{bmatrix} T_n & S_n \\ S_n & T_n \end{bmatrix}, \tag{2.10}$$

where

$$S_n = \begin{bmatrix} 0 & x_{-n+1} & x_{-n+2} & \cdots & x_{-1} \\ x_{n-1} & 0 & x_{-n+1} & \cdots & x_{-2} \\ x_{n-2} & x_{n-1} & 0 & \cdots & x_{-3} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_1 & x_2 & x_3 & \cdots & 0 \end{bmatrix}. \tag{2.11}$$

Then $T_n y$ can be multiplied as

$$C_{2n} \cdot \begin{bmatrix} y \\ 0_{n\times n} \end{bmatrix} = \begin{bmatrix} T_n & S_n \\ S_n & T_n \end{bmatrix} \cdot \begin{bmatrix} y \\ 0_{n\times n} \end{bmatrix} = \begin{bmatrix} T_n y \\ S_n y \end{bmatrix}, \tag{2.12}$$

which can be implemented to be done in $O(n \log n)$ time. ■

The way to compute the product of a Toeplitz matrix and a vector fast is clear from the above proof. In later Chapters 3, 5, 7, and 8, we will repeatedly use this property of a Toeplitz matrix to build efficient translation operators. We will address later in Chapter 9 the associated numerical instability problems for the product of this matrix and a vector.

2.4 Hankel matrices

Definition 2.8. *A matrix of the form*

$$H_n = H(x_{-n+1}, \cdots, x_0, \cdots, x_{n-1}) = \begin{bmatrix} x_{-n+1} & x_{-n+2} & x_{-n+3} & \cdots & x_0 \\ x_{-n+2} & x_{-n+3} & x_{-n+4} & \cdots & x_1 \\ x_{-n+3} & x_{-n+4} & x_{-n+5} & \cdots & x_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ x_0 & x_1 & x_2 & \cdots & x_{n-1} \end{bmatrix} \tag{2.13}$$

is called a Hankel matrix.

A Hankel is completely determined by its first column and last row. The entries of T_n are constant along the diagonals that are perpendicular to the main diagonal. It arises naturally in problems involving power moments. It has the following property [Golub96].

Theorem 2.9. *The product of any Hankel matrix and any vector can be done in* $O(n \log n)$ *time.*

Proof. Notice that if

$$I_p = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}, \tag{2.14}$$

is the backward identity permutation matrix, then $I_p H_n$ is a Toeplitz matrix for any Hankel matrix H_n, and $I_p T_n$ is a Hankel matrix for any Toeplitz matrix T_n. The product $H_n y$ for any vector y can be computed as follows [Kailath99]: first compute the product $(I_p H_n)\dot{y}$ of a Toeplitz matrix $I_p H_n$ and vector y as in (2.12), then apply the permutation to the vector $(I_p H_n)\dot{y}$ to have $P(P H_n)\dot{y}$, which is what we want since $I_p = I_p^t = I_p^{-1}$. ■

In Chapters 3, 5, and 8, we will repeatedly use this property of a Hankel matrix to build efficient translation operators. The fast computation of the product of a Hankel matrix and a vector is clear from the above proof.

2.5 Vandermonde matrices

Definition 2.10. *Suppose* $\{x_i,\ i = 0, 1, ..n\} \in \mathbb{C}^{n+1}$*, a matrix of the form*

$$V = V(x_0, x_1, ..., x_n) = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_0 & x_1 & \cdots & x_n \\ \cdots & \cdots & \cdots & \cdots \\ x_0^n & x_1^n & \cdots & x_n^n \end{bmatrix} \tag{2.15}$$

is called a Vandermonde matrix.

A Vandermonde matrix is completely determined by its second row. All rows are powers of the second row from the power 0 to power n. It is a fact that

$$\det A = \prod_{i,j=0,i>j}^{n} (x_i - x_j) \tag{2.16}$$

so a Vandermonde matrix is nonsingular if and only if the $(n+1)$ parameters $x_0, x_1, ..., x_n$ are distinct. In this dissertation we impose this requirement whenever we need the inverse of this matrix.

A Fourier matrix is a special case of Vandermonde matrix. Its transpose arises naturally in polynomial evaluations or polynomial interpolations. There exist efficient algorithms for fast matrix-vector product for a Vandermonde matrix, its transpose, its inverse, and the transpose of its inverse. All of them are of complexity $O(n \log^2 n)$, although there are associated stability problems [Driscoll97, Moore93]. The basic idea is to factor the matrices into products of sparse matrices, Toeplitz matrices and the like, so that the FFT can be applied to speed up the computations. We state these facts as a theorem below. The details can be found in [Driscoll97, GohbergL94, GohbergC94, Lu98, Moore93, Pan92].

Theorem 2.11. *The product of any Vandermonde matrix, its transpose, its inverse, or the transpose of its inverses with any vector is of complexity* $O(n \log^2 n)$.

In our later chapters, we give representations of the translation operators in factored forms in terms of Vandermonde matrices. There exist a number of algorithms for the product of a Vandermonde matrix and a vector and techniques to overcome the instability problems associated with the algorithm [Driscoll97, Moore93]. However, in this dissertation we do not use those factored repre-

sentations involving Vandermonde matrices in our implementations due to the complexity and instability of the algorithms.

Chapter 3

Fast Algorithms for Matrix-vector Products of the Pascal Matrix and its Relatives

In this chapter we present a few fast methods to compute the product of a Pascal matrix or its related matrix and a vector, which is crucial for developing fast algorithms for translation operators in FMM. We will postpone the discussion of the stability problem related to fast translation operators until Chapter 9.

3.1 Pascal matrix

The Pascal triangle arises in binomial expansion, probability, combinatorics and is familiar since high school . It also arises naturally in this thesis in our development of fast translation operators in 2D and rotation operators in 3D. Pascal was not the first to create his triangle. It has been discovered in China, Europe, and India. In China, it has been known as "Yang Hui's triangle" .

Definition 3.1. *A Pascal matrix is of the form*

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 2 & 1 & 0 & \cdots & 0 \\ 1 & 3 & 3 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{p-1}^0 & C_{p-1}^1 & C_{p-1}^2 & C_{p-1}^3 & \cdots & C_{p-1}^{p-1} \end{bmatrix}, \tag{3.1}$$

where $C_n^m = \frac{n!}{(n-m)!m!}$ *is the binomial coefficient.*

Notice that the entries in Pascal matrix are those in Pascal triangle; they are also coefficients of the binomial expansion.

3.2 Decomposition of Pascal matrix

It is easy to verify the following identity which we will use in our implementation of the fast algorithms.

Theorem 3.2. *The Pascal matrix* $\mathbf{P}$ *can be decomposed as,*

$$\mathbf{P} = diag(v_1) \cdot T \cdot diag(v_2), \tag{3.2}$$

where vectors

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 2! \\ 3! \\ \vdots \\ (p-1)! \end{bmatrix}, \quad v_2 = \begin{bmatrix} 1 \\ \frac{1}{1!} \\ \frac{1}{2!} \\ \frac{1}{3!} \\ \vdots \\ \frac{1}{(p-1)!} \end{bmatrix}, \tag{3.3}$$

and the matrix

$$T = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ \frac{1}{2!} & \frac{1}{1!} & 1 & \cdots & 0 \\ \frac{1}{3!} & \frac{1}{2!} & \frac{1}{1!} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{(p-1)!} & \frac{1}{(p-2)!} & \frac{1}{(p-3)!} & \cdots & 1 \end{bmatrix} \tag{3.4}$$

is a Toeplitz matrix.

Proof. Notice that the (n, m) entry P_{nm} of the Pascal matrix is

$$P_{nm} = \begin{cases} C_{n-1}^{m-1} & \text{if } n \geq m \\ 0 & \text{if } n < m \end{cases}, \tag{3.5}$$

where $C_{n-1}^{m-1} = \frac{(n-1)!}{(n-m)!(m-1)!}$. That is, every entry in n-th row of the Pascal matrix has a common factor $(n-1)!$, and every entry in m-th column of the Pascal matrix has a common factor $\frac{1}{(m-1)!}$. We can take out the common factor $(n-1)!$ of the n-th row and common factor $\frac{1}{(m-1)!}$ of the m-th column, and multiply from left side by a diagonal matrix which is the identity, except that the n-th entry in the diagonal is $(n-1)!$, and multiply from right side by a diagonal matrix which is the identity, except that the m-th entry in the diagonal is $\frac{1}{(m-1)!}$. This can be done for every row and column. Therefore we have factored the Pascal matrix into products of matrices with a Toeplitz matrix T in the middle and p diagonal matrices on the left of T, and p diagonal matrices on the right of T. Multiplying the diagonal matrices on the left and the right respectively, we end up with the diagonal matrices $diag(v_1)$ and $diag(v_2)$. ■

With the notation for Toeplitz matrix introduced earlier, T can be written as

$$T = \text{Toep}\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ \frac{1}{2} & 0 \\ \frac{1}{6} & 0 \\ \vdots & \vdots \\ \frac{1}{(p-1)!} & 0 \end{bmatrix}, \tag{3.6}$$

From this lemma, It is clear that the multiplication of a Pascal matrix P and a vector x can be done in three steps: first calculate the element-wise multiplication of $u = v_1. * x$, which requires p multiplications. Then calculate the product $w = Tu$ of Toeplitz matrix T and vector u as (2.12), which requires $O(p \log p)$ work by theorem 2.7. And finally calculate another element-wise multiplication of $v_1. * w$ to obtain the product Px. Therefore we have the following.

Theorem 3.3. *The multiplication of a $p \times p$ Pascal matrix and a p vector can be done in $O(p \log p)$ operations.*

The properties stated in the above lemma and theorem are repeatedly used in later Chapters 5 and 7 to build efficient translation operators and rotation operators.

While we will use the above decomposition to build fast algorithms for the product of the Pascal matrix and a vector, we found some other ways to factor the matrix which we state here.

3.2.1 Alternate decomposition 1

Lemma 3.4. *The Pascal matrix P can be decomposed as the following,*

$$P = V2 * V1^{-1}, \tag{3.7}$$

where

$$V1 = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ x_1 & x_2 & x_3 & x_4 & \cdots & x_p \\ x_1^2 & x_2^2 & x_3^2 & x_4^2 & \cdots & x_p^2 \\ x_1^3 & x_2^3 & x_3^3 & x_4^3 & \cdots & x_p^3 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{p-1} & x_2^{p-1} & x_3^{p-1} & x_4^{p-1} & \cdots & x_p^{p-1} \end{bmatrix} \tag{3.8}$$

$$V2 = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ (x_1+1) & (x_2+1) & (x_3+1) & \cdots & (x_p+1) \\ (x_1+1)^2 & (x_2+1)^2 & (x_3+1)^2 & \cdots & (x_p+1)^2 \\ (x_1+1)^3 & (x_2+1)^3 & (x_3+1)^3 & \cdots & (x_p+1)^3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (x_1+1)^{p-1} & (x_2+1)^{p-1} & (x_3+1)^{p-1} & \cdots & (x_p+1)^{p-1} \end{bmatrix} \tag{3.9}$$

are Vandermonde matrices, and $\{x_i, i = 1, 2, \cdots, p\}$ *are distinct numbers.*

Proof. Because P is a matrix with binomial coefficients, it is easy to see that

$$PV_1 = V_2. \tag{3.10}$$

Notice that $\{x_i, i = 1, 2, \cdots, p\}$ are distinct numbers and can be arbitrary. Hence V_1 is nonsingular and its inverse exists. Thus we have

$$P = V_2 * V_1^{-1}. \tag{3.11}$$

■

From Theorem 2.11, we know that a Vandermonde matrix and its inverse can be multiplied by vectors in $O(p \log^2 p)$ time. This decomposition also allows fast matrix-vector product for the Pascal matrix. However, it is slower than the previous decomposition. Furthermore, many existing algorithms for Vandermonde

matrices are not stable (see [Moore93] and [Driscoll97]). Therefore we will not use it in this work, instead we leave it to our future work.

3.2.2 Alternate decomposition 2

Lemma 3.5. *A Pascal matrix can be decomposed as the product of* $p-1$ *matrices*

$$P = A_1 * A_2 * \cdots * A_{p-1} \tag{3.12}$$

where

$$A_i = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \cdots & \vdots & \vdots \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 1 & 1 \end{bmatrix} \tag{3.13}$$

with p *number of* 1*'s as its diagonal entries, and* i *number of* 1*'s as its sub-diagonal entries starting from the position* $(p, p-1)$.

Proof. We will prove this by mathematical induction.

It is trivial for $p = 2$.

Now assume that for $p = n$,

$$P^{(n)} = A_1^{(n)} * A_2^{(n)} * \cdots * A_{n-1}^{(n)}, \tag{3.14}$$

where $P^{(n)}$ is the Pascal matrix of size $n \times n$, and $A_i^{(n)}$ is A_i as defined in (3.13) of size $n \times n$. For $p = n + 1$, we need to prove that

$$P^{(n+1)} = A_1^{(n+1)} * A_2^{(n+1)} * \cdots * A_n^{(n+1)}. \tag{3.15}$$

It is easy to see that

$$A_i^{(n+1)} = \begin{bmatrix} 1 & 0 \\ 0 & A_i^{(n)} \end{bmatrix} \quad \text{for } i = 1, 2, \cdots, n-1. \tag{3.16}$$

That is, we need to prove

$$P^{(n+1)} = \begin{bmatrix} 1 & 0 \\ 0 & A_1^{(n)} \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & A_2^{(n)} \end{bmatrix} * \cdots * \begin{bmatrix} 1 & 0 \\ 0 & A_{n-1}^{(n)} \end{bmatrix} * A_n^{(n+1)}. \tag{3.17}$$

By assumption, we have

$$\begin{bmatrix} 1 & 0 \\ 0 & A_1^{(n)} \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & A_2^{(n)} \end{bmatrix} * \cdots * \begin{bmatrix} 1 & 0 \\ 0 & A_{n-1}^{(n)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & P^{(n)} \end{bmatrix}. \tag{3.18}$$

Therefore, we need only to prove that

$$P^{(n+1)} = \begin{bmatrix} 1 & 0 \\ 0 & P^{(n)} \end{bmatrix} * A_n^{(n+1)}. \tag{3.19}$$

It is easy to see that the entries in the first column of both sides are one's, the entries in the first row of both sides are the same. For all other entries, we need to prove that

$$P_{ij}^{(n+1)} = P_{(i-1)(j-1)}^{(n)} + P_{(i-1)j}^{(n)} \tag{3.20}$$

This is trivial for all entries in the upper triangular part of the matrices since all entries are zeroes. This is also true for the entries of the diagonal since $P_{(i-1)j}^{(n)} = 0$, and $P_{ij}^{(n+1)}$ and $P_{(i-1)(j-1)}^{(n)}$ are all one's. What is left to prove is the lower triangular part of the matrices. We know

$$P_{ij}^{(n+1)} = C_{i-1}^{j-1}, \tag{3.21}$$

and

$$P_{(i-1)(j-1)}^{(n)} + P_{(i-1)j}^{(n)} = C_{i-2}^{j-2} + C_{i-2}^{j-1} = C_{i-1}^{j-1}. \tag{3.22}$$

Therefore for $p = n + 1$, we have

$$P^{(n+1)} = A_1^{(n+1)} * A_2^{(n+1)} * \cdots * A_n^{(n+1)}. \tag{3.23}$$

This completes the proof. ■

This decomposition would require $O(p^2)$ operations for matrix-vector product, but these are all additions and there are no multiplications. This may be suitable for some architectures.

3.3 Relatives of a Pascal matrix

We have shown some fast multiplication algorithm for a Pascal matrix and a vector in the last section. We will show similar algorithms for some matrices related to a Pascal matrix in this section.

3.3.1 The transpose of a Pascal matrix

We first consider the transpose of a Pascal matrix. It can be decomposed the same way as a Pascal matrix. Indeed, applying the transpose to different decompositions (3.2), (3.7), and (3.12) of the Pascal matrix, we would obtain decompositions which allow fast matrix-vector products. Therefore we also have the following theorem similar to Theorem 3.3.

Theorem 3.6. *The multiplication of the transpose of a Pascal matrix and a vector can be done in $O(p \log p)$ operations.*

Proof. Follows from Theorem 3.3. ■

3.3.2 The product of the Pascal matrix and its transpose

The next matrix that is related to a Pascal matrix is

$$\mathbf{PP} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & C_{p-1}^{0} \\ 1 & 2 & 3 & 4 & \cdots & C_{p}^{1} \\ 1 & 3 & 6 & 10 & \cdots & C_{p+1}^{2} \\ 1 & 4 & 10 & 20 & \cdots & C_{p+2}^{3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{p-1}^{0} & C_{p}^{1} & C_{p+1}^{2} & C_{p+2}^{3} & \cdots & C_{2p-2}^{p-1} \end{bmatrix}. \tag{3.24}$$

Decomposition 1

The following lemma says that the Cholesky decomposition of the matrix PP is product of the Pascal matrix times the transpose of the Pascal matrix.

Lemma 3.7. *The matrix* $\mathbf{PP}$ *is the product of the corresponding Pascal matrix and its transpose,*

$$\mathbf{PP} = \mathbf{P} * \mathbf{P}' \tag{3.25}$$

Proof. For any pair of numbers (i, j), $i = 0, 1, \cdots, p-1$, $j = 0, 1, \cdots, p-1$, we need to prove that

$$PP_{ij} = \sum_{k=0}^{p-1} P_{ik} P'_{kj}, \tag{3.26}$$

where PP_{ij} is the (i, j)-th entry of the matrix PP, P_{ik} is the (i, k)-th entry of the matrix P, and P'_{kj} is the (k, j)-th entry of the matrix P'. This is equivalent to

$$C_{i+j}^{i} = \sum_{k=0}^{\min(i,j)} C_{i}^{k} C_{j}^{k}. \tag{3.27}$$

This is a well-known identity in the theory of combinatorics [Edelman].

To prove it, we need the following identity

$$C_i^k = C_i^{i-k}, \tag{3.28}$$

which is true by definition of the binomial coefficients. We know to select i objects from a total of $(i+j)$ objects, there are C_{i+j}^i ways. A equivalent selection can also be done in three steps. First divide the total $(i+j)$ objects into two groups with i objects and j objects. Then select $i-k$ objects from the group with i objects, there are C_i^{i-k} ways to do the selection; and select k objects from the group with j objects, there are C_j^k ways to do the selection. Finally we put them together to have i objects out of a total of $(i+j)$ objects. In each selection, k has to be less than or equal to i and j. The number of ways to select using this process is

$$\sum_{k=0}^{\min(i,j)} C_i^k C_j^k. \tag{3.29}$$

■

A number of proofs can be found in [Edelman]. This implies all decompositions that admit fast matrix-vector product for the Pascal matrix can be applied to matrix PP, since we can first apply them to P' and then to P.

Decomposition 2

We also have the following factorization.

Lemma 3.8. *The matrix* **PP** *can be decomposed as the following,*

$$\mathbf{PP} = diag(v) \cdot H \cdot diag(v), \tag{3.30}$$

where

$$v = \begin{bmatrix} 1 \\ \frac{1}{1!} \\ \frac{1}{2!} \\ \frac{1}{3!} \\ \vdots \\ \frac{1}{(p-1)!} \end{bmatrix}, \text{ and } H = \begin{bmatrix} 0! & 1! & 2! & \cdots & (p-1)! \\ 1! & 2! & 3! & \cdots & p! \\ 2! & 3! & 4! & \cdots & (p+1)! \\ 3! & 4! & 5! & \cdots & (p+2)! \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (p-1)! & p! & (p+1)! & \cdots & (2p-2)! \end{bmatrix} \tag{3.31}$$

is a Hankel matrix.

Proof. This lemma can be proved in a way similar to the proof of (3.2). Notice that the (n, m) entry PP_{nm} of the matrix PP is

$$P_{nm} = C^n_{n+m}. \tag{3.32}$$

where $C^n_{n+m} = \frac{(n+m)!}{n!m!}$. That is, every entry in n-th row of the matrix has a common factor $\frac{1}{n!}$, and every entry in m-th column of the Pascal matrix has a common factor $\frac{1}{m!}$. We can take out the common factor $\frac{1}{n!}$ of the n-th row and common factor $\frac{1}{m!}$ of the m-th column, and multiply from left side by a diagonal matrix which is the identity, except that the n-th entry in the diagonal is $\frac{1}{n!}$, and multiply from right side by a diagonal matrix which is the identity, except that the m-th entry in the diagonal is $\frac{1}{m!}$. This can be done for every row and column. Therefore we have factored the Pascal matrix into products of matrices with a Hankel matrix H in the middle and p diagonal matrices on the left, and p diagonal matrices on the right. Multiplying the diagonal matrices on the left and the right respectively, we end up with the diagonal matrices $diag(v)$ and $diag(v)$.

■

It is clear from the lemmas above that the multiplication of the matrix PP and any vector x can be either done by successively apply P' and P to x, or first

calculate the element-wise vector product of $v. * x$, then apply Hankel matrix H to the product, finally with the obtained vector, do another element-wise vector product with v to arrive the result of $PP \cdot x$. In either process, the involved matrices are either Toeplitz matrices or Hankel matrices. By Theorems 2.7, and 2.9, we have the following.

Theorem 3.9. *The multiplication of matrix PP and any vector can be done in $O(p \log p)$ operations.*

Although we can use both decompositions to build our fast translation operators, the first one is less efficient than the second one. The reason is that the cost of the product of a Toeplitz matrix and a vector is the same as that of a Hankel matrix and a vector, and the first one requires two multiplications of a Toeplitz matrix and a vector. We will use the second decomposition in our implementations.

In this chapter we have discussed how to multiply the Pascal matrix and its related matrices to a vector efficiently through matrix decomposition. These decompositions will be repeatedly used in building fast translation operators in 2D in Chapter 5 and fast rotation operators in 3D in Chapter 7. Notice that the entries of the Pascal matrix have very different magnitudes of numbers, and there can exist instability problems if the decomposition is implemented naively. We will discuss the instability problems related to the translation operators in 2D and 3D, and how to avoid them, in Chapter 9.

Chapter 4

The Fast Multipole Method in Two Dimensions

In this chapter we follow the work of Greengard of the fast multipole method (FMM) in his dissertation [Greengard88]. The purpose is to establish the ideas about the FMM, translation involved, and representation of those operations as matrices. In Chapter 5, we will provide decomposition of these matrices to speed up the translation step. We will also provide a complexity analysis of the FMM.

In this chapter we start with representation of a potential field in two dimensions as a multipole expansion in a complex plane. Then we provide known results on how to calculate the coefficients of a new multipole/local expansion that results from translating an existing multipole/local expansion, and coefficients of a local expansion resulting from the translation of an existing multipole expansion, and rewrite them as translation operators in the form of matrix-vector product. Next we present the fast multipole method (FMM), derive related error analysis and analyze the complexity of the algorithm. Finally, we review some of the previous work that has been done to improve the complexity of the translation operators and hence reduce the overall complexity of the algorithm. A lot of material here such as multipole expansion of a field, translation theorems, the FMM algorithm, and the complexity analysis can be found in Greengard's disserta-

tion [Greengard88]. A similar complexity analysis can be found in [Greengard97] for three dimensions. We present a slightly different error bound (4.30) in the multipole to local translation theorem.

4.1 Potential field in a complex plane

In two dimensions, the potential at $(x, y) \in R^2$ due to a point charge of intensity q at (x_0, y_0) is given by:

$$\phi(x, y) = -q \log\left(\sqrt{(x - x_0)^2 + (y - y_0)^2}\right). \tag{4.1}$$

If we view a point (x, y) in two dimensional space as a point in the complex plane, $z = x + iy$, then we may express the potential at z due to a single point charge q at z_0 as:

$$\phi(z) = -q \operatorname{Re}\left(\log(z - z_0)\right), \tag{4.2}$$

where $\operatorname{Re}(z)$ is the real part of a complex number z. Following standard practice, we will in this dissertation refer to

$$\phi(x, y) = q \log(z - z_0) \tag{4.3}$$

as the potential due to a charge.

4.2 Multipole expansions

Suppose a point charge of intensity q is located at z_0. The potential at any point z is $\phi(z) = q \log(z - z_0)$. Using the fact that

$$\log(z - z_0) = \log(z) + \log\left(1 - \frac{z_0}{z}\right) \tag{4.4}$$

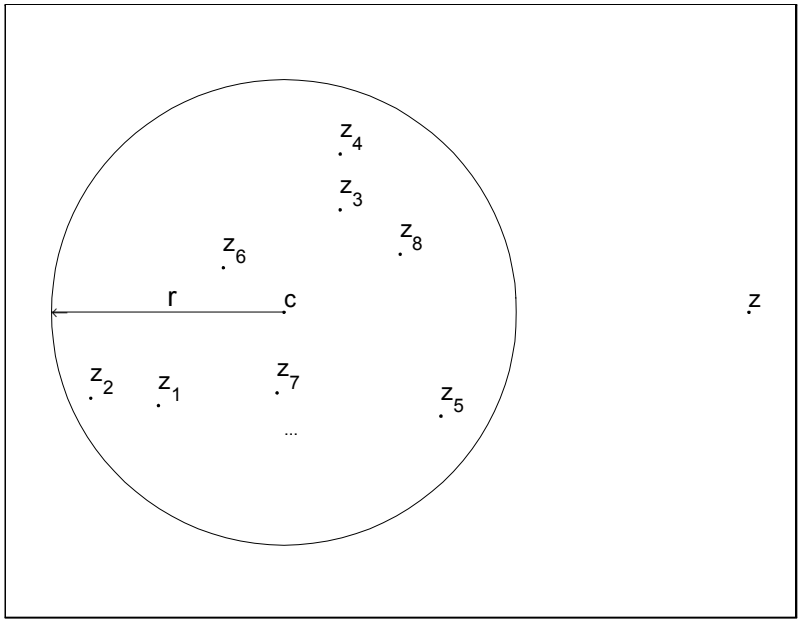

Figure 4.1: m charges in a disk centered at c with radius r

and the expansion

$$\log(1-w) = (-1)\sum_{k=1}^{\infty} \frac{w^k}{k}, \tag{4.5}$$

for any w such that $|w| < 1$, we have the following useful expansion centered at the origin, convergent for any z such that $|z| > |z_0|$,

$$\phi(z) = q\left(\log(z) - \sum_{k=1}^{\infty} \frac{1}{k}\left(\frac{z_0}{z}\right)^k\right). \tag{4.6}$$

More generally, we can have the following multipole expansion convergent at any point z outside a disk centered at point c, with radius $|z_0 - c|$,

$$\phi(z) = q\left(\log(z-c) - \sum_{k=1}^{\infty} \frac{1}{k}\left(\frac{z_0 - c}{z - c}\right)^k\right). \tag{4.7}$$

Start with this expansion, we can easily obtain the multipole expansion for a field containing m charges. The formal description is in the following theorem.

Theorem 4.1. *[Greengard88] (Multipole Expansion) Suppose that*

$$\phi(z) = \sum_{i=1}^{m} q_i \log(z - z_i) \tag{4.8}$$

is the potential due to a set of m charges of strengths $\{q_i,\ i = 1, \ldots, m\}$ *located at points* $\{z_i,\ i = 1, \ldots, m\}$*, with* $|z_i - c| < r$*. Then for any* $z \in C$ *with* $|z - c| > r$*, the potential* $\phi(z)$ *can be expressed as*

$$\phi(z) = Q \log(z - c) + \sum_{k=1}^{\infty} \frac{a_k}{(z - c)^k}, \tag{4.9}$$

where

$$Q = \sum_{i=1}^{m} q_i \quad \textit{and} \quad a_k = \sum_{i=1}^{m} \frac{-q_i (z_i - c)^k}{k}. \tag{4.10}$$

Furthermore, for any $p \geq 1$*, the error of truncating the infinite summation to p terms is given by*

$$\left| \phi(z) - Q \log(z - c) - \sum_{k=1}^{p} \frac{a_k}{(z - c)^k} \right| \leq \frac{A}{1 - |\frac{r}{z-c}|} \left| \frac{r}{z - c} \right|^{p+1}, \tag{4.11}$$

where

$$A = \sum_{i=1}^{m} |q_i|. \tag{4.12}$$

The proof of this theorem can be found in [Greengard88]. With this theorem, we can use a single multipole expansion to express the potential due to multiple charges inside a disk. See figure 4.1.

4.3 Translation operators for the two dimensional Laplace equation and their matrix forms

In this section we will provide three multipole translation theorems that describe the translation operators required by the FMM. While they can be found in the

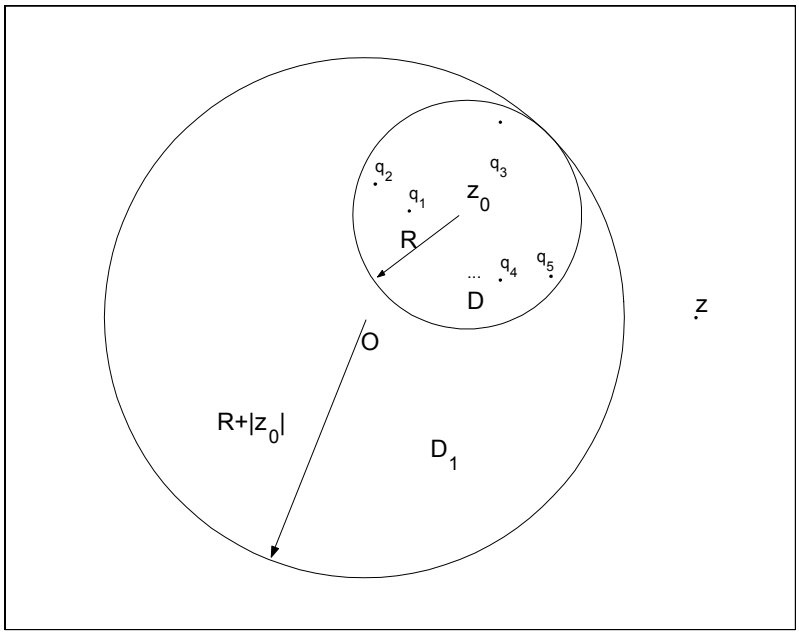

Figure 4.2: Multipole to Multipole Translation

literature [Greengard88], a modified error bound in Theorem 4.4 is presented, and in addition we rewrite the results in matrix form.

Theorem 4.2. *[Greengard88] (Translation of a Multipole Expansion) Suppose that*

$$\phi(z) = a_0 \log(z - z_0) + \sum_{k=1}^{\infty} \frac{a_k}{(z - z_0)^k} \tag{4.13}$$

is a multipole expansion of the potential due to a set of m charges of strengths $\{q_i,\ i = 1, \ldots, m\}$, located inside the circle D of radius R with center at z_0. Then for z outside of the circle D_1 of radius $(R + |z_0|)$ and center at the origin,

$$\phi(z) = a_0 \log(z) + \sum_{n=1}^{\infty} \frac{b_n}{z^n}, \tag{4.14}$$

where

$$b_n = -\frac{a_0 z_0^n}{n} + \sum_{k=1}^{n} a_k z_0^{n-k} C_{n-1}^{m-k}, \tag{4.15}$$

with C_n^k the binomial coefficients. Furthermore, for any $p \geq 1$, the error of truncating the infinite summation to p terms is given by

$$\left| \phi(z) - a_0 \log(z) - \sum_{n=1}^{p} \frac{b_n}{z^n} \right| \leq \left(\frac{A}{1 - \left| \frac{|z_0|+R}{z} \right|} \right) \left| \frac{|z_0| + R}{z} \right|^{p+1}, \tag{4.16}$$

with A defined by (4.12).

Proof. The coefficients $\{b_n,\ n = 1, ...p\}$ are easily obtained by reexpanding the Laurent series (4.13) around the origin using

$$\frac{a_k}{(z - z_0)^k} = \frac{a_k}{z^k} \sum_{n=0}^{\infty} C_{n+k-1}^n \left(\frac{z_0}{z} \right)^n. \tag{4.17}$$

For the error bound, observe that for a given set of charges, due to the uniqueness of the multipole expansion, the coefficients $\{b_n,\ n = 1, ...p\}$ directly computed using the multipole expansion theorem are the same as the ones obtained if we first compute the coefficients $\{a_k,\ k = 1, ...p\}$ using the multipole expansion theorem, and then compute the coefficients $\{b_n,\ n = 1, ...p\}$ from $\{a_k,\ k = 1, ...p\}$ as in this theorem. That is, two different ways result in the same error. Therefore the error bound follows from the Multipole Expansion Theorem 4.1. ■

Remarks:

1. Note that the error bound estimates the true error of the shifted multipole expansion from the potential induced by particles. It includes not only the difference between the shifted expansion and the original truncated expansion, but also the error caused by truncating the multipole expansion (4.13). Even it does not cause any error in the FMM, a comment made by Greengard after the theorem in [Greengard88] could be misleading. He states that "we may shift the center of a truncated multipole expansion without loss of precision". This could cause an improper understanding

of this error bound. If we look at error bound (4.16) and error bound (4.11) and consider the fact that $\{b_n, n = 1, ..., p\}$ only depends on $\{a_n, n = 0, 1, ..., p\}$, it is easy to get the wrong impression that final error of this shifted expansion stays the same. It is clear from the figure (4.2) that the term $|\frac{|z_0|+R}{z}|$ in (4.16) is greater than the term $|\frac{r}{z-c}|$, which is $\frac{R}{|z-z_0|}$ here, in (4.11).

2. Since our main interest is on speedup of the translations, for simplicity and without any change in the computational complexity, we are going to drop the first term due to $a_0 \log(z - z_0)$ in our later formulas. We can rewrite coefficients of the new expansion in terms of those of the old expansion in matrix form as follows,

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ \vdots \\ b_p \end{bmatrix} = SS(z_0) \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_p \end{bmatrix} \tag{4.18}$$

where

$$SS(z_0) = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ C_1^1 z_0 & 1 & 0 & 0 & \cdots & 0 \\ C_2^2 z_0^2 & C_2^1 z_0 & 1 & 0 & \cdots & 0 \\ C_3^3 z_0^3 & C_3^2 z_0^2 & C_3^1 z_0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{p-1}^{p-1} z_0^{p-1} & C_{p-1}^{p-2} z_0^{p-2} & C_{p-1}^{p-3} z_0^{p-3} & C_{p-1}^{p-4} z_0^{p-4} & \cdots & 1 \end{bmatrix}. \tag{4.19}$$

3. The multipole translation operator is the multipole-to-multipole translation operator. It is easy to see that the translation of a multipole expansion requires $O(p^2)$ work.

Theorem 4.3. *[Greengard88] (Translation of a Local Expansion) Suppose that*

$$\phi(z) = \sum_{k=0}^{p} a_k (z - z_0)^k \tag{4.20}$$

is a local expansion centered at z_0. Then

$$\phi(z) = \sum_{n=0}^{p} b_n z^n \tag{4.21}$$

is a local expansion centered at the origin, where

$$b_n = \sum_{k=n}^{p} a_k (-z_0)^{k-n} C_k^n. \tag{4.22}$$

Proof. If we expand the polynomial and combine like terms, we can easily get the results. ■

remarks:

1. Similarly we can rewrite the coefficients of the new expansion in terms of those of the old expansion in matrix form as follows,

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix} = RR(z_0) \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_p \end{bmatrix}, \tag{4.23}$$

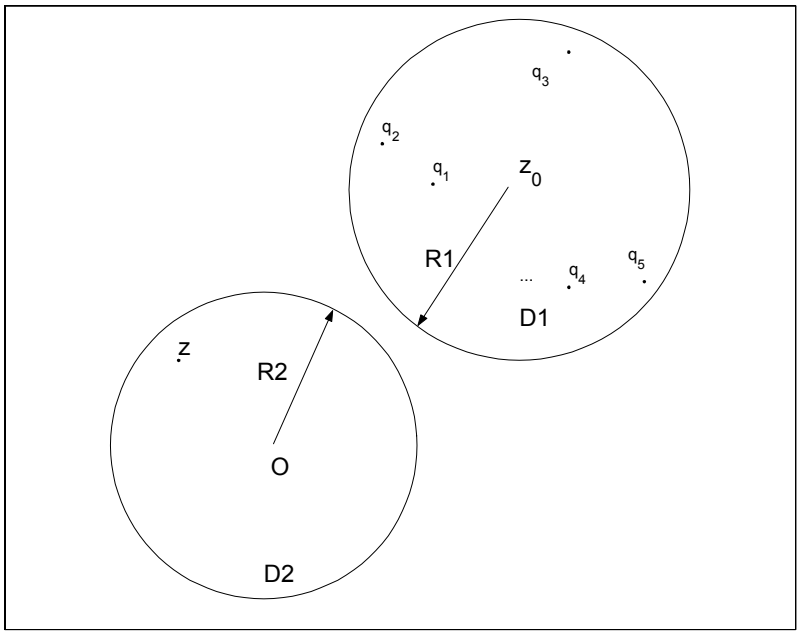

Figure 4.3: Multipole To Local Translation

where

$$RR(z_0) = \begin{bmatrix} 1 & -z_0 & z_0^2 & -z_0^3 & \cdots & (-z_0)^p \\ 0 & 1 & -2z_0 & 3z_0^2 & \cdots & p(-z_0)^{p-1} \\ 0 & 0 & 1 & -3z_0 & \cdots & C_{p-1}^2(-z_0)^{p-2} \\ 0 & 0 & 0 & 1 & \cdots & C_{p-1}^3(-z_0)^{p-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}, \tag{4.24}$$

is the local-to-local translator, or local translation matrix.

2. A translation of a local expansion requires $O(p^2)$ work.

Theorem 4.4. *[Greengard88] (Conversion of a Multipole to Local Expansion) Suppose that*

$$\phi(z) = a_0 \log(z - z_0) + \sum_{k=1}^{\infty} \frac{a_k}{(z - z_0)^k} \tag{4.25}$$

is a multipole expansion of the potential due to a set of m charges of strengths $\{q_i,\ i = 1, \ldots, m\}$ located inside the circle D_1 of radius R_1 with center at z_0 and that $|z_0| > R_1 + R_2$. Then this multipole expansion converges inside the circle D_2 of radius R_2 centered about origin. For z inside of the circle D_2 which lies outside of D_1, the potential can be expressed by the following local expansion:

$$\phi(z) = \sum_{n=0}^{\infty} b_n z^n, \tag{4.26}$$

where

$$b_n = \frac{1}{z_0^n} \sum_{k=1}^{\infty} \frac{a_k}{(-z_0)^k} C_{n+k-1}^n - \frac{a_0}{n z_0^n} * \delta(n) + (1 - \delta(n)) * a_0 \log(-z_0), \tag{4.27}$$

with

$$\delta(n) = \begin{cases} 0 & \text{for } n = 0 \\ 1 & \text{for } n \neq 0 \end{cases}. \tag{4.28}$$

Furthermore, for any $p \geq 1$, the error of truncating the infinite summation to p terms is given by

$$|\phi(z) - \sum_{n=0}^{p} b_n z^n| \ \leq \left(\frac{A}{1 - \left| \frac{z}{|z_0| - R_1} \right|} \right) \left| \frac{z}{|z_0| - R_1} \right|^{p+1}, \tag{4.29}$$

with A defined by (4.12). In FMM, $b_n,\ n = 0, 1, ..., p$, are calculated by setting $a_k = 0$, for $k = m + 1, ..., \infty$. Let us in this case denote b_n by b'_n. Then we have the following error bound,

$$|\phi(z) - \sum_{n=0}^{p} b'_n z^n| \ \leq \frac{A}{1 - |\frac{z}{|z_0| - R_1}|} \left| \frac{z}{|z_0| - R_1} \right|^{p+1} + \frac{A}{1 - \left| \frac{R_1}{z - z_0} \right|} \left| \frac{R_1}{z - z_0} \right|^{m+1}. \tag{4.30}$$

Proof. The first part can be proved in a similar way to the proof for the multipole to multipole translation. We only prove the last error bound.

From the error bound of the multipole expansion (4.16), it is easy to see that

$$\left| \phi(z) - a_0 \log(z - z_0) - \sum_{k=1}^{m} \frac{a_k}{(z - z_0)^k} \right| < \sum_{k=m+1}^{\infty} \frac{|a_k|}{|z - z_0|^k} < \frac{A}{1 - \left| \frac{R_1}{z - z_0} \right|} \left| \frac{R_1}{z - z_0} \right|^{m+1}. \tag{4.31}$$

This truncation is equivalent to letting $a_k = 0,\ k = m+1, m+2, \cdots$ in (4.27). We then have,

$$
\begin{aligned}
|\phi(z) - \sum_{n=0}^{p} b'_n z^n| &= \Bigg|\phi(z) - \sum_{n=0}^{p}\Big(\frac{1}{z_0^n}\sum_{k=1}^{m}\frac{a_k}{(-z_0)^k}C_{n+k-1}^n - \\
&\quad \frac{a_0}{nz_0^n} * \delta(n) + (1-\delta(n)) * a_0\log(-z_0)\Big)z^n\Bigg| \qquad (4.32)\\
&\le \Bigg|\phi(z) - \sum_{n=0}^{p}\Big(\frac{1}{z_0^n}\sum_{k=1}^{\infty}\frac{a_k}{(-z_0)^k}C_{n+k-1}^n - \\
&\quad \frac{a_0}{nz_0^n} * \delta(n) + (1-\delta(n)) * a_0\log(-z_0)\Big)z^n\Bigg| + \\
&\quad \Bigg|\sum_{n=0}^{p}\left(\frac{1}{z_0^n}\sum_{k=1}^{\infty}\frac{a_k}{(-z_0)^k}C_{n+k-1}^n\right)z^n - \\
&\quad \sum_{n=0}^{p}\left(\frac{1}{z_0^n}\sum_{k=1}^{m}\frac{a_k}{(-z_0)^k}C_{n+k-1}^n\right)z^n\Bigg| \qquad (4.33)\\
&\le \left(\frac{A}{1-|\frac{z}{|z_0|-R_1}|}\right)\left|\frac{z}{|z_0|-R_1}\right|^{p+1} + \\
&\quad \sum_{k=m+1}^{\infty}\left|\frac{a_k}{(-z_0)^k}\right|\sum_{n=0}^{\infty}\left|\frac{z}{z_0}\right|^n C_{n+k-1}^n \qquad (4.34)\\
&\le \left(\frac{A}{1-|\frac{z}{|z_0|-R_1}|}\right)\left|\frac{z}{|z_0|-R_1}\right|^{p+1} + \\
&\quad \sum_{k=m+1}^{\infty}\left|\frac{a_k}{(-z_0)^k}\right|\frac{1}{\left(1-\left|\frac{z}{z_0}\right|\right)^k} \qquad (4.35)\\
&\le \left(\frac{A}{1-|\frac{z}{|z_0|-R_1}|}\right)\left|\frac{z}{|z_0|-R_1}\right|^{p+1} + \qquad (4.36)\\
&\quad \frac{A}{1-\left|\frac{R_1}{z-z_0}\right|}\left|\frac{R_1}{z-z_0}\right|^{m+1}. \qquad (4.37)
\end{aligned}
$$

■

remarks:

1. Note that this bound estimates the error of the truncated local expansion from the exact potential induced by particles. It includes three sources of

error, the error caused by truncating the multipole expansion, the error caused by translation of the multipole expansion, and the error due to conversion of the multipole expansion to a local expansion. That is to say, this is the only error bound needed in the final error analysis of the FMM.

2. For simplicity, we drop the first term $a_0 \log(z - z_0)$ in our later formulas. We can rewrite the coefficients of the new expansion in terms of those of the old expansion in matrix form as follows,

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{p-1} \end{bmatrix} = SR(z_0) \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_p \end{bmatrix}, \tag{4.38}$$

where the matrix $SR(z_0)$ depends only on the translation vector z_0, and it is given by

$$SR(z_0) = \begin{bmatrix} -{z_0}^{-1} & {z_0}^{-2} & -{z_0}^{-3} & \cdots & (-1)^p {z_0}^{-p} \\ -{z_0}^{-2} & 2\,{z_0}^{-3} & -3\,{z_0}^{-4} & \cdots & (-1)^p C_p^1\, {z_0}^{-p-1} \\ -{z_0}^{-3} & 3\,{z_0}^{-4} & -6\,{z_0}^{-5} & \cdots & (-1)^p C_{p+1}^2\, {z_0}^{-p-2} \\ -{z_0}^{-4} & 4\,{z_0}^{-5} & -10\,{z_0}^{-6} & \cdots & (-1)^p C_{p+2}^3\, {z_0}^{-p-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -{z_0}^{-p} & C_p^1 {z_0}^{-p-1} & -C_{p+1}^2 {z_0}^{-p-2} & \cdots & (-1)^p C_{2p-2}^{p-1}\, {z_0}^{-2p+1} \end{bmatrix}. \tag{4.39}$$

where $SR(z_0)$ is the multipole-to-local translation matrix.

3. A conversion of a multipole expansion to a local expansion is $O(p^2)$ work.

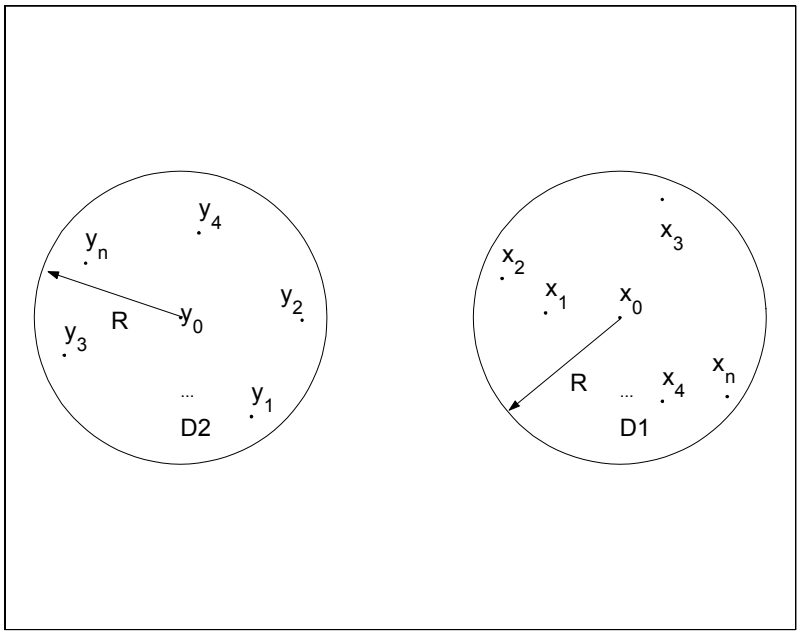

Figure 4.4: Well-separated case

The Fast Multipole Method (FMM) relies on these three operators to perform all of the necessary manipulations. It is clear from the formulas that each translation requires $O(p^2)$ operations if calculated directly.

4.4 What is the fast multipole method?

Before introducing the FMM, we first demonstrate the main idea with a special case [Greengard88], and then the nonadaptive scheme of the well known tree codes [Barnes86]. We show that the multipole expansion can be used to speed up evaluation of potential interactions. Then the FMM follows naturally.

4.4.1 A special case

Suppose that sources of strengths $\{q_i, i = 1, ..., m\}$ are located at the points $\{x_i, i = 1, ..., m\} \in C$ and we need to evaluate the potential at the points $\{y_i, i = 1, ..., n\} \in C$. We also assume that there exist points $x_0, y_0 \in C$ and a positive real number r such that $|x_i - x_0| < r,\ i = 1, ..., m,\ |y_j - y_0| < r,\ j = 1, ..., n,$ and $|x_0 - y_0| > 3r$, in which case we say that $\{x_i\}$ and $\{y_i\}$ are well-separated. The potential at evaluation points $\{y_j\}$ due to the sources at points $\{x_i\}$ can be computed directly by

$$\phi(y_j) = \sum_{i=1}^{m} q_i \log(y_j - x_i), \qquad j = 1, ...n. \tag{4.40}$$

It is easy to see that this requires $O(nm)$ operations. On the other hand we can use multipole expansion to reduce the number of operations to $O(m) + O(n)$.

First we use Theorem 4.1 (Multipole Expansion) to calculate the coefficients of a p-term multipole expansion of the potential due to the charges at the points $\{x_i, i = 1, ..., m\}$ about x_0, where p is determined by a desired precision ϵ according to the following inequality

$$|\phi(z) - Q\log(z - x_0) - \sum_{k=1}^{p} \frac{a_k}{(z - x_0)^k}| \ \leq (\frac{A}{1 - |\frac{1}{2}|})(\frac{1}{2})^{p+1} < \epsilon, \tag{4.41}$$

that is, p is of order $-\log_2(\epsilon)$. This requires $O(mp)$ operations. Evaluating the above multipole expansion at all points $\{y_j,\ j = 1, ..., n\}$ requires $O(np)$ operations. If $p << n$, the total number of operations is $O(m + n)$. For $m = n$, this method of calculation requires $O(n)$ operations, while the direct method requires $O(n^2)$ operations.

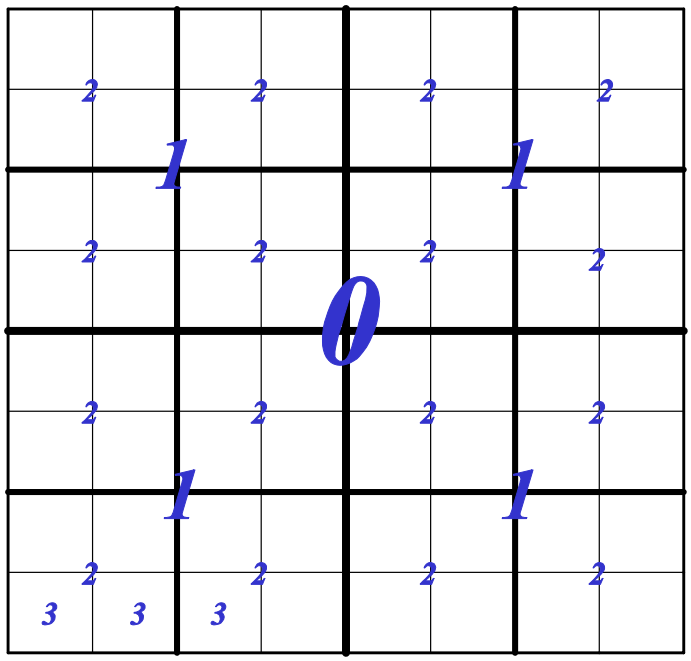

Figure 4.5: Four levels of box hierarchy

4.4.2 Nonadaptive tree codes

Notice that in the previous case, we require the source points and evaluation points to be well-separated, so that the whole set of source points are viewed as one cluster and the induced potential is expressed as a single multipole expansion. Usually this is not the case. However, we can subdivide the whole computation domain containing all the sources of the system into hierarchical boxes. Then we would have a lot of clusters of well-separated points, and could use the above method to efficiently calculate the interactions between clusters and particles which are far away and handle the interactions with particles which are nearby directly. This is the central strategy of the "tree codes".

To illustrate the main idea, we assume that the source points are fairly ho-

mogeneously distributed in a square so that adaptive refinement is not required. A hierarchy of boxes which refine the computational domain is constructed as follows. We start with the entire computational domain as level 0, or root. Level 0 is subdivided into four equal square child boxes which form level 1, and each of these has the level 0 box as its parent. Then each of the boxes at level 1 is again subdivided into four child boxes, the resulting sixteen boxes are considered as level 2. Recursively, we obtain level 3, 4,..., until the number of levels of refinement is roughly $\log_4 N$, where N is number of particles in the system. We call the four boxes at level $l+1$ obtained by subdivision of a box at level l its children. This imposes a natural tree structure on this box hierarchy. Two boxes are said to be neighbors if they are at the same level and share a boundary point. Two boxes are said to be well-separated if they are at the same level and are not neighbors. Interaction list of a box i at level l is the children of the neighbors of i's parent which are well-separated from box i (Figure 4.6).

It is clear that there are no pairs of well-separated boxes at levels 0 and 1. There are a number of well-separated pairs of boxes at level 2. For each one of the sixteen boxes at level 2, the multipole expansion around the center of the box is created to approximate the potential induced by the sources contained in the box. We can use the computed multipole expansion to calculate the interaction between the particles in all well-separated boxes (that is, any pair of boxes that are not neighbors in this case). Then, for each particle contained in a level 2 box, it remains to compute the interactions between particles contained in its box 's neighbors, and this is done recursively. First each level 2 box is refined to create level 3. For a given level 3 box, it is easy to see that other level 3 boxes that can be interacted with by means of multipole expansion are those defined as

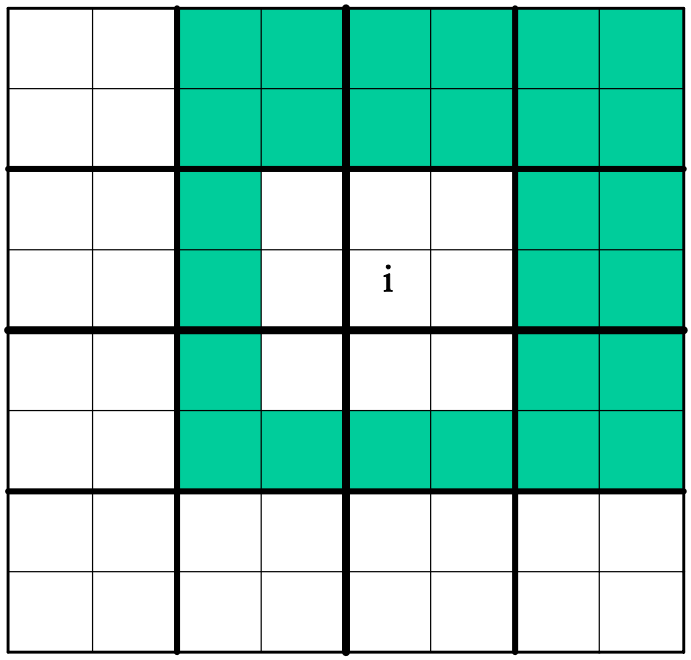

Figure 4.6: Interaction list for box i

members of its interaction list since those boxes that are outside the neighbors of the parent are already computed. Again what is left to be computed is between particles contained in neighboring boxes at level 3. This process is repeated from level 3 to level 4, from level 4 to level 5, until the finest level is reached. From the nature of this recursive process, for each particle we still have the particles contained in its box's neighbors to be interacted with, this is finally computed by direct calculation.

The amount of work at each level to create all expansions is approximately $O(Np)$ operations since a p-term multipole expansion is obtained for each particle at each level, where $p \approx \log_c \frac{1}{\epsilon}$ and $c = \frac{3}{\sqrt{2}}$. And the total amount of work at each level to evaluate is about $27Np$ operations since for each particle, there are at most 27 boxes whose multipole expansion are computed. And the final direct computation requires about $9N$ operations. So the total work is approximately $28Np\log_4(N) + 9N$.

4.4.3 The fast multipole method

Note that in the whole process of tree codes, only the multipole expansion theorem is involved. At every level, all multipole expansions for all boxes are constructed directly from particles. There is no connection between the different levels. In the FMM, three translation operators are used to calculate interactions between clusters.

The FMM mainly consists of two passes – an upward pass and a downward pass. In the upward pass, the multipole expansions for all boxes at the finest level are first formed. Theorem 4.2 is then used to construct the multipole expansions for all boxes at the next higher level – for each parent box, shift the centers of

all four child boxes to the center of the parent and add the coefficients together to obtain the expansion for the parent box; this is done recursively until level 2 is reached. In the downward pass, the local expansion for level 1 is initialized to zero first; then Theorems 4.3 and 4.4 are used to construct local expansions for all boxes at level 2 from the local expansions at level 1; This is done recursively from level 2 to level 3, level 3 to level 4, and so on, until the finest level is reached.

Before we start the formal description of the algorithm, we introduce some notation. $\Phi_{l,i}$ is the p-term multipole expansion about the center of the box i at level l, describing the potential field outside box i's neighbors due to all particles contained inside the box i. $\Psi_{l,i}$ is the p-term local expansion about the center of the box i at level l, describing the potential field induced by all particles outside box i's neighbors. $\tilde{\Psi}_{l,i}$ is the p-term local expansion about the center of the box i at level l, describing the potential field induced by all particles outside the neighbors of box i's parent.

Algorithm

Initialization. Choose precision to be desired ϵ and the number of levels $n \approx \log_4 N$. Set the length of multipole and local expansions to $p \approx \log(\frac{1}{\epsilon})$. Then the number of boxes at the finest level is 4^n, and the number of particles per box is $s \approx \frac{N}{4^n}$, since we still assume the sources are fairly homogeneously distributed in a square. Set up the hierarchical data structure and sort all points into the boxes at the finest level.

Upward Pass

step 1

For each box i at the finest level n, use Theorem 4.1 to form p-term multipole expansion $\Phi_{n,i}$, representing the potential field induced by all particles in the box.

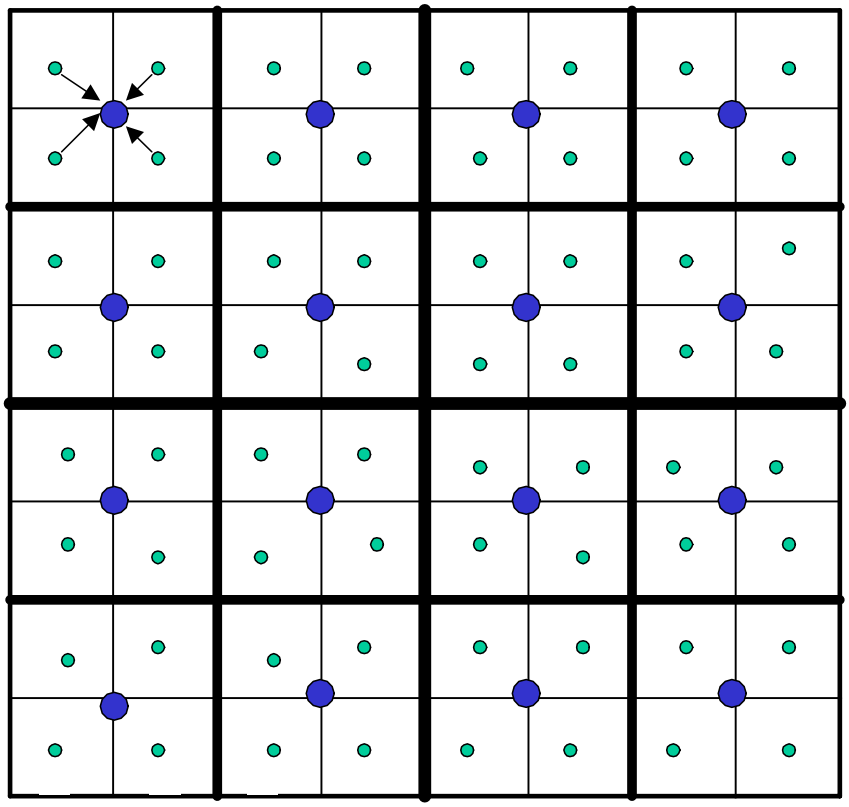

Figure 4.7: Step 2. Multipole to multipole translation

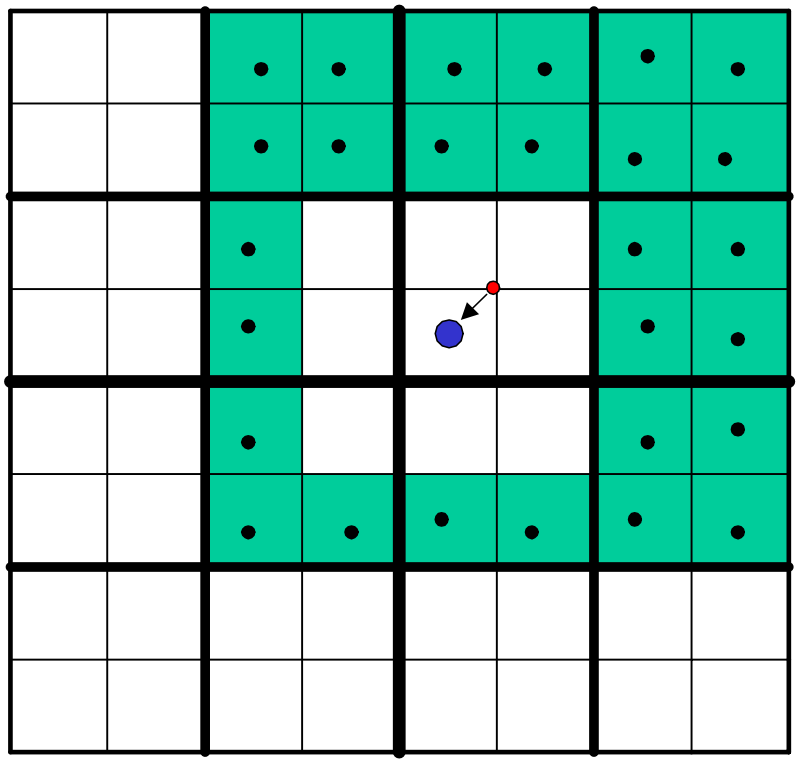

Figure 4.8: Step 3. Local to local translation

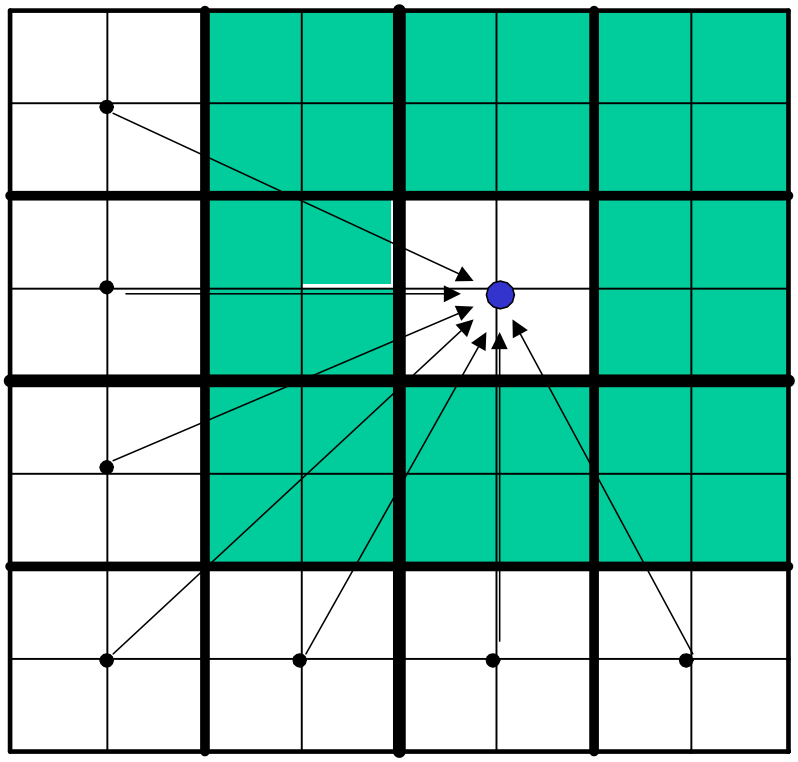

Figure 4.9: Step 3. Multipole to local translation at level 2

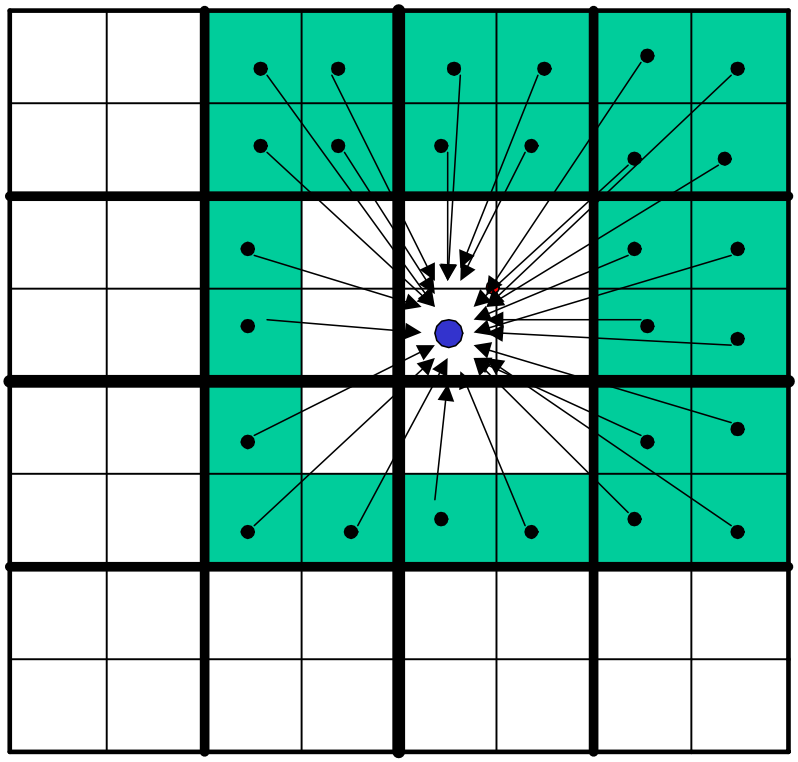

Figure 4.10: Step 3. Multipole to local translation

Record the coefficients of each expansion for all 4^n boxes.

step 2

For levels $l = n-1, n-2, ..., 2,$

For each box j at level l, use Theorem 4.2 to shift the centers of multipole expansions of its four child boxes to the center of box j and merge them to form $\Phi_{l,j}$ which represents the potential field induced by all particles in box j, or all particles in its four child boxes. see figure 4.7.

Downward Pass

Set $\Psi_{1,1} = \Psi_{1,2} = \Psi_{1,3} = \Psi_{1,4} = 0.$

step 3

For levels $l = 2, 3, ...n,$

For each box j at level l, use Theorem 4.3 to shift the center of local expansion Ψ_{l-1} of j's parent box to the center of box j to form $\tilde{\Psi}_{l,j}$. See figure 4.8.

Use Theorem 4.4 to convert the multipole expansions of all boxes that is in the interaction list of box j to a local expansion about the center of box j, and add all these to $\tilde{\Psi}_{l,j}$ to form $\Psi_{l,j}$, which represent the potential field induced by all particles outside box j's neighbors. See figure 4.9 and figure 4.10.

step 4

For boxes $j = 1, 2, ...4^n$ at the finest level,

For each particle at box j, evaluate $\Psi_{n,j}$ at the particle position.

step 5

For boxes $j = 1, 2, ...4^n$ at the finest level,

For each particle at box j, evaluate the interactions with particles in j's neighbor boxes directly.

4.5 Error analysis

It is obvious that the local translation operator in Theorem 4.3 is exact and that the truncated operators described in Theorem 4.1 and Theorem 4.4 introduce errors into the expansions. Even though it is equally obvious that the truncated multipole expansion operator described in Theorem 4.2 introduces error in the same way, there sometimes is a misconception that it causes no error [Greengard88]. The key point to make no mistake in the error analysis is to realize that although truncated operators in Theorems 4.1 and 4.2 cause errors, all the errors can be estimated using the single error bound (4.30) given in Theorem 4.4.

For a particular evaluation location y, the error of computed potential at this point caused by the entire system is

$$E(y) = \left| \sum_{i=1}^{N} \phi(y, x_i) - \sum_{i=1}^{N} \hat{\phi}(y, x_i) \right|, \tag{4.42}$$

where $\phi(y, x_i)$ is the potential at point y induced by a particle at point x_i and $\hat{\phi}(y, x_i)$ is the computed potential. From the process of the FMM, we know that the particles in the neighbor boxes of the box containing y at the finest level are handled directly, therefore there is no error in these calculations. All the rest of particles are contained in boxes that belong to different level of interaction lists. Let us denote by IL_l the interaction list for the box that contains point y at level l, P_b the set of all particles contained in box b. We have,

$$\begin{aligned} E(y) &= \left| \sum_{i=1}^{N} \Big(\phi(y, x_i) - \hat{\phi}(y, x_i) \Big) \right| \\ &\leq \left| \sum_{l=2}^{n} \sum_{b \in IL_l} \sum_{i \in P_b} \Big(\phi(y, x_i) - \hat{\phi}(y, x_i) \Big) \right| \end{aligned} \tag{4.43}$$

Recall that in the FMM, there are errors in steps 1, 2, and 3, but the total error in the final local expansions $\Psi_{n,j}$ is controlled by (4.16).

$$
\begin{aligned}
E(y) \quad \leq \quad & \sum_{l=2}^{n} \sum_{b \in IL_l} \left(\frac{\sum_{i \in P_b} |q_i|}{1 - |\frac{z}{|z_0| - R_1}|} \left| \frac{z}{|z_0| - R_1} \right|^{p+1} \right. && (4.44) \\
& \left. + \frac{\sum_{i \in P_b} |q_i|}{1 - \left| \frac{R_1}{z - z_0} \right|} \left| \frac{R_1}{z - z_0} \right|^{p+1} \right) && (4.45) \\
\leq \quad & \sum_{l=2}^{n} \sum_{b \in IL_l} \sum_{i \in P_b} |q_i| * C * \left(\frac{2^{\frac{1}{2}}}{4 - 2^{\frac{1}{2}}} \right)^{p},
\end{aligned}
$$

where $C = 2^{\frac{1}{2}} + 1$ is a constant. So the total error for the entire system is

$$
E(y) \leq C \sum_{i=1}^{N} |q_i| \left(\frac{2^{\frac{1}{2}}}{4 - 2^{\frac{1}{2}}} \right)^{p}. \tag{4.46}
$$

Of course, this error bound is very conservative, even though it is rigorous.

4.6 Discussion of complexity of the FMM

We present complexity analysis in this section. It is mostly following the work of Greengard and Rokhlin [Greengard88, Greengard97]. Suppose there are total N particles in the system. The FMM consists of two parts. The number of operations required for part one, the initialization step – construction of the data structure of the FMM algorithm, is usually $O(N \log N)$. The number of operations required for part two, the computational steps – the upward pass and downward pass, is $O(N)$ for a fixed $p \approx \log_c(\frac{1}{\epsilon})$, where $c = \frac{\sqrt{2}}{4-\sqrt{2}}$ (from (4.46) if we want a fixed precision, then $p \approx \log_c(\frac{N}{\epsilon})$. The resulting algorithm can be of complexity of $O(N \log N)$ or $O(N \log^2 N)$). In practice, the cost of the second part dominates the total cost, the CPU time of the first part compared with that

of the second part is negligible. Furthermore, the FMM is sometimes used as part of an iterative method to calculate a matrix-vector product. In this case, the initialization is done only once in the first step of the iterative method; the upward pass and downward pass steps are repeated for each iteration. An efficient second part is central to reduce the overall complexity. The complexity of each step of the second part of FMM is discussed next.

In step 1, for each particle, the potential induced by it is expressed as a p-term expansion about the center of the box that contains it at the finest level. The number of operations required at this step is $O(Np)$.

In step 2, each translation requires $O(p^2)$ operations. For level l, there are 4^l boxes, and for each box, it requires four translations. So the total number of operations is the order of

$$\sum_{l=2}^{n-1} 4 * 4^l * p^2 = \sum_{l=3}^{n} 4^l * p^2 = \frac{4^3 - 4^{n+1}}{1-4} p^2 \approx \frac{4}{3} 4^n p^2 = \frac{4}{3} \frac{N}{s} p^2. \tag{4.47}$$

In step 3, the local translation operator and the multipole to local translation operator are used. They both require $O(p^2)$ operations. At level l, there are 4^l boxes, and for each box, it requires one local translation and at most 27 multipole to local translations. The operation count in this step is approximately

$$\sum_{l=2}^{n} 4^l * p^2 + \sum_{l=2}^{n} 4^l * p^2 * 27 = \frac{4^2 - 4^{n+1}}{1-4} * 28 * p^2 \approx \frac{4}{3} * 4^n * 28 p^2 = \frac{112}{3} \frac{N}{s} p^2. \tag{4.48}$$

In step 4, each particle requires one evaluation of a p-term local expansion. This step requires $O(Np)$ operations.

In step 5, each particle in a box at the finest level interacts with particles contained at its 9 neighboring boxes. There are about s particles per box. Therefore this step requires $O(9Ns)$ operations.

The total cost for all five steps is approximately

$$2Np + \frac{116}{3}\frac{N}{s}p^2 + 9Ns. \tag{4.49}$$

It is clear that the total complexity can be optimized by appropriately selecting the parameter s. With $s \approx \frac{\sqrt{348}}{9}p$, the total number of operations is approximately $40Np$.

It is clear that the major obstacle to achieving reasonable efficiency at high precision is the cost of translation operators. There have been several improvements proposed [Hrycak98] since the original FMM appeared. We will discuss them in the next section.

4.7 Previous work on two dimensional translation operators

From the process of the FMM, it is easy to see that all translation operator are very expensive with complexity of the order of $O(p^2)$. And the multipole to local translation operator is used 27 times almost for all boxes in all hierarchical boxes. These are the main steps that slow down the speed of the FMM. To remedy this situation, Hrycak and Rokhlin [Hrycak98] constructed a new analytical apparatus, which diagonalizes most translation operators. The key idea is to represent the potential as a complex exponential through integral and quadrature, thus to reduce the linear but dense multipole to local translation operator to a diagonal one. We briefly record the core idea employed in that paper below.

They considered the potential $\frac{1}{z-z_0}$. First the potential can be represented as

an integral. If z and z_0 are complex numbers such that $\mathrm{Re}(z - z_0) > 0$, then

$$\frac{1}{z - z_0} = \int_0^\infty e^{-x(z-z_0)} \mathrm{d}x \tag{4.50}$$

Then this integral is approximated by the following formula using finite quadrature,

$$\int_0^\infty e^{-x(z-z_0)} \mathrm{d}x \sim \sum_{k=1}^{q} \omega_k e^{-x_k(z-z_0)} \tag{4.51}$$

with ω_k, x_k chosen to minimize the error of the approximation. Thus, the potential of a unit charge at the point z_0 can be approximated by a linear combination of exponentials as follows

$$\frac{1}{z - z_0} \sim \sum_{k=1}^{q} \omega_k e^{-x_k(z-z_0)} = \sum_{k=1}^{q} \omega_k e^{-x_k(w-z_0)} e^{-x_k(z-w)} \tag{4.52}$$

for any w in a certain domain. Suppose

$$\phi(z) = \sum_{i=1}^{m} \frac{q_i}{z - z_i} \tag{4.53}$$

is a potential due to a set of m charges of intensity $\{q_i,\ i = 1, ...m\}$ located at the locations $\{z_i,\ i = 1, ...m\}$ in a certain domain. By (4.52), it is easy to see

$$\phi(z) = \sum_{i=1}^{m} \frac{q_i}{z - z_i} \ \tilde{} \ \sum_{k=1}^{q} c_k e^{-x_k(z-w)}, \tag{4.54}$$

where the coefficients c_k is defined as

$$c_k = \omega_k \sum_{i=1}^{m} q_i e^{-x_k(w-z_i)} \quad \text{for all} \ \ k = 1, 2, ...q. \tag{4.55}$$

Therefore, the potential ϕ can be expanded as a linear combination of exponentials $e^{-x_k(z-w)}$, centered at w. Now that the translation operator is diagonal is obvious– for another point $\tilde{w}$, the potential ϕ can be expanded as

$$\phi(z) \ \tilde{} \ \sum_{k=1}^{q} \tilde{c}_k e^{-x_k(z-\tilde{w})}, \tag{4.56}$$

where

$$\tilde{c}_k = c_k e^{-x_k(\tilde{w}-w)} \quad \text{for all} \quad k = 1, 2, ...q. \tag{4.57}$$

For the classical Laguerre quadrature, the integral requires 56 nodes for 15 digit approximation, 28 nodes for 7 digit approximation and 14 nodes for 3 digit approximation. Hrycak and Rokhlin [Hrycak98] used better designed quadratures in [RokhlinS98]. It requires 8 nodes for 3 digit approximation, 16 nodes for 7 digit approximation and 33 nodes for 15 digit approximation.

With these preparations, multipole expansions can be easily converted into exponential expansions and exponential expansions into local power series expansions.

Another technique in the same paper [Hrycak98] to reduce the computational complexity of the algorithm is based on the fact that the centers of four child boxes with the same parent are located symmetrically around the center of their parent. Recall that in the second step of the upward pass, for each box, four multipole expansions associated with its four child boxes are shifted to the center of the box to form a multipole expansion. A straight forward implementation requires $4p^2$ operations per parent box. The theorem that describes the technique and the number of operations is recorded below.

Theorem 4.5. *Suppose that* $z_k = \frac{1+i}{\sqrt{2}} R * i^k$, *here* $i = (-1)^{\frac{1}{2}}$, *and*

$$\Phi^k(z) = \sum_{n=1}^{p} \frac{a_n^k}{(z - z_k)^k} \tag{4.58}$$

is a p-term multipole expansion of the potential due to a set of charges located inside the circle of radius R *centered at* $z_k, k = 1, 2, 3, 4$. *Then for* z *outside the circle of radius* $2R$ *and centered at the origin,*

$$\sum_{k=1}^{4} \Phi^k(z) = \sum_{l=1}^{p} \frac{b_l}{z^l}, \tag{4.59}$$

where

$$b_l = \sum_{k=1}^{4} \sum_{j=1}^{l} C_{l-1}^{j-1} z_k^{l-j} a_j^k. \tag{4.60}$$

Furthermore, the coefficients $b_1, b_2, ...b_p$ *can be computed in* $p^2 + 3p$ *operations.*

For proof, see [Hrycak98].

In summary, in the second step of the upward pass, the operation count is reduced to

$$\sum_{l=2}^{n-1} 4^l * (p^2 + 3p) \approx \frac{1}{3}\frac{N}{s}(p^2 + 3p). \tag{4.61}$$

And in the first step of the downward pass, the operation counts is reduced to

$$\sum_{l=2}^{n} 4^l * p^2 + \sum_{l=2}^{n} 4^l * (4pq + 27q + 4pq) \approx \frac{4}{3}\frac{N}{s}(p^2 + 8pq + 27q). \tag{4.62}$$

The total cost is

$$2Np + \frac{1}{3}\frac{N}{s}(5p^2 + 8pq + 27q + 3p) + 9Ns. \tag{4.63}$$

We can see that the operation cost is greatly reduced compared to the operation counts $2Np + \frac{116}{3}\frac{N}{s}p^2 + 9Ns$ of the original translation operators. Here q depends on the precision required.

Chapter 5

Efficient Translation Operators in Two Dimensions

From the previous chapter, we know that reducing the complexity of the translation operators is of fundamental importance to the FMM. In this chapter we present new efficient translation operators based on matrix decompositions, discuss how the complexity of these operators is reduced. We will not discuss how to achieve accuracy with the stability problems associated with them, and how to efficiently implement them, instead we will leave these issues to Chapter 9 where they will be treated with a same technique. Throughout this dissertation, we use $diag([x_1, x_2, \cdots, x_n]')$ to denote a diagonal matrix of order $n \times n$ with $\{x_1, x_2, \cdots, x_n\}$ as its diagonal entries.

5.1 Decomposition of translation operators in two dimensions

In this section we present several different ways to represent these three translation matrices and discuss how the number of operations for processing them can

be reduced.

5.1.1 Multipole translation matrix

It is straightforward to see that a truncated multipole-to-multipole translation matrix ($SS(z)$) can be represented as the product of three matrices as in the following lemma.

Lemma 5.1.

$$SS(z) = diag\begin{bmatrix} 1 \\ z \\ z^2 \\ z^3 \\ \vdots \\ z^{p-1} \end{bmatrix} \cdot \mathbf{P} \cdot diag\begin{bmatrix} 1 \\ z^{-1} \\ z^{-2} \\ z^{-3} \\ \vdots \\ z^{1-p} \end{bmatrix} \tag{5.1}$$

where

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 2 & 1 & 0 & \cdots & 0 \\ 1 & 3 & 3 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{p-1}^0 & C_{p-1}^1 & C_{p-1}^2 & C_{p-1}^3 & \cdots & C_{p-1}^{p-1} \end{bmatrix} \tag{5.2}$$

is a Pascal matrix.

Proof. This lemma can be proved in a way similar to the proof of (3.2). Notice that the (n, m) entry SS_{nm} of the matrix $SS(z)$ is

$$SS_{nm} = \begin{cases} C_n^m z^{n-m} & \text{if } n \geq m \\ 0 & \text{if } n < m \end{cases}. \tag{5.3}$$

It is easy to see that every entry in n-th row of the matrix has a common factor z^n, and every entry in m-th column of the matrix has a common factor z^{-m}. We can take out the common factor z^n of the n-th row and the common factor z^{-m} of the m-th column, and multiply from left side an identity matrix with the n-th entry in the diagonal replaced by z^n and multiply from right side an identity matrix with the m-th entry in the diagonal replaced by z^{-m}. This can be done for every row and column. Therefore we have factored the Pascal matrix into products of matrices with a Pascal matrix P in the middle and p diagonal matrices in the left, and p diagonal matrices in the right. Multiplying the diagonal matrices in the left and the right respectively, we have the decomposition we wanted. ■

In this lemma a constant matrix P, which is independent of the translation parameter z, is isolated from the translation matrix. This is the starting point of this research. Similar properties are later discovered for all other translation operators in 2D and all translation operators in 3D. After these properties are found, we are facing the multiplication of the constant matrix and a vector. This motivates us to find ways to represent the constant matrix. With a constant matrix, we can precompute or even approximate the matrix with some structured matrices which admit fast product before the real calculation. Indeed, we have shown a number of ways to decompose this Pascal matrix exactly in Chapter 3. Following results in Chapter 3, we have the following theorem.

Theorem 5.2. *Translation of a multipole expansion can be done in* $O(p \log p)$ *operations.*

Proof. It is clear from the previous lemma that a multipole translation involves three matrix-vector multiplications, two of which involve diagonal matrices and one that involves a Pascal matrix. From Chapter 3 we know that a Pascal matrix

can be multiplied by a vector in $O(p \log p)$ time. Therefore, the total cost requires $O(p \log p)$ operations. ■

Exponential series

A multipole translation operator can also be expressed as an exponential of a matrix. This is inspired by the property,

$$SS(z_1 + z_2) = SS(z_1) * SS(z_2), \tag{5.4}$$

and can be obtained by differentiate the matrix $SS(z)$. This could be used to approximate the translation operator with a few terms when $|z|$ is very small.

$$SS(z) = e^{Hz} = \sum_{n=0}^{\infty} \frac{H^n}{n!} z^n \tag{5.5}$$

where

$$H = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & p-1 & 0 \end{bmatrix} \tag{5.6}$$

The details can be found in [Call93].

5.1.2 Local translation matrix

Similar to the multipole translation matrix, a truncated $RR(z)$ matrix can be represented as the product of three matrices as in the following lemma.

Lemma 5.3.

$$RR(z) = diag \begin{bmatrix} 1 \\ -z^{-1} \\ z^{-2} \\ -z^{-3} \\ \vdots \\ (-z)^{1-p} \end{bmatrix} \cdot \mathbf{P}' \cdot diag \begin{bmatrix} 1 \\ -z^{1} \\ z^{2} \\ -z^{3} \\ \vdots \\ (-z)^{p-1} \end{bmatrix} \tag{5.7}$$

This lemma can be proved in exactly the same way as (5.1). We omit its proof. Again in this lemma a constant matrix P', which is independent of the translation parameter z, is isolated from the translation matrix. We have shown a number of ways, which allow fast product, to decompose the transpose of a Pascal matrix in Chapter 3. Therefore we also have the following similar theorem.

Theorem 5.4. *Translation of a local expansion can be done in $O(p \log p)$ operations.*

Proof. It is clear from the previous lemma that a local translation involves three matrix-vector multiplications, two of which involve diagonal matrices and one that involves the transpose of a Pascal matrix. From Chapter 3 we know that the transpose of a Pascal matrix can be multiplied by a vector in $O(p \log p)$ time. Therefore, the total cost requires $O(p \log p)$ operations. ■

5.1.3 Multipole to local translation matrix

A truncated $SR(z)$ matrix can be also represented as the product of three matrices as in the following lemma.

Lemma 5.5.

$$SR(z) = diag \begin{bmatrix} 1 \\ z^{-1} \\ z^{-2} \\ z^{-3} \\ \vdots \\ z^{1-p} \end{bmatrix} \cdot \mathbf{PP} \cdot diag \begin{bmatrix} -z^{-1} \\ z^{-2} \\ -z^{-3} \\ z^{-4} \\ \vdots \\ (-z)^{-p} \end{bmatrix} \tag{5.8}$$

where

$$\mathbf{PP} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & C_{p-1}^{0} \\ 1 & 2 & 3 & 4 & \cdots & C_{p}^{1} \\ 1 & 3 & 6 & 10 & \cdots & C_{p+1}^{2} \\ 1 & 4 & 10 & 20 & \cdots & C_{p+2}^{3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ C_{p-1}^{0} & C_{p}^{1} & C_{p+1}^{2} & C_{p+2}^{3} & \cdots & C_{2p-2}^{p-1} \end{bmatrix} \tag{5.9}$$

This lemma can be proved in exactly the same way as (5.1). We omit its proof. Again in this lemma a constant matrix PP, which is independent of the translation parameter z, is isolated from the translation matrix. We have shown a number of ways, which allow fast product, to decompose the matrix PP in Chapter 3. Therefore we also have the following similar theorem.

Theorem 5.6. *Translation of a multipole expansion to a local expansion can be done in $O(p \log p)$ operations.*

Proof. It is clear from the previous lemma that a multipole to local translation involves three matrix-vector multiplications, two of which involve diagonal matrices and one that involves the matrix PP. From Chapter 3 we know that the

matrix PP can be multiplied by a vector in $O(p \log p)$ time. Therefore, the total cost requires $O(p \log p)$ operations. ■

We have factored out a constant matrix in each of the three translation matrices. This is a key step. Its importance can never be overstated in this research, since for a constant matrix, we can almost always come up with a number of different ways to represent it, which can speed up the matrix-vector product. This can open some further research opportunities.

5.2 Complexity analysis.

From Theorems 4.2, 4.3, and 4.4, we know that order of all three original translation operators is of the order of $O(p^2)$. A direct implementation of these operators involving a straightforward complex matrix-vector product requires $8p^2$ flops. Our efficient implementation of the same operations based on matrix decomposition involving the fast Fourier transform reduces number of operations in matrix-vector product to $20p \log 2p$ flops, with one precomputed FFT. That is $O(p^2)$ operations in the original algorithm is reduced to approximately $\frac{5}{2}p \log(2p)$ operations in the new algorithm. Let us see how this would affect the overall complexity.

We know the total cost of the original algorithm for all five steps is approximately

$$2Np + \frac{116}{3}\frac{N}{s}p^2 + 9Ns, \tag{5.10}$$

where s is the maximum number of particles per box at the finest level. With $s \approx \frac{\sqrt{348}}{9}p$, the total number of operations is approximately

$$40Np. \tag{5.11}$$

In our new algorithm, the total cost would be

$$2Np + \frac{116}{3}\frac{N}{s} * \frac{5}{2}p\log(2p) + 9Ns. \tag{5.12}$$

With $s \approx \frac{(870p\log(2p))^{0.5}}{9}$, the improved count is approximately

$$\left(2 + 60\sqrt{\frac{\log(2p)}{p}}\right) Np. \tag{5.13}$$

That is when $p = 12$, the two algorithms should break even. For increased precision of about 15 digits, from error bound (4.46), we need $p = 55$. The number of operations in current algorithm is $1265N$, while the number of operations in the old algorithm is $2200N$. For more extended precision, the new algorithm should show more advantages than the old one. We would like to note that choosing a proper parameter s, which determines number of levels in the data structure, has a major effect over the efficiency of the program. For example, in the original algorithm proposed in [Greengard88], if s is chosen to be 1, for $p = 55$, the number of operations is $1.1709 * 10^5 N$; and if new translation operators are used, then the number of operations should improved to $3.6173 * 10^4 N$. Since the error bound (4.46) is very conservative, the number p of terms required is much smaller for the desired accuracy in the real calculation. Numerical experiments also indicate that the FMM with the new fast translation operators are faster than what theoretical analysis predicts. For numerical results, see Chapter 10.

5.3 Alternative strategy for computing the interaction list

In Theorem 4.5, note that $z_k = \frac{1+i}{\sqrt{2}} R * i^k$. The essential point in this theorem to reduce the complexity is that the centers of child boxes can be expressed as

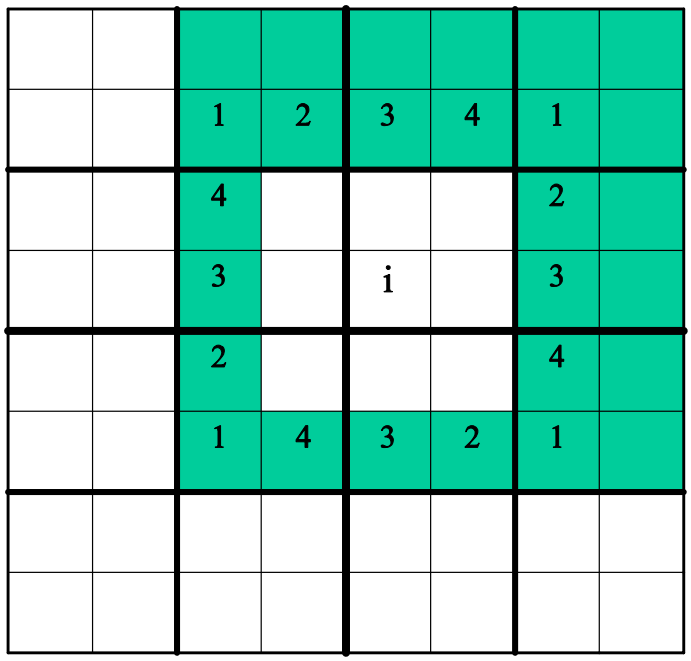

Figure 5.1: Lists of 4 boxes in an interaction list whose centers form a square

$z_k = z * i^k$. That is, this trick can be similarly employed to the multipole to local translation operator to reduce the operation count of conversion of a full interaction list of a box from $27p^2$ to $15p^2+12p$. (Look at figure 5.1, sixteen boxes in a full interaction list are labelled with numbers 1,2,3,4. Centers of four boxes with the same number inside form a square. Therefore there are four squares that can utilize this trick.).

Chapter 6

The FMM in Three Dimensions

In this chapter we follow the work of Greengard of the fast multipole method (FMM) in his dissertation [Greengard88]. The purpose is to establish the ideas about the FMM in 3D, translation involved, and representation of those operations as matrices. In Chapters 7 and 8, we will provide decomposition of these matrices to speed up the translation step. We will also provide a complexity analysis of the FMM with the accelerated translation operations.

In this chapter we first present how to represent a potential field in three dimensional space as a multipole expansion. Then we give results on how to calculate the coefficients of a new multipole/local expansion that results from translating an existing multipole/local expansion, and coefficients of a local expansion resulting from the translation of an existing multipole expansion, and rewrite them as translation operators in the form of matrix-vector product. Next we are ready to discuss the differences between the FMM in two dimensions and the FMM in three dimensions, and analyze the complexity of the algorithm. Finally, we review previous work that has been done to improve the complexity of the translation operators and hence reduce the overall complexity of the algorithm. A lot of material here such as the multipole expansion of a field, spher-

ical harmonics, translation theorems, and the FMM algorithm can be found in Greengard's dissertation [Greengard88]. The complexity analysis can be found in [Greengard97]. We use a different definition of spherical harmonics from that in Greengard and Rokhlin 's work [Greengard88, Greengard97]. We also present a slightly different error bound (6.42) for the multipole to local translation theorem.

6.1 The series expansion of a potential field in three dimensional space

In three dimensions, the potential at $P = (x, y, z) \in R^3$ due to a point charge of unit intensity at $P_0 = (x_0, y_0, z_0)$ is given by:

$$\phi(x, y, z) = \frac{1}{R}, \tag{6.1}$$

where $R = \|P - P_0\| = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}$, denoting the distance between points P and P_0.

A series expansion for the potential at P in terms of its distance r from the origin O will lead us to a multipole expansion[Kellogg53]. A natural coordinate system of choice to express distance from the origin would be a spherical coordinate system. Let $P = (r, \theta, \phi)$ and $P_0 = (r_0, \theta_0, \phi_0)$ in the spherical coordinates system. The law of cosines yields

$$R^2 = r^2 + r_0^2 - 2rr_0 \cos\gamma, \tag{6.2}$$

where γ is the angle between the rays OP and OP_0, and satisfies

$$\cos\gamma = \cos\theta\cos\theta_0 + \sin\theta\sin\theta_0\cos(\phi - \phi_0). \tag{6.3}$$

The potential at P of a unit particle at P_0 can be rewritten

$$\frac{1}{R} = \frac{1}{r\sqrt{1 - 2\frac{r_0}{r}\cos\gamma + \frac{r_0^2}{r^2}}} \tag{6.4}$$

Now set $\frac{r_0}{r} = \mu$ and $\cos\gamma = u$, then

$$\frac{1}{R} = \frac{1}{r\sqrt{1 - 2u\mu + \mu^2}} \tag{6.5}$$

But $(1 - 2u\mu + \mu^2)^{-\frac{1}{2}}$ is the generating function of Legendre polynomials,

$$\frac{1}{\sqrt{1 - 2u\mu + \mu^2}} = \sum_{n=0}^{\infty} P_n(u)\mu^n \tag{6.6}$$

where P_n is the Legendre polynomial of degree n, and $\mu < 1$. So for $r > r_0$, we have the multipole expansion

$$\frac{1}{R} = \frac{1}{r}\sum_{n=0}^{\infty} P_n(\cos\gamma)\left(\frac{r_0}{r}\right)^n = \sum_{n=0}^{\infty} r_0^n P_n(\cos\gamma)\frac{1}{r^{n+1}}. \tag{6.7}$$

Similarly for $r < r_0$, we have the local expansion

$$\frac{1}{R} = \frac{1}{r_0}\sum_{n=0}^{\infty} P_n(\cos\gamma)\left(\frac{r}{r_0}\right)^n = \sum_{n=0}^{\infty} r^n P_n(\cos\gamma)\frac{1}{r_0^{n+1}}. \tag{6.8}$$

Using the property of the Legendre polynomials that $P_n(\cos\gamma) \leq 1$, we easily obtain the following two error bounds,

$$\left|\frac{1}{R} - \sum_{n=0}^{p} r_0^n P_n(\cos\gamma)\frac{1}{r^{n+1}}\right| \leq \frac{1}{r - r_0}\left(\frac{r_0}{r}\right)^{p+1}, \text{ for } r > r_0; \tag{6.9}$$

$$\left|\frac{1}{R} - \sum_{n=0}^{p} r^n P_n(\cos\gamma)\frac{1}{r_0^{n+1}}\right| \leq \frac{1}{r_0 - r}\left(\frac{r}{r_0}\right)^{p+1}, \text{ for } r < r_0. \tag{6.10}$$

Now let us define the spherical harmonics of degree n and order m

$$Y_n^m(\theta,\phi) = (-1)^m\sqrt{\frac{(n-m)!}{(n+m)!}}P_n^m(\cos\theta)e^{im\phi}, \tag{6.11}$$

where $P_n^m(x)$ are the associated Legendre functions, which can be defined by Rodrigues's formula

$$P_n^m(x) = (1-x^2)^{m/2}\frac{d^m}{dx^m}P_n(x), \text{ for } m > 0, \tag{6.12}$$

and

$$P_n^{-m}(x) = (-1)^m\frac{(n-m)!}{(n+m)!}P_n^m(x). \tag{6.13}$$

The associated Legendre polynomial $P_n(x)$ can be produced from

$$P_n^m(x) = \frac{(1-x^2)^{m/2}}{2^n n!}\frac{d^{n+m}}{dx^{n+m}}(x^2-1)^n, \text{ for } m = -l, \cdots, -1, 0, 1, \cdots, l. \tag{6.14}$$

The associated Legendre functions satisfy the following recursion

$$(n-m+1)P_{n+1}^m = (2n+1)xP_n^m(x) - (n+m)P_{n-1}^m(x), \tag{6.15}$$

and

$$P_n^{m+2} = 2\,(m+1)\,\frac{x}{(1-x^2)^{\frac{1}{2}}}P_n^{m+1}(x) - (n-m)(n+m+1)P_n^m(x). \tag{6.16}$$

The spherical harmonics are given a variety of definitions in the literature, our definition agrees with that of Edmonds [Edmonds60], except for a factor $\left(\frac{2n+1}{4\pi}\right)^{1/2}$. We note as well that for $m < 0$ this definition disagrees with that used by Greengard and Rokhlin [Greengard88, Greengard97, Cheng99] by a factor $(-1)^m$. Using the spherical harmonic addition theorem

$$P_n(\cos\gamma) = \sum_{m=-n}^{n}(-1)^m Y_n^{-m}(\theta_0,\phi_0)Y_n^m(\theta,\phi), \tag{6.17}$$

we have the multipole expansion,

$$\frac{1}{R} = \sum_{n=0}^{\infty} r_0^n\frac{1}{r^{n+1}}\sum_{m=-n}^{n}(-1)^m Y_n^{-m}(\theta_0,\phi_0)Y_n^m(\theta,\phi). \tag{6.18}$$

Therefore it is straightforward to form a multipole expansion for a field containing a set of k charges. The formal description is in the following theorem [Greengard88].

Theorem 6.1. *(Multipole Expansion) Suppose that*

$$\Phi(z) = \sum_{i=1}^{k} \frac{q_i}{\|P_i - P\|} \tag{6.19}$$

is the potential due to a set of k charges of strengths $\{q_i,\ i = 1, \ldots, k\}$ *located at points* $\{P_i = (r_i, \theta_i, \phi_i),\ i = 1, \ldots, k\}$*, with* $|r_i| < a$*. Then for any* $P = (r, \theta, \phi) \in R^3$ *with* $|r| > a$*, the potential* $\Phi(z)$ *can be expressed as the following*

$$\Phi(z) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{M_n^m}{r^{n+1}} Y_n^m(\theta, \phi), \tag{6.20}$$

where

$$M_n^m = \sum_{i=1}^{k} (-1)^m q_i * r_i^n * Y_n^{-m}(\theta_i, \phi_i). \tag{6.21}$$

Furthermore, for any $p \geq 1$*, the error of truncating the infinite summation to order p is given by*

$$\left| \Phi(z) - \sum_{n=0}^{p} \sum_{m=-n}^{n} \frac{M_n^m}{r^{n+1}} Y_n^m(\theta, \phi) \right| \leq \frac{A}{r - a} (\frac{a}{r})^{p+1}, \tag{6.22}$$

where

$$A = \sum_{i=1}^{k} |q_i|. \tag{6.23}$$

Its proof can be found in [Greengard88].

With this theorem, we can use a single multipole expansion to express the far field potential due to multiple charges inside a sphere.

Note that in 3D we have p^2 terms of coefficients in the expansions as opposed to p terms of coefficients in the 2D case.

6.2 Translation operators in the three dimensional Laplace equation and their matrix forms

In this section we will provide three modified multipole translation theorems given in [Greengard88] and [Cheng99], which are key to the FMM algorithm.

Theorem 6.2. *(Translation of a Multipole Expansion) Suppose that a set of N charges of strengths $\{q_i,\ i = 1, \ldots, N\}$ are located at points $\{P_i,\ i = 1, ..., N\}$ inside the sphere S of radius a with center at $Q = (\rho, \alpha, \beta)$ and that for any point $P = (r, \theta, \phi)$ outside the sphere S, the potential due to these charges is given by multipole expansion*

$$\Phi(P) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O_n^m}{r'^{n+1}} Y_n^m(\theta', \phi'), \tag{6.24}$$

Where $P - Q = (r', \theta', \phi')$. Then the potential at any point P outside of the sphere S_1 of radius $(a + \rho)$ with center at the origin can be expressed as,

$$\Phi(P) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} \frac{M_j^k}{r^{j+1}} Y_j^k(\theta, \phi), \tag{6.25}$$

where

$$M_j^k = \sum_{n=0}^{j} \sum_{m=\max(k+n-j,-n)}^{\min(k+j-n,n)} \frac{O_{j-n}^{k-m}(-1)^m A_n^m A_{j-n}^{k-m} \rho^n Y_n^{-m}(\alpha, \beta)}{A_j^k}, \tag{6.26}$$

with

$$A_n^m = \frac{1}{\sqrt{(n-m)!(n+m)!}}. \tag{6.27}$$

Furthermore, for any $p \geq 1$, the error of the truncated series is given by

$$\left| \Phi(P) - \sum_{j=0}^{p} \sum_{k=-j}^{j} \frac{M_j^k}{r^{j+1}} Y_j^k(\theta, \phi) \right| \ \leq \left(\frac{\sum_{i=1}^{N} |q_i|}{r - (a + \rho)|} \right) \left(\frac{a + \rho}{r} \right)^{p+1}. \tag{6.28}$$

Its proof can be found in [Greengard88] and [Cheng99].

Remark: We can rewrite coefficients of the new expansion in terms of those of the old expansion in matrix form as follows,

$$M = SS(\rho, \alpha, \beta) * O \tag{6.29}$$

where

$$M = \left[\begin{array}{cccccccccc} M_0^0 & M_1^{-1} & M_1^0 & M_1^1 & ... & M_p^{-p} & M_p^{-p+1} & M_p^{-p+2} & ... & M_p^p \end{array} \right]', \tag{6.30}$$

$$O = \left[\begin{array}{cccccccccc} O_0^0 & O_1^{-1} & O_1^0 & O_1^1 & ... & O_p^{-p} & O_p^{-p+1} & O_p^{-p+2} & ... & O_p^p \end{array} \right]', \tag{6.31}$$

and $SS(\rho, \alpha, \beta)$ is the dense matrix mapping the multipole expansion coefficients vector O into the shifted multipole expansion coefficients vector M.

Theorem 6.3. *(Translation of a Local Expansion) Suppose that*

$$\Phi(P) = \sum_{n=0}^{p} \sum_{m=-n}^{n} O_n^m r'^n Y_n^m(\theta', \phi') \tag{6.32}$$

is a local expansion centered at $Q = (\rho, \alpha, \beta)$, Where $P = (r, \theta, \phi)$, and $P - Q = (r', \theta', \phi')$. Then the local expansion centered at origin is

$$\Phi(P) = \sum_{j=0}^{p} \sum_{k=-j}^{j} L_j^k r^j Y_j^k(\theta, \phi), \tag{6.33}$$

where

$$L_j^k = \sum_{n=j}^{p} \sum_{m=k-n+j}^{k-j+n} \frac{O_n^m (-1)^{n-j} A_j^k A_{n-j}^{m-k} \rho^{n-j} Y_{n-j}^{m-k}(\alpha, \beta)}{A_n^m}, \tag{6.34}$$

with A_n^m defined by (6.27).

Its proof can be found in [Greengard88] and [Cheng99].

Remark: We can rewrite coefficients of the new expansion in terms of those of old expansion in the matrix format as follows,

$$L = RR(\rho, \alpha, \beta) * O \tag{6.35}$$

where

$$O = \begin{bmatrix} O_0^0 & O_1^{-1} & O_1^0 & O_1^1 & ... & O_p^{-p} & O_p^{-p+1} & O_p^{-p+2} & ... & O_p^p \end{bmatrix}', \tag{6.36}$$

$$L = \begin{bmatrix} L_0^0 & L_1^{-1} & L_1^0 & L_1^1 & ... & L_p^{-p} & L_p^{-p+1} & L_p^{-p+2} & ... & L_p^p \end{bmatrix}', \tag{6.37}$$

and $RR(\rho, \alpha, \beta)$ is the dense matrix mapping the local (regular) expansion coefficients vector O into the shifted local (regular) expansion coefficients vector L.

Theorem 6.4. *(Conversion of a Multipole to a Local Expansion) Suppose that a set of N charges of strengths $\{q_i,\ i = 1, \ldots, N\}$ are located at points $\{P_i,\ i = 1, ..., N\}$ inside the sphere S_1 of radius R_1 with center at $Q = (\rho, \alpha, \beta)$ and $\rho > R_1 + R_2$. Suppose further that the potential at any point P outside the sphere S_1 due to these charges is given by multipole expansion*

$$\Phi(P) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O_n^m}{r'^{n+1}} Y_n^m(\theta', \phi'), \tag{6.38}$$

Where $P - Q = (r', \theta', \phi')$. Then the potential at any point $P = (r, \theta, \phi)$ inside of the sphere S_2 which lies outside of the sphere S_1 centered at the origin with radius R_2 can be expressed as,

$$\Phi(P) = \sum_{j=0}^{\infty} \sum_{k=-j}^{j} L_j^k r^j Y_j^k(\theta, \phi), \tag{6.39}$$

where

$$L_j^k = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \frac{O_n^m (-1)^{n+k} A_n^m A_j^k Y_{j+n}^{m-k}(\alpha, \beta)}{A_{j+n}^{m-k} \rho^{j+n+1}}, \tag{6.40}$$

with A_n^m *defined by (6.27). Furthermore, for any* $p \geq 1$*, the error of truncating the infinite summation is given by*

$$\left| \Phi(P) - \sum_{j=0}^{p} \sum_{k=-j}^{j} L_j^k r^j Y_j^k(\theta, \phi) \right| \leq \left(\frac{\sum_{i=1}^{N} |q_i|}{\rho - R_1 - r} \right) \left| \frac{r}{\rho - R_1} \right|^{p+1}. \tag{6.41}$$

In FMM, L_j^k*,* $j = 0, 1, \cdots, p$*,* $k = -j, \cdots, -1, 0, 1, \cdots, j$*, are calculated by truncating* O_n^m*, that is ,* $O_n^m = 0$*, for* $n = p+1, \cdots$*. Let us in this case denote* L_j^k *by* $\hat{L}_j^k$*, we have the following error bound,*

$$\begin{aligned} \left| \Phi(P) - \sum_{j=0}^{p} \sum_{k=-j}^{j} \hat{L}_j^k r^j Y_j^k(\theta, \phi) \right| \quad \leq \quad & \left(\frac{\sum_{i=1}^{N} |q_i|}{\rho - R_1 - r} \right) \left| \frac{r}{\rho - R_1} \right|^{p+1} + \\ & \left(\frac{\sum_{i=1}^{N} |q_i|}{\rho - R_1 - r} \right) \left| \frac{R_1}{\rho - r} \right|^{p+1} \end{aligned} \tag{6.42}$$

A proof of this theorem can be found in [Greengard88] and [Cheng99]. The last inequality can be proved in a way similar to that in 4.30.

remarks:

1. Note that the bound (6.42) estimates the error of the final shifted truncated expansion from the true potential. Thus this bound is all we need in the error analysis.

2. We can rewrite coefficients of the new expansion in terms of those of old expansion in matrix form as follows,

$$L = SR(\rho, \alpha, \beta) * O \tag{6.43}$$

where

$$O = \left[\begin{array}{cccccccccc} O_0^0 & O_1^{-1} & O_1^0 & O_1^1 & ... & O_p^{-p} & O_p^{-p+1} & O_p^{-p+2} & ... & O_p^p \end{array} \right]', \tag{6.44}$$

$$L = \left[\begin{array}{cccccccccc} L_0^0 & L_1^{-1} & L_1^0 & L_1^1 & ... & L_p^{-p} & L_p^{-p+1} & L_p^{-p+2} & ... & L_p^p \end{array} \right]', \tag{6.45}$$

and $SR(\rho, \alpha, \beta)$ is the dense matrix mapping the multipole (singular) expansion coefficients vector O into the shifted local (regular) expansion coefficients vector L.

The Fast Multipole Method (FMM) relies on these three linear operators to perform all of the necessary manipulations. It is clear from the formulas that the matrices representing these operators are dense, so that each translation would require $O(p^4)$ operations if we apply the corresponding matrix to truncated expansion with $O(p^2)$ coefficients. This is usually the primary obstacle to good performance of most existing FMM implementations.

6.3 The fast multipole method

Now that we have all the necessary translation operators we are ready to describe the FMM in three dimensions. Despite the increased mathematical complexity of the three-dimensional case, the framework of the FMM algorithm in three dimensions is the same as in two dimensions. It also has two parts, the construction of the data structure, and the computational step – two passes, five steps. We do not repeat the whole algorithm, instead we just note the modifications needed here. In three dimensions, the computational box is a cube that contains the whole computational domain. Therefore each box at level l will be equally subdivided into eight child cube boxes at level $l + 1$ instead of four square boxes.

In general, the number of neighbors of a box increases from 9 to 27 and the size of the interaction list increases from 27 to 189.

Since the radius of the sphere that encloses a cube of length $2a$, which is $3^{0.5}a$, is bigger than the radius of the circle that encloses a square of length $2a$, which is $2^{0.5}a$, the error decays much more slowly, like $(0.76)^p$. Further, the current number of terms is p^2, therefore the number of terms needed is a much bigger number. In the original FMM proposed in Greengard's thesis [Greengard88], he suggested to increase the neighbor list to include "second nearest neighbor", so that boxes which interact via multipole expansions are separated by at least two intervening boxes of the same size. The error then decays faster, like $(0.4)^p$. However, the number of neighbors increases from 27 to 125 and the size of the interaction list increases from 189 to 875. This proves to be too much of a burden to the FMM. Later more efficient new translation operators are developed [Greengard97], the FMM uses the scheme with 27 neighbors and 189 as the size of interaction list for a box.

6.4 Error analysis

The error analysis can be done in a similar way to that of the two dimensional case. The key is to realize that although truncated operators in Theorems 6.1 and 6.2 cause errors, all the errors can be estimated using the single error bound (6.42) given in Theorem 6.4.

For a particular evaluation location y, the error of computed potential at this point caused by the entire system is

$$E(y) = \left| \sum_{i=1}^{N} \phi(y, x_i) - \sum_{i=1}^{N} \hat{\phi}(y, x_i) \right|, \tag{6.46}$$

where $\phi(y, x_i)$ is the potential at point y induced by a particle at point x_i and $\hat{\phi}(y, x_i)$ is the computed potential. From the process of the FMM, we know that the particles in the neighbor boxes of the box containing y at the finest level are handled directly, therefore there is no error in these calculations. All the rest of particles are contained in boxes that belong to different level of interaction lists. Let us denote by IL_l the interaction list for the box that contains point y at level l, P_b the set of all particles contained in box b. We have,

$$\begin{aligned} E(y) &= \left| \sum_{i=1}^{N} \Big(\phi(y, x_i) - \hat{\phi}(y, x_i) \Big) \right| \\ &\leq \left| \sum_{l=2}^{n} \sum_{b \in IL_l} \sum_{i \in P_b} \Big(\phi(y, x_i) - \hat{\phi}(y, x_i) \Big) \right| \end{aligned} \tag{6.47}$$

Recall that in the FMM, there are errors in steps 1, 2, and 3, but the total error in the final local expansions $\Psi_{n,j}$ is controlled by (6.42). Denote respectively R_1, ρ, r in the error bound (6.42) at level l by R_l, ρ_l, r_l. We have

$$E(y) \leq \sum_{l=2}^{n} \sum_{b \in IL_l} \left(\frac{\sum_{i \in P_b} |q_i|}{\rho_l - R_l - r_l} \left| \frac{r_l}{\rho_l - R_l} \right|^{p+1} + \right. \tag{6.48}$$

$$\left. \frac{\sum_{i \in P_b} |q_i|}{\rho_l - R_l - r_l} \left| \frac{R_l}{\rho_l - r_l} \right|^{p+1} \right) \tag{6.49}$$

$$\leq \sum_{l=2}^{n} \sum_{b \in IL_l} \frac{\sum_{i \in P_b} |q_i|}{(2 - 3^{0.5}) a_l} \left(\frac{3^{0.5}}{4 - 3^{0.5}} \right)^{p+1}$$

$$\leq \sum_{l=2}^{n} \frac{\sum_{b \in IL_l} \sum_{i \in P_b} |q_i|}{(2 - 3^{0.5}) 2^{-l}} \left(\frac{3^{0.5}}{4 - 3^{0.5}} \right)^{p+1}. \tag{6.50}$$

Assuming particles are homogeneously distributed and with same magnitude of the intensities, and ignoring the boundary effect, we have in general,

$$\sum_{b \in IL_l} \sum_{i \in P_b} |q_i| \leq \frac{C_1 N}{8^{l-2}}, \tag{6.51}$$

for some constant $C_1 \leq 1$. Therefore,

$$E(y) \leq \sum_{l=2}^{n} \frac{C_1 N 2^l}{(2-3^{0.5})8^{l-2}} \left(\frac{3^{0.5}}{4-3^{0.5}}\right)^{p+1} \tag{6.52}$$

$$\leq \sum_{l=2}^{n} \frac{64 C_1 N}{(2-3^{0.5})4^{l}} \left(\frac{3^{0.5}}{4-3^{0.5}}\right)^{p+1}. \tag{6.53}$$

So the total error for the entire system is

$$E(y) \leq CN \left(\frac{3^{0.5}}{4-3^{0.5}}\right)^{p+1}, \tag{6.54}$$

where $C = \frac{16C_1}{3(2-\sqrt{3})}$ is a constant. Of course, this error bound is very conservative, even though it is rigorous.

6.5 Complexity of the FMM

We now go on to analyze complexity. These results are adapted from [Greengard97]. Suppose there is a total of N particles in the system. Set the number of levels to $n \approx \log_8 N$ and the length of multipole and local expansions to $p^2 \approx (\log(\frac{1}{\epsilon}))^2$. Then the number of boxes at the finest level is 8^n, and the number of particles per box is $s \approx \frac{N}{8^n}$. We conclude that the number of operations required for the initialization of the FMM algorithm is again usually $O(N \log N)$ and the number of operations required for the computational steps, i.e. the upward pass and downward pass, is $O(N)$ for a fixed $p \approx \log_c(\frac{1}{\epsilon})$, where $c = \frac{\sqrt{3}}{4-\sqrt{3}}$. (Similar to (4.46), if we want a fixed precision, then $p \approx \log_c(\frac{N}{\epsilon})$. The resulting algorithm can be of complexity of $O(N \log^2 N)$ or $O(N \log^4 N)$.) It is again true that the cost of the computational step dominates the total cost. An efficient second part is central to reduce the total complexity. The complexity of each step of the second part of FMM is analyzed in detail as the following.

In step 1, for each particle, the potential induced by it is expressed as a p^2-term expansion about the center of the box that contains it at the finest level. The number of operations required at this step is $O(Np^2)$.

In step 2, each translation requires $O(p^4)$ operations. For level l, there are 8^l boxes, and for each box, it requires 8 translations. So the total number of operations is the order of

$$\sum_{l=2}^{n-1} 8 * 8^l * p^4 = \sum_{l=3}^{n} 8^l * p^4 = \frac{8^3 - 8^{n+1}}{1-8} p^4 \approx \frac{8}{7} 8^n p^4 = \frac{8}{7} \frac{N}{s} p^4. \tag{6.55}$$

In step 3, local translation operator and multipole to local translation operator are used. They both require $O(p^4)$ operations. At level l, there are 8^l boxes, and for each box, it requires one local translation and at most 189 multipole to local translations. The operation count in this step is

$$\sum_{l=2}^{n} 8^l * p^4 + \sum_{l=2}^{n} 8^l * p^4 * 189 = \frac{8^2 - 8^{n+1}}{1-8} * 190 * p^4 \approx \frac{8}{7} * 8^n * 190 p^4 = \frac{1520}{7} \frac{N}{s} p^4. \tag{6.56}$$

In step 4, each particle requires one evaluation of a p^2-term local expansion. This step requires $O(Np^2)$ operations.

In step 5, each particle in a box at the finest level interacts with particles contained at its 27 neighboring boxes. And there are about s particles per box. Therefore this step requires $O(27Ns)$ operations.

The total cost for all five steps is approximately

$$2Np^2 + \frac{1528}{7} \frac{N}{s} p^4 + 27Ns. \tag{6.57}$$

It is clear that the total complexity can be optimized by selecting an appropriate parameter s. With $s \approx \sqrt{\frac{1528}{189}} p^2$, the total number of operations is approximately $156Np^2$.

It is clear that the major obstacle to achieving reasonable efficiency at high precision is the cost of translation operators. There have been several improvements proposed [GreengardL88, Cheng99] since the original FMM appeared. We will discuss them in next section.

6.6 Previous work on three dimensional translation operators

In this section we first introduce the Euler angles for the rotation group [Edmonds60] to prepare some mathematical preliminaries. Then we describe how to represent each of these three translation matrices as a product of a rotation matrix, a coaxial translation matrix and another rotation matrix. It is well known that applying a rotation matrix or a coaxial translation matrix to an arbitrary vector requires only $O(p^3)$ operations. Therefore, this approach reduces the cost of translation operators from $O(p^4)$ to $O(p^3)$ [White96]. Finally the exponential representation (or "plane wave" expansions) as described in the two-dimensional space to diagonalize the multipole-to-local translation is presented, which further reduces the complexity of multipole to local translation operators.

6.6.1 The Euler angles

It is often very convenient to use the Euler Angles to link a set of three mutually perpendicular axes (x, y, z) (called frame of reference S, always assuming right-handed frame of axes) to a new set of three mutually perpendicular axes $(\hat{x}, \hat{y}, \hat{z})$ (called frame of reference $\hat{S}$) due to a rotation about the fixed origin. In the present case a new set of axes with specified $\hat{z}$ direction, which is along the

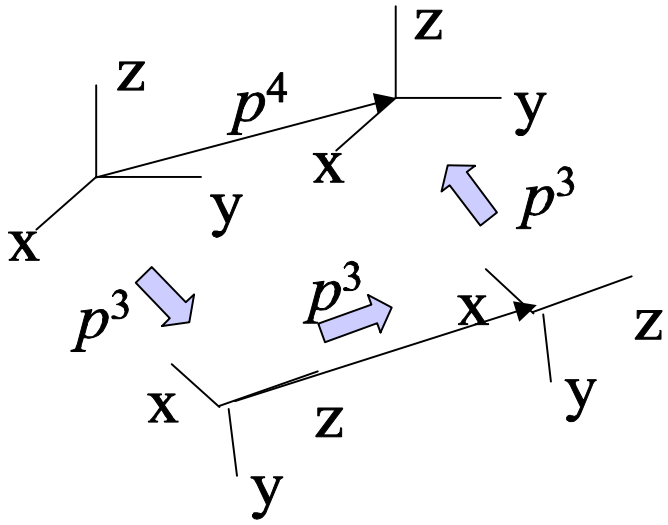

Figure 6.1: Rotation-based translations

vector starting from the center of the desired multipole or local expansion to the center of the original multipole or local expansion, can be obtained by performing three rotations about two of the three axes. The Euler angles ($\alpha\beta\gamma$) are described by a sequence of three rotations of vectors:

1. A rotation $\alpha(0 \leq \alpha < 2\pi)$ about $n_1 = (0, 0, 1)$,written as $R_z(\alpha)$.

2. A rotation $\beta(0 \leq \beta \leq \pi)$ about $n_2 = (-\sin\alpha, \cos\alpha, 0)$, written as $R_{y'}(\beta)$.

3. A rotation $\gamma(0 \leq \gamma < 2\pi)$ about $n_3 = (\cos\alpha\sin\beta, \sin\alpha\sin\beta, \cos\beta)$ written as $R_{z''}(\gamma)$.

The effect of such rotations of the frame of reference is denoted by operator $R(\ \alpha\beta\gamma)$. So we can formally express the effect of such rotations of the frame of

reference upon the spherical harmonics $Y_n^m(\theta,\phi)$ as the following equation:

$$R(\alpha\beta\gamma)Y_n^m(\theta,\phi) = Y_n^m(\hat{\theta},\hat{\phi}). \tag{6.58}$$

It is easy to see that $R(\beta 00)Y_n^m(\theta,\phi) = Y_n^m(\theta,\phi-\beta)$. Therefore, when a rotation about z-axis is performed, the resulting vector consisting of the coefficients of the new multipole or local expansion can be obtained by multiplying a diagonal matrix $D(\beta 00)$, where $D(\ \alpha\beta\gamma)$ is the matrix corresponding to the rotation $R(\alpha\beta\gamma)$. The matrix of $D(\alpha\beta\gamma)$ acting on j-th degree of spherical harmonics is denoted by $D^{(j)}(\alpha\beta\gamma)$, or $D^{(j)}$. We shall also write

$$D_{m'm}^{(j)}(0\beta 0) = d_{m'm}^{(j)}(\beta). \tag{6.59}$$

Therefore

$$D_{m'm}^{(j)}(\alpha\beta\gamma) = e^{im'\gamma}d_{m'm}^{(j)}(\beta)e^{im\alpha}. \tag{6.60}$$

6.6.2 Rotation-based translations

We denote the special cases of the matrices of the coaxial translations by $SS(\rho)$, $RR(\rho)$, $SR(\rho)$, which shift a distance ρ in the z-direction. From [Cheng99], we know that,

Lemma 6.5.

$$SS(\rho,\alpha,\beta) = D(-\beta 00)D(0-\alpha 0)SS(\rho)D(0\alpha 0)D(\beta 00), \tag{6.61}$$

$$RR(\rho,\alpha,\beta) = D(-\beta 00)D(0-\alpha 0)RR(\rho)D(0\alpha 0)D(\beta 00), \tag{6.62}$$

$$SR(\rho,\alpha,\beta) = D(-\beta 00)D(0-\alpha 0)SR(\rho)D(0\alpha 0)D(\beta 00). \tag{6.63}$$

It is clear that the cost of these operators implemented as the factorization of the above matrices is

$$O\left(p^2\right) + O\left(p^3\right) + O\left(p^3\right) + O\left(p^3\right) + O\left(p^2\right). \tag{6.64}$$

Therefore the total computational cost of the FMM is approximately

$$2Np^2 + \frac{1528}{7}\frac{N}{s}3p^3 + 27Ns. \tag{6.65}$$

With $s \approx \frac{\sqrt{10696}}{21}p^{\frac{3}{2}}$, the cost becomes

$$2Np^2 + 270Np^{\frac{3}{2}}. \tag{6.66}$$

6.6.3 Exponential representation

Very similar to the two-dimensional case, exponential representation is constructed to reduce the cost of multipole to local translation operator [Cheng99, GreengardL88]. The key idea is the same as in two dimensions, it is to represent the potential as plane wave expansion through integral and quadrature, thus to reduce the linear but dense multipole to local translation operator to a diagonal one.

They considered the potential $\frac{1}{R} = \frac{1}{\|P-P_0\|}$ at a evaluation location $P = (x, y, z)$ induced by a unit charge at $P_0 = (x_0, y_0, z_0)$. First the potential can be represented as an integral when $(z - z_0) > 0$, then

$$\frac{1}{R} = \frac{1}{2\pi}\int_0^\infty e^{-\lambda(z-z_0)}\int_0^{2\pi} e^{i\lambda((x-x_0)\cos\alpha+(y-y_0)\sin\alpha)}\mathrm{d}\alpha\mathrm{d}\lambda \tag{6.67}$$

Then this integral is approximated by the following formula using finite quadratures. Thus, the potential of a unit charge at the point z_0 can be approximated by a linear combination of exponentials or plane waves as the following

$$\left|\frac{1}{R} - \sum_{k=1}^{s(\epsilon)}\frac{\omega_k}{M_k}\sum_{j=1}^{M_k} e^{-\lambda_k(z-z_0)}e^{i\lambda_k|(x-x_0)\cos\alpha_{j,k}+(y-y_0)\sin\alpha_{j,k}|}\right| < \epsilon \tag{6.68}$$

with the integers $s(\epsilon)$ and triplets $\{M_k, \omega_k, \lambda_k,\ k = 1, 2, ...s(\epsilon)\}$ all depend on ϵ, and $\alpha_{j,k} = \frac{2\pi j}{M_k}$. Now the rest process are very similar to the two dimensional

process. It is clear now potential can be represented as exponential expansions and their translations are diagonal. Theorems are provided that multipole expansions can be converted into exponential expansions and exponential expansions into local expansions. Combining these with the fact that an interaction list can be partitioned into six directional lists, each of which can be processed at once, a fast multipole to local translation operator results.

Chapter 7

Fast Rotation Transform

7.1 Introduction

From the previous chapter, we know that translation operators can be decomposed as the product of coaxial translations and rotations. We will develop a fast algorithm for coaxial translations in the next chapter. In this chapter we develop a fast algorithm for the rotation operation, which we call the fast rotation transform.

Such fast transforms are also important in other scientific disciplines, such as quantum mechanics, geoscience, and computer vision, where there is increased interest in high-degree spherical harmonic expansions. A lot of literature is devoted to computing the matrix elements fast using recurrence relations [Gumerov2001, Risbo96, Choi99]. The computations involved are extremely costly. This fast rotation transform will greatly improve the situations.

Driscoll and Healy [Driscoll94] developed a fast algorithm to compute spherical harmonic ("Fourier") transforms and convolutions on the 2-sphere, which is related to this. Their algorithm can be used to produce the coefficients of spherical harmonic expansion of a function by sampling points on sphere. Our

algorithm can be used to compute the coefficients of a multipole/local expansion on rotation transform.

To make this chapter independent, first, we repeat necessary information about the rotation group $SO(3)$ and spherical harmonics. After this, we present results on decomposing the rotational matrix that allow fast matrix-vector product. Then we discuss the complexity of the fast rotation transform.

We define a rotation about a given axis to be the one that carries a right-handed screw in the positive direction along that axis. As mentioned before, the Euler angles are the most convenient way to parameterize the rotation group $SO(3)$. Given any two sets of right-handed frame of axes, S_1 and S_2, sharing a common origin, we can always find the Euler angles $(\alpha\beta\gamma)$ so that we can rotate frame S_1 into frame S_2 as follows,

1. A rotation α $(0 \leq \alpha < 2\pi)$ about $n_1 = (0, 0, 1)$, written as $R_z(\alpha)$.

2. A rotation β $(0 \leq \beta \leq \pi)$ about $n_2 = (-\sin\alpha, \cos\alpha, 0)$, written as $R_{y'}(\beta)$.

3. A rotation γ $(0 \leq \gamma < 2\pi)$ about $n_3 = (\cos\alpha\sin\beta, \sin\alpha\sin\beta, \cos\beta)$ written as $R_{z''}(\gamma)$.

The effect of such rotations of the frame of reference is denoted by operator $R(\alpha\beta\gamma) = R_{z''}(\gamma)R_{y'}(\beta)R_z(\alpha)$. Let us consider a function $f : R^3 \to C$ under frame S_1, or $y = f(r, \theta, \phi)$, where $y \in C$. After rotation $R(\alpha\beta\gamma)$, we would have $f(r, \theta', \phi')$. This can be expressed formally by

$$R(\alpha\beta\gamma)f(r, \theta, \phi) = f(r, \theta', \phi'). \tag{7.1}$$

For example, the effect of such rotations of the frame of reference upon the spherical harmonics $Y_n^m(\theta, \phi)$ can be expressed as the following equation:

$$R(\alpha\beta\gamma)Y_n^m(\theta, \phi) = Y_n^m(\hat{\theta}, \hat{\phi}). \tag{7.2}$$

It is easy to see that $R(\beta 00)Y_n^m(\theta,\phi) = Y_n^m(\theta,\phi-\beta)$ since a rotation about z axis only changes the angle ϕ, not θ. Therefore, when a rotation about z-axis is performed, the resulting vector consisting of the coefficients of the new multipole or local expansion can be obtained by multiplying a diagonal matrix $D(\beta 00)$ to the vector of the coefficients of the original expansion, where $D(\alpha\beta\gamma)$ is the matrix corresponding to the rotation $R(\alpha\beta\gamma)$. The following lemma is needed to reduce the representation of the group $SO(3)$ on a linear space $L^2(S^2)$ into a sum of irreducible representations, where $L^2(S^2)$ is a linear space spanned by an orthonormal basis consisting spherical harmonics

Lemma 7.1. *Under any rotation, each spherical harmonic of degree n is transformed into a linear combination of the spherical harmonics of the same degree n, that is,*

$$R(\alpha\beta\gamma)Y_n^m(\theta,\phi) = \sum_{l=-n}^{n} Y_n^l(\theta,\phi). \tag{7.3}$$

The $(2n+1)$ spherical harmonics Y_n^l, $-n \le l \le n$ form a invariant space under rotation.

A proof can be found in [Edmonds60].

The matrix $D(\alpha\beta\gamma)$ acting on j-th degree of spherical harmonics is denoted by $D^{(j)}(\alpha\beta\gamma)$, or $D^{(j)}$. We shall also write

$$D_{m'm}^{(j)}(0\beta 0) = d_{m'm}^{(j)}(\beta). \tag{7.4}$$

Combining all these facts, we have

$$D_{m'm}^{(j)}(\alpha\beta\gamma) = e^{im'\gamma} d_{m'm}^{(j)}(\beta) e^{im\alpha}. \tag{7.5}$$

7.2 Decomposition of the rotational matrix

It is clear from (7.5) that in order to reduce the order of the complexity of the rotation matrix we need only treat $D(0\beta 0)$. The entries of this matrix are given by the following formula [Edmonds60]:

$$d^{(j)}_{m'm}(\beta) = \left[\frac{(j+m')!(j-m')!}{(j+m)!(j-m)!}\right]^{\frac{1}{2}} * \tag{7.6}$$

$$\sum_{\sigma} C^{j-m'-\sigma}_{j+m} C^{\sigma}_{j-m} (-1)^{j-m'-\sigma} (\cos\frac{\beta}{2})^{2\sigma+m'+m} (\sin\frac{\beta}{2})^{2j-2\sigma-m'-m} \tag{7.7}$$

A similarity transformation may be employed to express a $D(0\beta 0)$ in terms of a rotation about the z-axis, i.e. a $D(\beta 00)$, which is diagonal (see [Edmonds60]):

$$D(0\beta 0) = D(-\frac{\pi}{2}00)D(0-\frac{\pi}{2}0)D(\beta 00)D(0\frac{\pi}{2}0)D(\frac{\pi}{2}00). \tag{7.8}$$

Hence we need only treat $D(0\frac{\pi}{2}0)$ and $D(0-\frac{\pi}{2}0)$, which are constant matrices. It is easy to see that

$$D(0-\frac{\pi}{2}0) = D(0\frac{\pi}{2}0)'. \tag{7.9}$$

Let us consider the case when $\beta = \frac{\pi}{2}$. From (7.6), the entries of this rotation matrix can be simplified as the following,

$$d^{(j)}_{m'm} = \left(\frac{1}{2}\right)^{j} \left[\frac{(j+m')!(j-m')!}{(j+m)!(j-m)!}\right]^{\frac{1}{2}} \sum_{\sigma} C^{j-m'-\sigma}_{j+m} C^{\sigma}_{j-m} (-1)^{j-m'-\sigma}. \tag{7.10}$$

7.2.1 Decomposition 1

Theorem 7.2. *This matrix can be factored as the following,*

$$D^{(j)}(0\frac{\pi}{2}0) = diag(v_1) * diag(v_2) * rot180(P) * diag(v_3) * diag(v_2) * P * diag(v_4), \tag{7.11}$$

where $v_1(m) = [(j+m)!(j-m)!]^{-\frac{1}{2}}$, $v_2(m) = (-1)^{j+m}$, $v_3(m) = [2^{-m}]$, $v_4(m) = [(j+m)!(j-m)!]^{\frac{1}{2}}$, $m = -j, \cdots, 0, \cdots j$, P *is Pascal matrix*

$$P = \begin{bmatrix} C_0^0 & 0 & 0 & ... & 0 \\ C_1^0 & C_1^1 & 0 & ... & 0 \\ C_2^0 & C_2^1 & C_2^2 & ... & 0 \\ ... & ... & ... & ... & ... \\ C_{2j}^0 & C_{2j}^1 & C_{2j}^2 & ... & C_{2j}^{2j} \end{bmatrix}, \tag{7.12}$$

and $rot180(P)$ *is a matrix resulted from rotating Pascal matrix* 180 *degree,*

$$rot180(P) = \begin{bmatrix} C_{2j}^{2j} & ... & C_{2j}^2 & C_{2j}^1 & C_{2j}^0 \\ ... & ... & ... & ... & ... \\ 0 & ... & C_2^2 & C_2^1 & C_2^0 \\ 0 & ... & 0 & C_1^1 & C_1^0 \\ 0 & ... & 0 & 0 & C_0^0 \end{bmatrix}. \tag{7.13}$$

Proof. Let us verify this identity.

First from (7.10) it is easy to show that

$$(-1)^{m'} d_{m'm}^{(j)} \left(\frac{(j+m')!(j-m')!}{(j+m)!(j-m)!} \right)^{1/2} = (-1)^m d_{mm'}^{(j)} \left(\frac{(j+m')!(j-m')!}{(j+m)!(j-m)!} \right)^{1/2}. \tag{7.14}$$

For the pair (m', m), we know $d_{m'm}^{(j)}$. We need only to compute it on the right side. We compute the corresponding entry, which is denoted by $a_{m'm}$, of the matrix

$$2^j * rot180(P) * diag(v_3) * diag(v_2) * P, \tag{7.15}$$

and then multiply out the rest of diagonal matrices. The entries in the row corresponding to m' in the matrix $rot180(P)$ are

$$RP_{m't} = \begin{cases} 0 & \text{if } t < m' \\ C_{j-m'}^{j-t} & \text{if } t \geq m' \end{cases}. \tag{7.16}$$

The entries in the column corresponding to m in matrix P are

$$P_{tm} = \begin{cases} 0 & \text{if } t < m \\ C_{j+t}^{j+m} & \text{if } t \geq m \end{cases}. \tag{7.17}$$

We have

$$a_{m'm} = \sum_{j \geq t \geq \max(m,m')} (-2)^{j-t} C_{j-m'}^{j-t} C_{j+t}^{j+m}, \tag{7.18}$$

which is the coefficient of x^{j+m} in $(1+x)^{j+m'}(1-x)^{j-m'}$ and is the same as

$$2^j \left(\frac{(j+m')!(j-m')!}{(j+m)!(j-m)!} \right)^{1/2} d_{m'm}^{(j)}. \tag{7.19}$$

Therefore, we have

$$d_{m'm}^{(j)} = 2^{-j} \left(\frac{(j+m')!(j-m')!}{(j+m)!(j-m)!} \right)^{-1/2} a_{m'm} \tag{7.20}$$

which completes the proof. ■

Remark: In this decomposition, we have factored the rotation matrix into products of diagonal matrices, a Pascal matrix, and the rotation of Pascal matrix. Therefore from Chapter 3, we know that for each j, the matrix-vector product $D^{(j)}(0\frac{\pi}{2}0)x$ for any vector x, costs $O((2j+1)\log(2j+1))$ operations. We will base our implementation of the fast rotation transform on this decomposition of the rotation matrix.

7.2.2 Decomposition 2

It can be easily verified that the following is also true.

Theorem 7.3. *This matrix can also be factored as the following,*

$$D = (\frac{1}{2})^j * diag(v_1) * V_1 * diag(v_2) * V_2^{-1} * diag(v_3), \tag{7.21}$$

where $v_1(m) = [(j+m)!(j-m)!]^{-\frac{1}{2}}$, $v_2(m) = [(1-x_m)^{2j}]$, $v_3(m) = [(j+m)!(j-m)!]^{\frac{1}{2}}$,

$$V_1 = \begin{bmatrix} 1 & 1 & ... & 1 \\ \frac{1+x_1}{1-x1} & \frac{1+x_2}{1-x_2} & ... & \frac{1+x_n}{1-x_n} \\ ... & ... & ... & ... \\ \left(\frac{1+x_1}{1-x_1}\right)^{n-1} & \left(\frac{1+x_2}{1-x2}\right)^{n-1} & ... & \left(\frac{1+x_n}{1-x_n}\right)^{n-1} \end{bmatrix}, \tag{7.22}$$

and

$$V_2 = \begin{bmatrix} 1 & 1 & ... & 1 \\ x_1 & x_2 & ... & x_n \\ ... & ... & ... & ... \\ x_1^{n-1} & x_2^{n-1} & ... & x_n^{n-1} \end{bmatrix}, \tag{7.23}$$

and $\{x_i, i = 1, 2, \cdots, p\}$ *are distinct numbers.*

Proof. From the proof of the previous theorem, it is easy to see that

$$diag(v_1)^{-1} 2^j D * diag(v_3)^{-1} V_2 = V_1 * diag(v_2). \tag{7.24}$$

Notice that $\{x_i, i = 1, 2, \cdots, p\}$ are distinct numbers, and hence V_2 is invertible. Thus we have our desired decomposition. ■

Since V_1 and V_2 are Vandermonde matrices, and the product of any Vandermonde matrix of order $n \times n$ and any vector of order n can be done in time complexity $O(n \log^2 n)$ as mentioned before. Even though this factorization also admits a fast matrix-vector product, we are not going to implement it due to the speed and stability issues of algorithms for fast multiplication of a Vandermonde matrix and a vector.

7.3 A fast rotation algorithm

In this section we present a fast rotation algorithm based on equations (7.5), (7.8), and (7.11) and leave the implementation and stability issues to Chapter 9. Given any two right-hand screw coordinate systems S_1 and S_2 that share a common origin, and a finite spherical harmonic expansion of a band-limited function $f(\theta, \phi)$,

$$f(\theta, \phi) = \sum_{n=0}^{p-1} \sum_{m=-n}^{n} a_{nm} Y_n^m(\theta, \phi) \tag{7.25}$$

in a coordinate system S_1. We can compute the coefficients

$$\{b_{nm,}\ n = 0, 1, \cdots, p-1,\ m = -n. \cdots, 0, \cdots, n\} \tag{7.26}$$

of the spherical harmonic expansion of the function $f(\theta', \phi')$ in S_2 from

$$\{a_{nm,}\ n = 0, 1, \cdots, p-1,\ m = -n. \cdots, 0, \cdots, n\} \tag{7.27}$$

as follows. Suppose that the rotation required from S_1 to S_2 is $R(\alpha\beta\gamma)$.

Algorithm 7.4. *for* $j = 0, 1, 2, \cdots, p-1$,

1. *Compute* $x_{j*}^{(1)} = D^{(j)}(00(\alpha + \frac{\pi}{2}))a_{j*}$, *where* a_{j*} *is the vector of the* $(2j+1)$ *coefficients of the* j*-degree harmonic expansion*
2. *Compute* $x_{j*}^{(2)} = D^{(j)}(0\frac{\pi}{2}0)x_{j*}^{(1)}$, *using equation (7.11).*
3. *Compute* $x_{j*}^{(3)} = D^{(j)}(00\beta)x_{j*}^{(2)}$.
4. *Compute* $x_{j*}^{(4)} = D^{(j)}(0-\frac{\pi}{2}0)x_{j*}^{(3)}$, *using the transpose of the equation (7.11).*
5. *Compute* $b_{j*} = D^{(j)}(00(\gamma - \frac{\pi}{2}))x_{j*}^{(4)}$.

end

7.4 Complexity

We analyze the complexity of the matrix-vector product for the rotational matrix for the first decomposition. Since both a Pascal matrix and its rotation can be factored into products of two diagonal matrices and one Toeplitz matrix. Each Toeplitz matrix of order n requires $O(10n\log(2n))$ flops besides a precomputed FFT. Therefore, for each j, the matrix-vector product $D^{(j)}(0\frac{\pi}{2}0)x^{(j)}$ for any vector $x^{(j)}$, costs $O(40(2j+1)\log(4j+2))$ flops. Thus, for the matrix-vector product $D(0\frac{\pi}{2}0)x$ for any vector x, it costs approximately

$$\sum_{j=0}^{p-1} O(40(2j+1)\log(4j+2)) \approx O(40p^2\log(4p-2)). \tag{7.28}$$

For the whole rotation, it costs approximately $80p^2\log(4p-2)$.

Chapter 8

Efficient Translation Operators in Three Dimensions

Now we know how to represent each of these three translation matrices as a product of a rotation matrix, a coaxial translation matrix and another rotation matrix. In this chapter we present a new matrix factorization based on these representations and show how to further reduce number of operations. We decompose the coaxial translation matrix to improve the efficiency of a product with it from $O(p^3)$ to $O(p^2 \log p)$. Recalling that we have developed algorithms in previous chapter to reduce the complexity of a rotation operation from $O(p^3)$ to $O(p^2 \log p)$, we obtain a scheme to reduce the cost of 3-D translation operators from $O(p^4)$ to $O(p^2 \log p)$.

8.1 Factorization of the coaxial translation matrices

Let us consider the translation operator when the translation is done along the z-axis, that is, we consider the coaxial translations $SS(\rho)$, $RR(\rho)$, $SR(\rho)$.

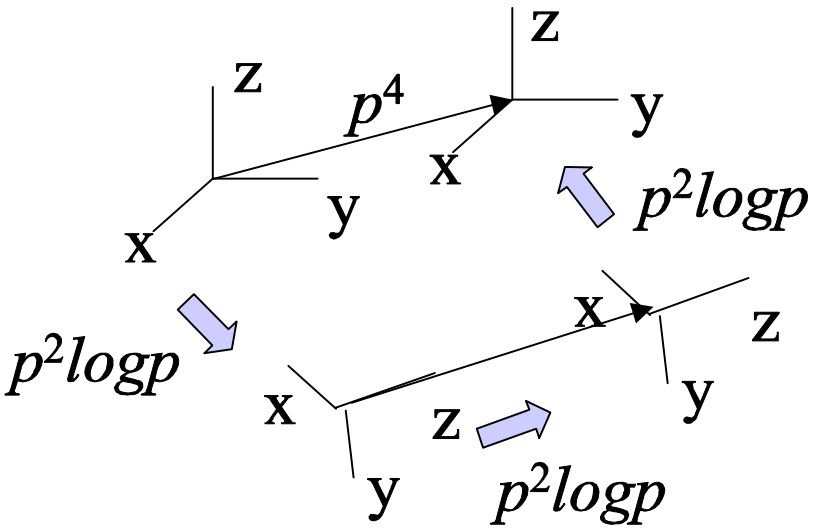

Figure 8.1: An efficient rotation based translation

8.1.1 Multipole translation

When multipole translation is done along the z-axis, the equation (6.26) becomes simpler,

$$M_j^k = \sum_{n=0}^{j-|k|} \frac{O_{j-n}^k A_{j-n}^k A_n^0 \rho^n}{A_j^k}. \tag{8.1}$$

The matrix format is for $k = -p, -p+1, ..., -1, 0, 1, ..., p$,

$$M^{(k)} = SS^{(k)}(\rho) * O^{(k)} \tag{8.2}$$

where

$$M^{(k)} = [\ M_{|k|}^k \quad M_{|k|+1}^k \quad M_{|k|+2}^k \quad ... \quad M_p^k\ ]', \tag{8.3}$$

$$O^{(k)} = [\ O_{|k|}^k \quad O_{|k|+1}^k \quad O_{|k|+2}^k \quad ... \quad O_p^k\ ]', \tag{8.4}$$

and

$$SS^{(k)}(\rho)=\begin{bmatrix} \frac{A_0^0A_{|k|}^k}{A_{|k|}^k} & 0 & 0 & ... & 0 \\ \frac{A_1^0A_{|k|}^k}{A_{|k|+1}^k}\rho & \frac{A_0^0A_{|k|+1}^k}{A_{|k|+1}^k} & 0 & ... & 0 \\ \frac{A_2^0A_{|k|}^k}{A_{|k|+2}^k}\rho^2 & \frac{A_1^0A_{|k|+1}^k}{A_{|k|+2}^k}\rho & \frac{A_0^0A_{|k|+2}^k}{A_{|k|+2}^k} & ... & 0 \\ ... & ... & ... & ... & ... \\ \frac{A_{p-|k|}^0A_{|k|}^k}{A_p^k}\rho^{p-|k|} & \frac{A_{p-|k|-1}^0A_{|k|+1}^k}{A_p^k}\rho^{p-|k|-1} & \frac{A_{p-|k|-2}^0A_{p-2}^k}{A_p^k}\rho^{p-|k|-2} & ... & \frac{A_0^0A_p^k}{A_p^k} \end{bmatrix}. \tag{8.5}$$

This matrix can be factorized as in the following lemma.

Lemma 8.1.

$$SS^{(k)}(\rho)=diag\begin{bmatrix} 1 \\ \rho \\ \rho^2 \\ ... \\ \rho^{p-|k|} \end{bmatrix} * SS^{(k)}(1) * diag\begin{bmatrix} 1 \\ \frac{1}{\rho} \\ \frac{1}{\rho^2} \\ ... \\ \frac{1}{\rho^{p-|k|}} \end{bmatrix}, \tag{8.6}$$

where

$$SS^{(k)}(1)=\begin{bmatrix} \frac{A_0^0A_{|k|}^k}{A_{|k|}^k} & 0 & 0 & ... & 0 \\ \frac{A_1^0A_{|k|}^k}{A_{|k|+1}^k} & \frac{A_0^0A_{|k|+1}^k}{A_{|k|+1}^k} & 0 & ... & 0 \\ \frac{A_2^0A_{|k|}^k}{A_{|k|+2}^k} & \frac{A_1^0A_{|k|+1}^k}{A_{|k|+2}^k} & \frac{A_0^0A_{|k|+2}^k}{A_{|k|+2}^k} & ... & 0 \\ ... & ... & ... & ... & ... \\ \frac{A_{p-|k|}^0A_{|k|}^k}{A_p^k} & \frac{A_{p-|k|-1}^0A_{|k|+1}^k}{A_p^k} & \frac{A_{p-|k|-2}^0A_{p-2}^k}{A_p^k} & ... & \frac{A_0^0A_p^k}{A_p^k} \end{bmatrix} \tag{8.7}$$

is a constant matrix.

Proof. Notice that for column $i,\ i=0,1,\cdots,p-|k|$, of matrix $SS^{(k)}(\rho)$, there is a common factor $\frac{1}{\rho^i}$; and for row $j,\ j=0,1,\cdots,p-|k|$, of matrix $SS^{(k)}(\rho)$, there is a common factor ρ^j. Therefore we have this factorization. ■

It can be further factored as in the following lemma.

Lemma 8.2.

$$SS^{(k)}(\rho) = diag\begin{bmatrix} 1/A^k_{|k|} \\ \rho/A^k_{|k|+1} \\ \rho^2/A^k_{|k|+2} \\ ... \\ \rho^{p-|k|}/A^k_p \end{bmatrix} * MM * diag\begin{bmatrix} A^k_{|k|} \\ A^k_{|k|+1}/\rho \\ A^k_{|k|+2}/\rho^2 \\ ... \\ A^k_p/\rho^{p-|k|} \end{bmatrix}, \tag{8.8}$$

where

$$MM = \begin{bmatrix} A^0_0 & 0 & 0 & ... & 0 \\ A^0_1 & A^0_0 & 0 & ... & 0 \\ A^0_2 & A^0_1 & A^0_0 & ... & 0 \\ ... & ... & ... & ... & ... \\ A^0_{p-|k|} & A^0_{p-|k|-1} & A^0_{p-|k|-2} & ... & A^0_0 \end{bmatrix}, \tag{8.9}$$

is a Toeplitz matrix, which admits fast matrix vector multiplication.

Proof. Notice that for column $i,\ i = 0, 1, \cdots, p-|k|$, of matrix $SS^{(k)}(1)$, there is a common factor $A^k_{|k|+i}$; and for row $j,\ j = 0, 1, \cdots, p-|k|$, of matrix $SS^{(k)}(1)$, there is a common factor $\frac{1}{A^k_{|k|+j}}$. We factor $SS^{(k)}(1)$ as the products of three matrices, matrix MM sandwiched by two diagonal matrices. Then we multiply out the diagonal matrices on each side to arrive the factorization in this lemma.

■

Theorem 8.3. *The truncated matrix vector product (8.2) for each k can be done in time $O((p-|k|)\log(p-|k|))$. The whole set of coefficients M can be obtained in $O(p^2 \log p)$ time.*

Proof. From the previous lemma, to multiply matrix $SS^{(k)}(\rho)$ with any vector requires three matrix-vector products, which involve two diagonal matrices and

one Toeplitz matrix. We know the matrix-vector product for a Toeplitz matrix requires $O((p-|k|)\log(p-|k|))$ operations, while the product for a diagonal matrix requires $O(p-|k|)$ operations. Therefore the matrix-vector product for each $SS^{(k)}(\rho)$ can be done in $O((p-|k|)\log(p-|k|))$ operations. Thus the whole set of coefficients M can be obtained in

$$\sum_{k=-p}^{p} O((p-|k|)\log(p-|k|)) = O(p^2 \log p) \tag{8.10}$$

time. ■

8.1.2 Local translation

When local translation is done along $z-axis$, the equation (6.34) becomes simpler,

$$L_j^k = \sum_{n=j}^{p} \frac{O_n^k A_j^k A_{n-j}^0 \rho^{n-j}}{(-1)^{n+j} A_n^k}. \tag{8.11}$$

The matrix format is for $k = -p, -p+1, ..., -1, 0, 1, ..., p$,

$$L^{(k)} = RR^{(k)}(\rho) * O^{(k)} \tag{8.12}$$

where

$$L^{(k)} = [\ L_{|k|}^k \quad L_{|k|+1}^k \quad L_{|k|+2}^k \quad ... \quad L_p^k\ ]', \tag{8.13}$$

$$O^{(k)} = [\ O_{|k|}^k \quad O_{|k|+1}^k \quad O_{|k|+2}^k \quad ... \quad O_p^k\ ]', \tag{8.14}$$

and

$$RR^{(k)}(\rho) = \begin{bmatrix} \frac{A_0^0 A_{|k|}^k}{A_{|k|}^k} & -\frac{A_1^0 A_{|k|}^k}{A_{|k|+1}^k}\rho & \frac{A_2^0 A_{|k|}^k}{A_{|k|+2}^k}\rho^2 & ... & \frac{A_{p-|k|}^0 A_{|k|}^k}{(-1)^{p+|k|} A_p^k}\rho^{p-|k|} \\ 0 & \frac{A_0^0 A_{|k|+1}^k}{A_{|k|+1}^k} & -\frac{A_1^0 A_{|k|+1}^k}{A_{|k|+2}^k}\rho & ... & \frac{A_{p-|k|-1}^0 A_{|k|+1}^k}{(-1)^{p+|k|+1} A_p^k}\rho^{p-|k|-1} \\ 0 & 0 & \frac{A_0^0 A_{|k|+2}^k}{A_{|k|+2}^k} & ... & \frac{A_{p-|k|-2}^0 A_{p-2}^k}{(-1)^{p+|k|+2} A_p^k}\rho^{p-|k|-2} \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & \frac{A_0^0 A_p^k}{A_p^k} \end{bmatrix}. \tag{8.15}$$

This matrix can be similarly factored as in the following lemma.

Lemma 8.4.

$$RR^{(k)}(\rho) = diag\begin{bmatrix} 1 \\ \frac{1}{\rho} \\ \frac{1}{\rho^2} \\ \cdots \\ \frac{1}{\rho^{p-|k|}} \end{bmatrix} * RR^{(k)}(1) * diag\begin{bmatrix} 1 \\ \rho \\ \rho^2 \\ \cdots \\ \rho^{p-|k|} \end{bmatrix} \tag{8.16}$$

where

$$\begin{bmatrix} \frac{A_0^0 A_{|k|}^k}{A_{|k|}^k} & -\frac{A_1^0 A_{|k|}^k}{A_{|k|+1}^k} & \frac{A_2^0 A_{|k|}^k}{A_{|k|+2}^k} & \cdots & \frac{A_{p-|k|}^0 A_{|k|}^k}{(-1)^{p+|k|} A_p^k} \\ 0 & \frac{A_0^0 A_{|k|+1}^k}{A_{|k|+1}^k} & -\frac{A_1^0 A_{|k|+1}^k}{A_{|k|+2}^k} & \cdots & \frac{A_{p-|k|-1}^0 A_{|k|+1}^k}{(-1)^{p+|k|+1} A_p^k} \\ 0 & 0 & \frac{A_0^0 A_{|k|+2}^k}{A_{|k|+2}^k} & \cdots & \frac{A_{p-|k|-2}^0 A_{p-2}^k}{(-1)^{p+|k|+2} A_p^k} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \frac{A_0^0 A_p^k}{A_p^k} \end{bmatrix} \tag{8.17}$$

is a constant matrix.

This lemma can be proved in the same way as that of the corresponding lemma for the multipole translation. We omit it here.

It can be further factored as in the following lemma.

Lemma 8.5.

$$RR^{(k)}(\rho) = diag\begin{bmatrix} A_{|k|}^k \\ -A_{|k|+1}^k/\rho \\ A_{|k|+2}^k/\rho^2 \\ \cdots \\ A_p^k/(-\rho)^{p-|k|} \\ \cdots \end{bmatrix} * LL * diag\begin{bmatrix} 1/A_{|k|}^k \\ -\rho/A_{|k|+1}^k \\ \rho^2/A_{|k|+2}^k \\ \cdots \\ (-\rho)^{p-|k|}/A_p^k \\ \cdots \end{bmatrix}, \tag{8.18}$$

where

$$LL = \begin{bmatrix} A_0^0 & A_1^0 & A_2^0 & ... & A_{p-|k|}^0 \\ 0 & A_0^0 & A_1^0 & ... & A_{p-|k|-1}^0 \\ 0 & 0 & A_0^0 & ... & A_{p-|k|-2}^0 \\ ... & ... & ... & ... & ... \\ 0 & 0 & 0 & ... & A_0^0 \end{bmatrix}, \tag{8.19}$$

is again a Toeplitz matrix.

This lemma can be proved in the same way as that of the corresponding lemma for the multipole translation. We omit it here.

Theorem 8.6. *The truncated matrix vector product (8.12) for each k can be done in time $O((p-|k|)\log(p-|k|))$. The whole set of coefficients L can be obtained in time $O(p^2 \log p)$.*

This theorem can be proved in the same way as that of the corresponding theorem for the multipole translation. We omit it here.

8.1.3 Multipole to local translation

When multipole to local translation is done along the z-axis, the equation (6.40) becomes simpler,

$$L_j^k = \sum_{n=|k|}^{p} \frac{O_n^k A_n^k A_j^k}{(-1)^{n+k} A_{j+n}^0 \rho^{n+j+1}}. \tag{8.20}$$

The matrix format is for $k = -p, -p+1, ..., -1, 0, 1, ..., p$,

$$L^{(k)} = SR^{(k)}(\rho) * O^{(k)} \tag{8.21}$$

where

$$L^{(k)} = [\ L_{|k|}^k \quad L_{|k|+1}^k \quad L_{|k|+2}^k \quad ... \quad L_p^k\ ]', \tag{8.22}$$

$$O^{(k)} = [\ O^k_{|k|} \quad O^k_{|k|+1} \quad O^k_{|k|+2} \quad ... \quad O^k_p\ ]', \tag{8.23}$$

and

$$SR^{(k)}(\rho) = \begin{bmatrix} \frac{A^k_{|k|}A^k_{|k|}}{(-1)^{2|k|}A^0_{2|k|}\rho^{2|k|+1}} & \frac{A^k_{|k|+1}A^k_{|k|}}{(-1)^{2|k|+1}A^0_{2|k|+1}\rho^{2|k|+2}} & \cdots & \frac{A^k_pA^k_{|k|}}{(-1)^{p+k}A^0_{|k|+p}\rho^{p+|k|+1}} \\ \frac{A^k_{|k|}A^k_{|k|+1}}{(-1)^{2|k|}A^0_{2|k|+1}\rho^{2|k|+2}} & \frac{A^k_{|k|+1}A^k_{|k|+1}}{(-1)^{2|k|+1}A^0_{2|k|+2}\rho^{2|k|+3}} & \cdots & \frac{A^k_pA^k_{|k|+1}}{(-1)^{p+k}A^0_{|k|+p+1}\rho^{p+|k|+2}} \\ \frac{A^k_{|k|}A^k_{|k|+2}}{(-1)^{2|k|}A^0_{2|k|+2}\rho^{2|k|+3}} & \frac{A^k_{|k|+1}A^k_{|k|+2}}{(-1)^{2|k|+1}A^0_{2|k|+3}\rho^{2|k|+4}} & \cdots & \frac{A^k_pA^k_{|k|+2}}{(-1)^{p+k}A^0_{|k|+p+2}\rho^{p+|k|+3}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{A^k_{|k|}A^k_p}{(-1)^{2|k|}A^0_{|k|+p}\rho^{|k|+p+1}} & \frac{A^k_{|k|+1}A^k_p}{(-1)^{2|k|+1}A^0_{|k|+p+1}\rho^{p+|k|+2}} & \cdots & \frac{A^k_pA^k_p}{(-1)^{p+k}A^0_{2p}\rho^{2p+1}} \end{bmatrix}. \tag{8.24}$$

We will use the same trick of the multipole translation on the multipole to local translation. All lemmas and theorem can be proved similarly. We only list the results and omit their proofs.

The matrix $SR^{(k)}(\rho)$ can be factored as in the following lemma.

Lemma 8.7.

$$SR^{(k)}(\rho) = (-1)^k * diag \begin{bmatrix} \rho^{-|k|-1} \\ \rho^{-|k|-2} \\ \rho^{-|k|-3} \\ \cdots \\ \rho^{-p-1} \end{bmatrix} * SR^{(k)}(1) * diag \begin{bmatrix} \rho^{-|k|} \\ \rho^{-|k|-1} \\ \rho^{-|k|-2} \\ \cdots \\ \rho^{-p} \end{bmatrix} \tag{8.25}$$

where

$$SR^{(k)}(1) = \begin{bmatrix} \frac{A^k_{|k|}A^k_{|k|}}{(-1)^{|k|}A^0_{2|k|}} & \frac{A^k_{|k|+1}A^k_{|k|}}{(-1)^{|k|+1}A^0_{2|k|+1}} & \frac{A^k_{|k|+2}A^k_{|k|}}{(-1)^{|k|+2}A^0_{2|k|+2}} & \cdots & \frac{A^k_pA^k_{|k|}}{(-1)^pA^0_{|k|+p}} \\ \frac{A^k_{|k|}A^k_{|k|+1}}{(-1)^{|k|}A^0_{2|k|+1}} & \frac{A^k_{|k|+1}A^k_{|k|+1}}{(-1)^{|k|+1}A^0_{2|k|+2}} & \frac{A^k_{|k|+2}A^k_{|k|+1}}{(-1)^{|k|+2}A^0_{2|k|+3}} & \cdots & \frac{A^k_pA^k_{|k|+1}}{(-1)^pA^0_{|k|+p+1}} \\ \frac{A^k_{|k|}A^k_{|k|+2}}{(-1)^{|k|}A^0_{2|k|+2}} & \frac{A^k_{|k|+1}A^k_{|k|+2}}{(-1)^{|k|+1}A^0_{2|k|+3}} & \frac{A^k_{|k|+2}A^k_{|k|+2}}{(-1)^{|k|+2}A^0_{2|k|+4}} & \cdots & \frac{A^k_pA^k_{|k|+2}}{(-1)^pA^0_{|k|+p+2}} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{A^k_{|k|}A^k_p}{(-1)^{|k|}A^0_{|k|+p}} & \frac{A^k_{|k|+1}A^k_p}{(-1)^{|k|+1}A^0_{|k|+p+1}} & \frac{A^k_{|k|+2}A^k_p}{(-1)^{|k+2|}A^0_{|k|+p+2}} & \cdots & \frac{A^k_pA^k_p}{(-1)^pA^0_{2p}} \end{bmatrix} \tag{8.26}$$

is a constant matrix.

It can be further factored as in the following lemma.

Lemma 8.8.

$$SR^{(k)}(\rho) = (-1)^k * diag \begin{bmatrix} A^k_{|k|}\rho^{-|k|-1} \\ A^k_{|k|+1}\rho^{-|k|-2} \\ A^k_{|k|+2}\rho^{-|k|-3} \\ \cdots \\ A^k_p\rho^{-p-1} \end{bmatrix} * ML * diag \begin{bmatrix} A^k_{|k|}(-\rho)^{-|k|} \\ A^k_{|k|+1}(-\rho)^{-|k|-1} \\ A^k_{|k|+2}(-\rho)^{-|k|-2} \\ \cdots \\ A^k_p(-\rho)^{-p} \end{bmatrix}, \tag{8.27}$$

where

$$ML = \begin{bmatrix} \frac{1}{A^0_{2|k|}} & \frac{1}{A^0_{2|k|+1}} & \frac{1}{A^0_{2|k|+2}} & \cdots & \frac{1}{A^0_{|k|+p}} \\ \frac{1}{A^0_{2|k|+1}} & \frac{1}{A^0_{2|k|+2}} & \frac{1}{A^0_{2|k|+3}} & \cdots & \frac{1}{A^0_{|k|+p+1}} \\ \frac{1}{A^0_{2|k|+2}} & \frac{1}{A^0_{2|k|+3}} & \frac{1}{A^0_{2|k|+4}} & \cdots & \frac{1}{A^0_{|k|+p+2}} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \frac{1}{A^0_{|k|+p}} & \frac{1}{A^0_{|k|+p+1}} & \frac{1}{A^0_{|k|+p+2}} & \cdots & \frac{1}{A^0_{2p}} \end{bmatrix}, \tag{8.28}$$

is a Hankel matrix.

Theorem 8.9. *The truncated matrix vector product (8.21) for each k can be done in time $O((p-|k|)\log(p-|k|))$. The whole set of coefficients L can be obtained in time*

$$\sum_{k=-p}^{p} O((p-|k|)\log(p-|k|)), \tag{8.29}$$

which is $O(p^2 \log p)$.

The coaxial translation operators are factored into product of diagonal matrices and Toeplitz/Hankel matrices. The translation operators could be unstable if it is implemented directly based these factorizations. We will leave the instability and implementation issues to next chapter.

8.2 Complexity analysis

From Theorems 6.2, 6.3, and 6.4, we know that order of all three translation operators is of the order of $O(p^4)$. A direct implementation of matrix-vector products with these operators requires roughly $8p^4$ flops. Our efficient implementation of the same operations based on matrix decomposition involves fast Fourier transform which reduces number of operations in matrix-vector product to roughly $170p^2 \log 4p$ flops. That is $O(p^4)$ operations in the original algorithm is reduced to approximately $\frac{85}{4}p^2 \log(4p)$ operations in the new algorithm. Let us see how this would affect the total complexity.

We know the total cost of the original algorithm for all five steps is approximately

$$2Np^2 + \frac{1528}{7}\frac{N}{s}p^4 + 27Ns. \tag{8.30}$$

With $s \approx \sqrt{\frac{1528}{189}}p^2$, the total number of operations is approximately $156Np^2$. In our current new algorithm, the total cost would be

$$2Np^2 + \frac{1528}{7}\frac{N}{s} * \frac{85}{4}p^2 \log(4p) + 9Ns. \tag{8.31}$$

With $s \approx \frac{\left(228480p^2 \log(4p)\right)^{0.5}}{21}$, the improved count is approximately

$$2Np^2 + 410\sqrt{\log(4p)}Np. \tag{8.32}$$

According to this result, the break even p is 5. In general, the new translation operators should significantly speed up the computations. If the same accuracy is required, the parameter p should be bigger in the three-dimensional case than that in the two-dimensional case. It implies that the new operators are indispensable for three dimensions. We would like to note as in the two-dimensional case that

choosing a proper parameter s, which determines number of levels in the data structure, has a major effect over the efficiency of the program.

Chapter 9

Stability Issues and Implementation

We have developed efficient algorithms for the product of the $p \times p$ Pascal matrix or its related matrices and a vector in Chapter 3. Based on these algorithms in Chapter 3, we have developed new fast algorithms for the translation operators in 2D in Chapter 5 and in 3D in Chapters 7 and 8. We demonstrated that all translation operators in the 2D and 3D FMM could be decomposed as products of structured matrices and therefore each of them requires only $O(p \log p)$ operations. If implemented naively, these algorithms have a common problem of numerical instability, even though they are very fast. In this chapter we discuss the implementation of these fast algorithms and modifications required to achieve numerical stability. We first describe the algorithm to compute the product of a Toeplitz matrix and a vector in $O(p \log p)$ time. Next we show how to implement efficient algorithms for the product of the Pascal matrix or its related matrices and a vector. We introduce a parameter α into the implementation to achieve stability. Then we present details on implementation of translation operators for the 2D and 3D FMM. We demonstrate the effectiveness of this parameter α with an example of the multipole translation operator in 2D. We will also show the effectiveness of this parameter α with numerical experiments for FMM calculations

in the next chapter.

9.1 Implementation of fast multiplication of a Toeplitz matrix and a vector

A Toeplitz matrix is completely determined by its first column and first row. Denote a Toeplitz matrix with first column vector

$$c = \begin{bmatrix} c_0 & c_1 & c_2 & ... & c_{p-1} \end{bmatrix}' \tag{9.1}$$

and first row vector

$$r = \begin{bmatrix} c_0 & r_1 & r_2 & ... & r_{p-1} \end{bmatrix} \tag{9.2}$$

by $Toep(c, r')$. We want to compute the product

$$\tilde{y} = Toep(c, r') * \tilde{x} \tag{9.3}$$

in $O(p \log p)$ time, given $Toep(c, r')$ and $\tilde{x}$ is a vector with length p. This can be achieved by first embedding $Toep(c, r')$ into a $2p \times 2p$ circulant matrix C_{2p}, since

$$y = \begin{bmatrix} \tilde{y} \\ ... \end{bmatrix} = C_{2p} \cdot \begin{bmatrix} \tilde{x} \\ 0 \end{bmatrix} \equiv \begin{bmatrix} Toep & S_p \\ S_p & Toep \end{bmatrix} \cdot \begin{bmatrix} \tilde{x} \\ 0 \end{bmatrix} = \begin{bmatrix} Toep \cdot \tilde{x} \\ S_p \cdot \tilde{x} \end{bmatrix} \tag{9.4}$$

and

$$S_p = \begin{bmatrix} 0 & c_{n-1} & ... & c_2 & c_1 \\ r_{n-1} & 0 & ... & c_3 & c_2 \\ ... & ... & ... & ... & ... \\ r_2 & r_3 & ... & 0 & c_{n-1} \\ r_1 & r_2 & ... & r_{n-1} & 0 \end{bmatrix}. \tag{9.5}$$

Then the product $y = C_{2p} \cdot x$ can be formed in the following four steps where u is the vector

$$\begin{bmatrix} c_0 & c_1 & \ldots & c_{n-1} & 0 & r_{n-1} & \ldots & r_1 \end{bmatrix}', \tag{9.6}$$

and $x = \begin{bmatrix} \tilde{x}' & 0 & 0 & \ldots & 0 \end{bmatrix}'$ is the expanded vector of $\tilde{x}$ with p zeros,

1. f =FFT(x),
2. g =FFT(u),
3. $h = f. * g$, here h is a vector from the element-wise multiplication of the vectors f and g.
4. y =IFFT(h).

Then $\tilde{y}$ is obtained from first p components of y.

9.2 Pascal matrix and its relatives

A fast algorithm for computing the product of a Pascal matrix and a vector is based on the decomposition (3.2). Given a Pascal matrix P of size $p \times p$, we can factor it into

$$P = \text{diag}(v_1) \cdot T(c_p, r_p') \cdot \text{diag}(v_2), \tag{9.7}$$

where

$$v_1 = \begin{bmatrix} 1 & 1! & 2! & 3! & \cdots & (p-1)! \end{bmatrix}', \tag{9.8}$$

$$v_2 = \begin{bmatrix} 1 & 1! & \frac{1}{2!} & \frac{1}{3!} & \cdots & \frac{1}{(p-1)!} \end{bmatrix}', \tag{9.9}$$

$$c_p = \begin{bmatrix} 1 & 1! & \frac{1}{2!} & \frac{1}{3!} & \cdots & \frac{1}{(p-1)!} \end{bmatrix}', \tag{9.10}$$

$$r_p = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{9.11}$$

For a vector

$$a = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{p-1} \end{bmatrix}', \tag{9.12}$$

the product

$$Pa = \mathrm{diag}(v_1) \cdot T(c_p, r'_p) \cdot \mathrm{diag}(v_2) \cdot a \tag{9.13}$$

requires three matrix-vector products, two of which involve diagonal matrices, one of which involves Toeplitz matrix. Therefore the product Px can be done in $O(p \log p)$ time instead of $O(p^2)$ time required by straightforward matrix-vector product. A naive implementation of the above method shows that the precision gets worse as p gets bigger. The reason for this is that the entries in the Toeplitz matrix and the vector v_2 are of very different magnitudes, varying approximately from 1 to $\frac{1}{(p-1)!}$. When we compute the matrix-vector product, we need to compute the FFT of the two vectors

$$u = \begin{bmatrix} 1 & 1! & \frac{1}{2!} & \frac{1}{3!} & \cdots & \frac{1}{(p-1)!} & 0 & 0 & \cdots & 0 \end{bmatrix}', \tag{9.14}$$

$$x = \begin{bmatrix} a_0 & 1!a_1 & \frac{1}{2!}a_2 & \frac{1}{3!}a_3 & \cdots & \frac{1}{(p-1)!}a_{p-1} & 0 & 0 & \cdots & 0 \end{bmatrix}'. \tag{9.15}$$

For a large p, when we compute the FFT of u and x, the final result would be the same if we simply treat the entries such as $\frac{1}{(p-1)!}$ as zeros, and this causes the numerical instability. Therefore we need to find a way to increase the effect of entries of smaller magnitude by bringing all nonzero terms in u and x to the same magnitude. This can be done by multiplying or dividing the entries by some constant factors and still preserving the same structure, viz. a Toeplitz matrix. Indeed, the Pascal matrix can be expressed by introducing a new parameter α as follows,

$$P(\alpha) = \mathrm{diag}(v_1(\alpha)) \cdot Toep\left[c_p(\alpha), r'_p\right] \cdot \mathrm{diag}(v_2(\alpha)), \tag{9.16}$$

where

$$v_1(\alpha) = \left[\begin{array}{ccccccc} 1 & \frac{1}{\alpha} & \frac{2}{\alpha^2} & \frac{6}{\alpha^3} & ... & \frac{(p-1)!}{\alpha^{p-1}} \end{array} \right]', \tag{9.17}$$

$$v_2(\alpha) = \left[\begin{array}{ccccccc} 1 & \frac{\alpha}{1} & \frac{\alpha^2}{2} & \frac{\alpha^3}{6} & ... & \frac{\alpha^{p-1}}{(p-1)!} \end{array} \right]', \tag{9.18}$$

$$c_p(\alpha) = \left[\begin{array}{ccccccc} 1 & \frac{\alpha}{1} & \frac{\alpha^2}{2} & \frac{\alpha^3}{6} & ... & \frac{\alpha^{p-1}}{(p-1)!} \end{array} \right]'. \tag{9.19}$$

With this factorization, it is possible to implement a fast, numerically stable algorithm by selecting a proper value of α.

Now we need to select a proper α so that the magnitude of maximum and minimum of the nonzero entries in the vector $v_2(\alpha)$ and $c_p(\alpha)$ are at the same level. The Fast Fourier transform is applied to the two vectors

$$x(\alpha) = \left[\begin{array}{cccccccccc} a_0 & \frac{\alpha a_1}{1} & \frac{\alpha^2 a_2}{2} & \frac{\alpha^3 a_3}{6z^3} & ... & \frac{\alpha^{p-1} a_{p-1}}{(p-1)!} & 0 & 0 & ... & 0 \end{array} \right]' \tag{9.20}$$

and

$$u(\alpha) = \left[\begin{array}{cccccccccc} 1 & \alpha & \frac{1}{2}\alpha^2 & \frac{1}{6}\alpha^3 & ... & \frac{1}{(p-1)!}\alpha^{p-1} & 0 & 0 & ... & 0 \end{array} \right]'. \tag{9.21}$$

Assuming all entries of $a = \left[\begin{array}{ccccc} a_0 & a_1 & a_2 & ... & a_{p-1} \end{array} \right]'$ are of the same magnitude, the entries in $x(\alpha)$ and the entries in $u(\alpha)$ are of the same magnitude. So we can choose one proper value α so that all nonzero entries of $x(\alpha)$ and $u(\alpha)$ are as close as possible.

Let us consider the function $f(m) = \frac{\alpha^m}{m!}$, $m = 0, 1, 2, ..., p-1$ and we want the maximum and minimum of this function to be as close as possible. If $\alpha \geq p-1$, then $f_{\min} = 1$, and $f_{\max} = \frac{\alpha^{p-1}}{(p-1)!}$. In this case, we should choose $\alpha = p-1$. If $1 \leq \alpha < p-1$, then when $0 \leq m \leq \alpha$, $f_{\min} = 1$, and $f_{\max} = \frac{\alpha^{[\alpha]}}{([\alpha])!}$; when $\alpha < m \leq p-1$, $f_{\min} = \frac{\alpha^{p-1}}{(p-1)!}$, and $f_{\max} = \frac{\alpha^{[\alpha]}}{([\alpha])!}$. Comparing these two cases, it is easy to see that the proper value of α should be $1 \leq \alpha < p-1$ and we need to

select α such that

$$\min_{\alpha}(\max(\frac{\alpha^{\alpha}}{\alpha!}, \frac{\alpha^{\alpha}(p-1)!}{\alpha^{p-1}\alpha!})). \tag{9.22}$$

Using Stirling's formula,

$$\frac{\alpha^{\alpha}}{\alpha!} \approx \frac{\alpha^{\alpha}e^{\alpha}}{(2\pi)^{0.5}\alpha^{\alpha+0.5}} = \frac{e^{\alpha}}{(2\pi\alpha)^{0.5}}; \tag{9.23}$$

and

$$\frac{\alpha^{\alpha}(p-1)!}{\alpha^{p-1}\alpha!} \approx \frac{\alpha^{\alpha}(2\pi)^{0.5}(p-1)^{p-1+0.5}e^{\alpha}}{\alpha^{p-1}(2\pi)^{0.5}\alpha^{\alpha+0.5}e^{p-1}} = \sqrt{\frac{p-1}{\alpha}}\left(\frac{p-1}{\alpha e}\right)^{p-1} e^{\alpha}. \tag{9.24}$$

So when $\frac{p-1}{\alpha e} \approx 1$, we would achieve our objective, that is, when

$$\alpha \approx \frac{p-1}{e}, \tag{9.25}$$

the magnitude of the nonzero entries of $x(\alpha)$ and $u(\alpha)$ are about the closest. This can provide us an initial value for the proper value of α.

Since the fast translation operators in 2D and 3D involve the Pascal matrix, its relatives and other Toeplitz matrices, the effectiveness of this technique of introducing a new parameter α to bring stability to the fast algorithms is demonstrated through its implementation in these fast translation operators. We will show an example with the fast multipole translation operator: how the parameter α varies affects the accuracy of the translation operator in next section. We will discuss the values of p for which this technique can be used in the end of this chapter and show its effectiveness in Chapter 10.

We would also like to note that the modification does not have much effect on the complexity of the operator: once p is known (e.g. from the error bound), we can precompute $c_p(\alpha)$ and the FFT of $u(\alpha)$ and store it before we start the computation of the matrix-vector product. The vectors $x(\alpha)$ and $v_1(\alpha)$ can be computed from $c_p(\alpha)$. For a fixed p, we can even precompute and test a few

values of α so that we achieve numerical stability. This is often required in the implementation of the fast translation operators later on.

Since for the transpose matrix P' of the Pascal matrix and the matrix PP, the product of the Pascal matrix P and its transpose P', the corresponding decompositions are very similar and the process to introduce the parameter α is the same, we can do it in the same way as for the Pascal matrix.

9.3 Implementation of the fast translation operators in 2D

9.3.1 Multipole translation operator

The multipole translation operator is expressed through matrices using (4.18). Combining (5.1) and (3.2), we have the following,

$$SS(z) = \mathrm{diag}(v_1) \cdot Toep \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ \frac{1}{2} & 0 \\ \frac{1}{6} & 0 \\ \vdots & \vdots \\ \frac{1}{(p-1)!} & 0 \end{bmatrix} \cdot \mathrm{diag}(v_2) \tag{9.26}$$

Where

$$v_1 = \begin{bmatrix} 1 & z & 2z^2 & 6z^3 & \ldots & z^{p-1}\,(p-1)! \end{bmatrix}', \tag{9.27}$$

and

$$v_2 = \begin{bmatrix} 1 & \frac{1}{z} & \frac{1}{2z^2} & \frac{1}{6z^3} & \ldots & \frac{z^{-p+1}}{(p-1)!} \end{bmatrix}'. \tag{9.28}$$

The multipole translation operator computes a product $b = SS(z) \cdot a$ for a given multipole expansion coefficient vector

$$a = \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_p \end{bmatrix}'. \tag{9.29}$$

The cost of direct implementation of this operator is $O(p^2)$. Let us consider an efficient implementation.

Note that the multiplication of a diagonal matrix by a vector can be done in $O(p)$ time and the product of a Toeplitz matrix and a vector can be done in $O(p \log p)$ time. Thus we can compute the product $b = SS(z)a$ in $O(p \log p)$ time.

From (5.1), we can see that a Pascal matrix is present in the multipole translation operator. So it has the same problem as the Pascal matrix. We only need to apply the same technique to this translation operator.

The multipole translation operator can be expressed by introducing a new parameter α as follows,

$$SS(z) = \mathrm{diag}(v_1) \cdot Toep \begin{bmatrix} 1 & 1 \\ \alpha & 0 \\ \frac{\alpha^2}{2} & 0 \\ \frac{\alpha^3}{6} & 0 \\ \vdots & \vdots \\ \frac{\alpha^{p-1}}{(p-1)!} & 0 \end{bmatrix} \cdot \mathrm{diag}(v_2), \tag{9.30}$$

where

$$v_1 = \begin{bmatrix} 1 & \frac{z}{\alpha} & \frac{2z^2}{\alpha^2} & \frac{6z^3}{\alpha^3} & \dots & \frac{z^{p-1}(p-1)!}{\alpha^{p-1}} \end{bmatrix}', \tag{9.31}$$

and

$$v_2 = \begin{bmatrix} 1 & \frac{\alpha}{z} & \frac{\alpha^2}{2z^2} & \frac{\alpha^3}{6z^3} & \dots & \frac{\alpha^{p-1}}{(p-1)!z^{p-1}} \end{bmatrix}'. \tag{9.32}$$

Then with a properly selected α, we can apply the fast algorithm to the product of a Toeplitz matrix and a vector.

To compute the product, the fast Fourier transform is applied to two vectors

$$x = \begin{bmatrix} a_1 & \frac{\alpha a_2}{z} & \frac{\alpha^2 a_3}{2z^2} & \frac{\alpha^3 a_4}{6z^3} & \dots & \frac{\alpha^{p-1} a_p}{(p-1)! z^{p-1}} & 0 & 0 & \dots & 0 \end{bmatrix}' \tag{9.33}$$

and

$$u = \begin{bmatrix} 1 & \alpha & \frac{1}{2}\alpha^2 & \frac{1}{6}\alpha^3 & \dots & \frac{1}{(p-1)!}\alpha^{p-1} & 0 & 0 & \dots & 0 \end{bmatrix}'. \tag{9.34}$$

If in vector x, we assume that $\frac{a_k}{z^k} \tilde{} O(1)$ (see Theorem 4.1, $a_k = \sum_{i=1}^{m} \frac{-q_i (z_i - c)^k}{k}$, and $|z_i - c| < z$), then the entries in x and the entries in u are of the same magnitude. We can select a proper value α in the same way as the case for the Pascal matrix so that the numerical stability is achieved. We show with a numerical experiment (see figure 9.1) that as α changes, the accuracy of the operator changes. It shows that choosing a proper value of α is an effective way to obtain accurate result. More discussion about the relation between the accuracy and number p of terms can be found in the numerical result chapter.

9.3.2 Local translation operator

The local translation operator is expressed by matrices as in equation (4.23). It acts by computing a product $b = RR(z) \cdot a$ for a given multipole expansion coefficient vector

$$a = \begin{bmatrix} a_0 & a_1 & a_2 & \dots & a_{p-1} \end{bmatrix}'. \tag{9.35}$$

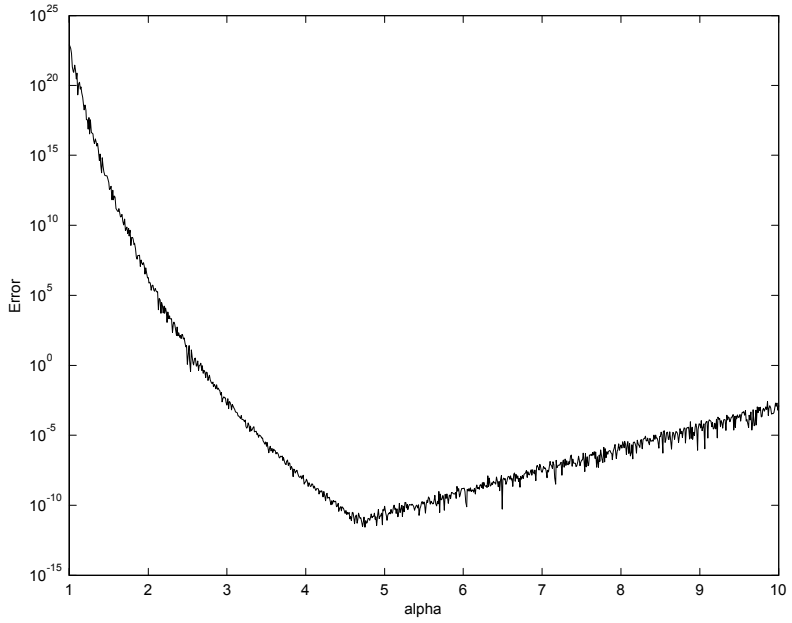

Figure 9.1: As α changes, the accuracy of the operator changes. In this experiment, we set $p = 61$, $z_0 = 0.4$, $z = 2.0$, that is, a multipole expansion centered at point $z_0 = 0.4$ with 61 terms and randomly generated 61 numbers as its coefficients are evaluated at point $z = 2.0$. This evaluation is taken as the true value. The error showed in graph is the difference between the true value and the value evaluated at the same point of the translated multipole expansion using the fast algorithm with the parameter α. It shows with a proper value of α, the fast algorithm can produce very accurate result, which is even better than the straightforward implementation of the translation operators (this is not showed in this graph, but showed in the same experiment).

From (5.7) and (3.2), we have,

$$RR(z) = diag(v_3) \cdot Toep \left[\begin{array}{cc} 1 & 1 \\ 0 & 1 \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{6} \\ \vdots & \vdots \\ 0 & \frac{1}{(p-1)!} \end{array} \right] \cdot diag(v_4), \tag{9.36}$$

where

$$v_3 = \left[\begin{array}{ccccccc} 1 & -\frac{1}{z} & \frac{1}{2z^2} & -\frac{1}{6z^3} & \ldots & \frac{(-z)^{-p+1}}{(p-1)!} \end{array} \right]', \tag{9.37}$$

and

$$v_4 = \left[\begin{array}{cccccc} 1 & -z & 2z^2 & -6z^3 & \ldots & (-z)^{p-1}\,(p-1)! \end{array} \right]'. \tag{9.38}$$

The cost of direct implementation of this operator is $O(p^2)$. An efficient implementation of this operator, which is also of the time complexity of $O(p \log p)$, can be done very similarly to the implementation of multipole translation operator.

The local translation operator can be implemented based on the following,

$$RR(z) = \mathrm{diag}(v_3) \cdot Toep \left[\begin{array}{cc} 1 & 1 \\ 0 & \alpha \\ 0 & \frac{\alpha^2}{2} \\ 0 & \frac{\alpha^3}{6} \\ \vdots & \vdots \\ 0 & \frac{\alpha^{p-1}}{(p-1)!} \end{array} \right] \cdot \mathrm{diag}(v_4), \tag{9.39}$$

where

$$v_3(\alpha) = \left[\begin{array}{ccccccc} 1 & -\frac{\alpha}{z} & \frac{\alpha^2}{2z^2} & -\frac{\alpha^3}{6z^3} & \ldots & \frac{\alpha^{p-1}}{(p-1)!(-z)^{p-1}} \end{array} \right]', \tag{9.40}$$

and

$$v_4(\alpha) = \begin{bmatrix} 1 & -\frac{z}{\alpha} & \frac{2z^2}{\alpha^2} & -\frac{6z^3}{\alpha^3} & \ldots & \frac{(-z)^{p-1}(p-1)!}{\alpha^{p-1}} \end{bmatrix}', \tag{9.41}$$

with α selected as in (9.25).

9.3.3 Multipole to local translation operator

Similarly, the multipole to local translation operator is expressed by matrices as in equation (4.38). It is to compute a product $b = SR(z) \cdot a$ for a given multipole expansion coefficient vector

$$a = \begin{bmatrix} a_1 & a_2 & \ldots & a_p \end{bmatrix}'. \tag{9.42}$$

It is well known that if

$$I_p = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}, \tag{9.43}$$

is the backward identity permutation, then for any Hankel matrix H, I_pH is a Toeplitz matrix, for any Toeplitz matrix T, I_pT is a Hankel matrix [Kailath99]. It is also true that $I_p = I_p' = I_p^{-1}$. Therefore, from equations (5.8) and (3.30), we

have,

$$SR(z) = \mathrm{diag}(v_5) \cdot I_p \cdot Toep \left[\begin{array}{cc} (p-1)! & (p-1)! \\ (p-2)! & p! \\ (p-3)! & (p+1)! \\ \vdots & \vdots \\ 1! & (2p-3)! \\ 0! & (2p-2)! \end{array} \right] \cdot \mathrm{diag}(v_6), \tag{9.44}$$

where

$$v_5 = \left[\begin{array}{ccccccc} 1 & \frac{1}{z} & \frac{1}{2z^2} & \frac{1}{6z^3} & \dots & \frac{z^{-p+1}}{(p-1)!} \end{array} \right]', \tag{9.45}$$

and

$$v_6 = \left[\begin{array}{ccccccc} -\frac{1}{z} & \frac{1}{z^2} & -\frac{1}{2z^3} & \frac{1}{6z^4} & \dots & \frac{(-z)^{-p}}{(p-1)!} \end{array} \right]'. \tag{9.46}$$

The cost of direct implementation of this operator is $O(p^2)$. An efficient implementation of this operator, which is also of the time complexity of $O(p \log p)$, can be done very similarly to the implementation of multipole translation operator.

The multipole to local translation operator can be implemented based on the following,

$$SR(z) = \mathrm{diag}(v_5) \cdot I_p \cdot Toep \left[\begin{array}{cc} \frac{(p-1)!}{\alpha^p} & \frac{(p-1)!}{\alpha^p} \\ \frac{(p-2)!}{\alpha^{p-1}} & \frac{p!}{\alpha^{p+1}} \\ \frac{(p-3)!}{\alpha^{p-2}} & \frac{(p+1)!}{\alpha^{p+2}} \\ \vdots & \vdots \\ \frac{1!}{\alpha^2} & \frac{(2p-3)!}{\alpha^{2p-2}} \\ \frac{0!}{\alpha} & \frac{(2p-2)!}{\alpha^{2p-1}} \end{array} \right] \cdot \mathrm{diag}(v_6), \tag{9.47}$$

where

$$v_5(\alpha) = \left[\begin{array}{ccccccc} 1 & \frac{\alpha}{z} & \frac{\alpha^2}{2z^2} & \frac{\alpha^3}{6z^3} & \dots & \frac{\alpha^{p-1}}{(p-1)!z^{p-1}} \end{array} \right]', \tag{9.48}$$

and

$$v_6(\alpha) = \left[\begin{array}{ccccc} -\frac{\alpha}{z} & \frac{\alpha^2}{z^2} & -\frac{\alpha^3}{2z^3} & \cdots & \frac{\alpha^p}{(-z)^p(p-1)!} \end{array} \right]', \tag{9.49}$$

and α is selected as in (9.25).

9.4 Implementation of the fast rotation algorithm in 3D

In this section we consider the implementation of Algorithm 7.4 (fast rotation algorithm) based on factorizations (7.5), (7.8), and (7.11). This algorithm basically involves three different types of rotation: $D(00\alpha)$, $D(0\frac{\pi}{2}0)$, $D(\beta 00)$. Two of these $D(00\alpha)$, and $D(\beta 00)$ are rotation about z-axis, which amounts to a product of a unitary diagonal matrix and a vector; there should pose no difficulty in the implementation. Next we discuss $D(0\frac{\pi}{2}0)$.

Recall that

$$D^{(j)}(0\frac{\pi}{2}0) = diag(v_1) * diag(v_2) * rot180(P) * diag(v_3) * diag(v_2) * P * diag(v_4), \tag{9.50}$$

where $v_1(m) = [(j+m)!(j-m)!]^{-\frac{1}{2}}$, $v_2(m) = (-1)^{j+m}$, $v_3(m) = [2^{-m}]$, $v_4(m) = [(j+m)!(j-m)!]^{\frac{1}{2}}$, $m = -j, \cdots, 0, \cdots j$, P is Pascal matrix, and $rot180(P)$ is a matrix resulted from rotating Pascal matrix 180 degree. We can take out factor $(j!)^{-1}$ from all entries in matrix $diag(v_1)$ and factor $(j!)$ from all entries in matrix $diag(v_4)$ and multiply these two factors together to reduce possible numerical problems.

It is easy to verify that

$$rot180(P) = I_p * P * I_p, \tag{9.51}$$

where I_p is the backward identity permutation matrix. Therefore we have only to deal with the matrix-vector product for the Pascal matrix, which we have done before.

9.5 Implementation of the fast coaxial translation operators in 3D

9.5.1 Multipole translation operator

The multipole translation operator is expressed by matrices as in equation (8.2). It acts by computing the products for $k = -p, -p+1, ..., -1, 0, 1, ..., p$,

$$M^{(k)} = SS^{(k)}(\rho) * O^{(k)} \tag{9.52}$$

where

$$M^{(k)} = [\ M^k_{|k|} \quad M^k_{|k|+1} \quad M^k_{|k|+2} \quad ... \quad M^k_p\ ]', \tag{9.53}$$

$$O^{(k)} = [\ O^k_{|k|} \quad O^k_{|k|+1} \quad O^k_{|k|+2} \quad ... \quad O^k_p\ ]'. \tag{9.54}$$

From (8.8), we have

$$SS^{(k)}(\rho) = \mathrm{diag}\begin{bmatrix} 1/A^k_{|k|} \\ \rho/A^k_{|k|+1} \\ \rho^2/A^k_{|k|+2} \\ ... \\ \rho^{p-|k|}/A^k_p \end{bmatrix} \cdot Toep\begin{bmatrix} 1 & 1 \\ 1 & 0 \\ \frac{1}{2} & 0 \\ \frac{1}{6} & 0 \\ \vdots & \vdots \\ \frac{1}{(p-|k|)!} & 0 \end{bmatrix} \cdot \mathrm{diag}\begin{bmatrix} A^k_{|k|} \\ A^k_{|k|+1}/\rho \\ A^k_{|k|+2}/\rho^2 \\ ... \\ A^k_p/\rho^{p-|k|} \end{bmatrix}. \tag{9.55}$$

The cost of direct implementation of this operator is $O(p^2)$. Since the multiplication of a diagonal matrix by a vector can be done in $O(p)$ time and the product of

a Toeplitz matrix and a vector can be done in $O(p \log p)$ time, we can implement this product with an algorithm of time complexity of $O(p \log p)$.

Noticing again that the entries in the Toeplitz matrix and two diagonal matrices are of very different magnitudes, we need to make modifications, which is very similarly to the implementation of multipole translation operator in 2D, to this factorization of matrix as the following,

$$SS^{(k)}(\rho) = \text{diag}\begin{bmatrix} \frac{1}{A^k_{|k|}} \\ \frac{\rho}{\alpha A^k_{|k|+1}} \\ \frac{\rho^2}{\alpha^2 A^k_{|k|+2}} \\ \cdots \\ \frac{\rho^{p-|k|}}{\alpha^{p-|k|} A^k_p} \end{bmatrix} \cdot Toep \begin{bmatrix} 1 & 1 \\ \alpha & 0 \\ \frac{\alpha^2}{2} & 0 \\ \frac{\alpha^3}{6} & 0 \\ \vdots & \vdots \\ \frac{\alpha^{p-1}}{(p-|k|)!} & 0 \end{bmatrix} \cdot \text{diag}\begin{bmatrix} A^k_{|k|} \\ \alpha A^k_{|k|+1}/\rho \\ \alpha^2 A^k_{|k|+2}/\rho^2 \\ \cdots \\ \alpha^{p-|k|} A^k_p/\rho^{p-|k|} \end{bmatrix}. \tag{9.56}$$

In this factorization, we can also take out factor $A^k_{|k|}$ from all entries in the last diagonal matrix and factor $\frac{1}{A^k_{|k|}}$ from all entries in the first diagonal matrix and multiply these two constant factors together.

9.5.2 Local translation operator

The local translation operator is expressed by matrices as in equation (8.12). It acts by computing the products for $k = -p, -p+1, ..., -1, 0, 1, ..., p$,

$$L^{(k)} = RR^{(k)}(\rho) * O^{(k)} \tag{9.57}$$

where

$$L^{(k)} = [\ L^k_{|k|} \quad L^k_{|k|+1} \quad L^k_{|k|+2} \quad ... \quad L^k_p\ ]', \tag{9.58}$$

$$O^{(k)} = [\ O^k_{|k|} \quad O^k_{|k|+1} \quad O^k_{|k|+2} \quad ... \quad O^k_p\ ]', \tag{9.59}$$

From (8.18), we have

$$RR^{(k)}(\rho) = \mathrm{diag}\begin{bmatrix} A^k_{|k|} \\ -A^k_{|k|+1}/\rho \\ A^k_{|k|+2}/\rho^2 \\ \cdots \\ A^k_p/(-\rho)^{p-|k|} \end{bmatrix} \cdot Toep \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 0 & \frac{1}{2} \\ 0 & \frac{1}{6} \\ \vdots & \vdots \\ 0 & \frac{1}{(p-|k|)!} \end{bmatrix} \cdot \mathrm{diag}\begin{bmatrix} 1/A^k_{|k|} \\ -\rho/A^k_{|k|+1} \\ \rho^2/A^k_{|k|+2} \\ \cdots \\ (-\rho)^{p-|k|}/A^k_p \end{bmatrix}. \tag{9.60}$$

The cost of direct implementation of this operator is $O(p^2)$. Since the multiplication of a diagonal matrix by a vector can be done in $O(p)$ time and the product of a Toeplitz matrix and a vector can be done in $O(p \log p)$ time, we can implement this product with an algorithm of time complexity of $O(p \log p)$.

Noticing again that the entries in the Toeplitz matrix and two diagonal matrices are of very different magnitude, we need to make modifications, which is very similarly to the implementation of local translation operator in 2D, to this factorization of matrix as the following,

$$RR^{(k)}(\rho) = \mathrm{diag}\begin{bmatrix} A^k_{|k|} \\ -\alpha A^k_{|k|+1}/\rho \\ \alpha^2 A^k_{|k|+2}/\rho^2 \\ \cdots \\ \alpha^{p-|k|} A^k_p/(-\rho)^{p-|k|} \end{bmatrix} \cdot Toep \begin{bmatrix} 1 & 1 \\ 0 & \alpha \\ 0 & \frac{\alpha^2}{2} \\ 0 & \frac{\alpha^3}{6} \\ \vdots & \vdots \\ 0 & \frac{\alpha^{p-1}}{(p-|k|)!} \end{bmatrix} \cdot \mathrm{diag}\begin{bmatrix} \frac{1}{A^k_{|k|}} \\ \frac{-\rho}{\alpha A^k_{|k|+1}} \\ \frac{\rho^2}{\alpha^2 A^k_{|k|+2}} \\ \cdots \\ \frac{(-\rho)^{p-|k|}}{\alpha^{p-|k|} A^k_p} \end{bmatrix}. \tag{9.61}$$

In this factorization, we can also take out factor $A^k_{|k|}$ from all entries in the first diagonal matrix and factor $\frac{1}{A^k_{|k|}}$ from all entries in the last diagonal matrix and multiply these two constant factors together.

9.5.3 Multipole to local translation operator

Similarly, the multipole to local translation operator is expressed by matrices as in equation (8.21). It acts by computing the products for $k = -p, -p+1, ..., -1, 0, 1, ..., p$,

$$L^{(k)} = SR^{(k)}(\rho) * O^{(k)} \tag{9.62}$$

where

$$L^{(k)} = [\ L^k_{|k|} \quad L^k_{|k|+1} \quad L^k_{|k|+2} \quad ... \quad L^k_p\ ]', \tag{9.63}$$

$$O^{(k)} = [\ O^k_{|k|} \quad O^k_{|k|+1} \quad O^k_{|k|+2} \quad ... \quad O^k_p\ ]', \tag{9.64}$$

It is well known that if

$$I_p = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & \cdots & 1 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 1 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \end{bmatrix}, \tag{9.65}$$

is the backward identity permutation, then for any Hankel matrix H, I_pH is a Toeplitz matrix. Therefore, from (8.27), we have

$$SR^{(k)}(\rho) = (-1)^k \cdot \mathrm{diag}(v_1) \cdot I_p^{(k)} \cdot Toep(c, r') \cdot \mathrm{diag}(v_2), \tag{9.66}$$

where

$$v_1 = \begin{bmatrix} \frac{A^k_{|k|}}{\rho^{|k|+1}} \\ \frac{A^k_{|k|+1}}{\rho^{|k|+2}} \\ \frac{A^k_{|k|+2}}{\rho^{|k|+3}} \\ ... \\ \frac{A^k_p}{\rho^{p+1}} \end{bmatrix}, v_2 = \begin{bmatrix} \frac{A^k_{|k|}}{(-\rho)^{+|k|}} \\ \frac{A^k_{|k|+1}}{(-\rho)^{|k|+1}} \\ \frac{A^k_{|k|+2}}{(-\rho)^{|k|+2}} \\ ... \\ \frac{A^k_p}{(-\rho)^p} \end{bmatrix}, c = \begin{bmatrix} \frac{\alpha^{|k|+p+1}}{A^0_{|k|+p}} \\ \frac{1}{A^0_{|k|+p-1}} \\ \cdots \\ \frac{1}{A^0_{2|k|+1}} \\ \frac{1}{A^0_{2|k|}} \end{bmatrix}, r = \begin{bmatrix} \frac{1}{A^0_{|k|+p}} \\ \frac{1}{A^0_{|k|+p+1}} \\ \cdots \\ \frac{1}{A^0_{2p-1}} \\ \frac{1}{A^0_{2p}} \end{bmatrix}' \tag{9.67}$$

The cost of direct implementation of this operator is $O(p^2)$. Since the multiplication of a diagonal matrix by a vector can be done in $O(p)$ time and the product of a Toeplitz matrix and a vector can be done in $O(p \log p)$ time, we can implement this product with an algorithm of time complexity of $O(p \log p)$.

Noticing again that the entries in the Toeplitz matrix and two diagonal matrices are of very different magnitudes, we need to make modifications, which is very similarly to the implementation of local translation operator in 2D, to this factorization of matrix as the following,

$$SR^{(k)}(\rho) = (-1)^k \cdot \mathrm{diag}(v_1(\alpha)) \cdot I_p^{(k)} \cdot Toep(c(\alpha), r(\alpha)') \cdot \mathrm{diag}(v_2(\alpha)), \qquad (9.68)$$

where

$$v_1 = \begin{bmatrix} \frac{\alpha A^k_{|k|}}{\rho^{|k|+1}} \\ \frac{\alpha^2 A^k_{|k|+1}}{\rho^{|k|+2}} \\ \frac{\alpha^3 A^k_{|k|+2}}{\rho^{|k|+3}} \\ \cdots \\ \frac{\alpha^{p-|k|+1} A^k_p}{\rho^{p+1}} \end{bmatrix}, v_2 = \begin{bmatrix} \frac{A^k_{|k|}}{(-\rho)^{+|k|}} \\ \frac{\alpha A^k_{|k|+1}}{(-\rho)^{|k|+1}} \\ \frac{\alpha^2 A^k_{|k|+2}}{(-\rho)^{|k|+2}} \\ \cdots \\ \frac{\alpha^{p-|k|} A^k_p}{(-\rho)^p} \end{bmatrix}, c = \begin{bmatrix} \frac{\alpha^{-|k|-\rho-1}}{A^0_{|k|+p}} \\ \frac{\alpha^{-|k|-\rho}}{A^0_{|k|+p-1}} \\ \cdots \\ \frac{\alpha^{-2|k|-2}}{A^0_{2|k|+1}} \\ \frac{\alpha^{-2|k|-1}}{A^0_{2|k|}} \end{bmatrix}, r = \begin{bmatrix} \frac{\alpha^{-|k|-\rho-1}}{A^0_{|k|+p}} \\ \frac{\alpha^{-|k|-\rho-2}}{A^0_{|k|+p+1}} \\ \cdots \\ \frac{\alpha^{-2p}}{A^0_{2p-1}} \\ \frac{\alpha^{-2p-1}}{A^0_{2p}} \end{bmatrix}' \qquad (9.69)$$

In this factorization, we can also take out factor $A^k_{|k|}$ from all entries in the first diagonal matrix, factor $A^k_{|k|}$ from all entries in the last diagonal matrix, and factor $\frac{1}{A^0_{2|k|}}$ from the Toeplitz matrix and multiply these three constant factors together.

9.6 Further discussion of the stability

We discuss how we can further address the stability problem in case of high precision requirement. In 2D as we will demonstrate in numerical experiments in the next chapter, with number p of terms used in all multipole/local expansions

equal to 21, we achieve 15 digit accuracy. It is also shown that the introduction of parameter α works well for $p = 49$ (in our numerical experiments, we use $\alpha = \frac{p}{3e}$). This implies that we could reach 30 digit accuracy for the 2D case with this strategy, which is more than enough for any applications.

In three dimensional space, as demonstrated in our numerical experiments (see Table 10.5) the same strategy proves to stabilize the fast translation operators. The FMM with the fast translation operators produces the same accuracy as the FMM with the original translation operators with the number of terms in the expansions up to 400. In practice, the fast translation operators are stable.

That said, we would like to explore what we can do when this strategy stops working. It will eventually stop working since the entries will be of very different magnitudes with some large number p. In this case the corresponding Toeplitz matrix could be subdivided into blocks of little Toeplitz matrices, each of which consists elements of similar magnitude, therefore the above method could be used *on each block* to help to bring stability to the computation with the same complexity. Furthermore, if we view each Toeplitz block as one entry in the whole matrix, the whole matrix is a Toeplitz matrix again as the following,

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & A_0 & A_{-1} & A_{-2} & A_{-3} & A_{-4} & A_{-5} & \cdots \\ \cdots & A_1 & A_0 & A_{-1} & A_{-2} & A_{-3} & A_{-4} & \cdots \\ \cdots & A_2 & A_1 & A_0 & A_{-1} & A_{-2} & A_{-3} & \cdots \\ \cdots & A_3 & A_2 & A_1 & A_0 & A_{-1} & A_{-2} & \cdots \\ \cdots & A_4 & A_3 & A_2 & A_1 & A_0 & A_{-1} & \cdots \\ \cdots & A_5 & A_4 & A_3 & A_2 & A_1 & A_0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \tag{9.70}$$

where $\{A_i, i = ..., -2, -1, 0, 1, 2...\}$ are Toeplitz matrices. Therefore fast algorithm could be applied to each of the individual matrix as well as the whole matrix.

Chapter 10

Numerical Results

The original translation operators (Chapters 4, 6) and the more efficient new translation operators (Chapter 5, 7, 8, 9) for the two and three dimensional coulombic potential are implemented and used with the black box FMM program developed in C++ by Gumerov, Duraiswami, and Borovikov [Gumerov2003]. In the implementation of the new translation operators, the FFTW package [Frigo2003] developed by Frigo and Johnson is used to perform the fast Fourier transform. Numerical experiments have been carried out on a personal computer with a Pentium III 931 MHz processor.

The results of our experiments are summarized in the following tables and graphs with all timings (wall clock time) given in seconds. The intensity of the particles for simplicity reason is taken to be 1.

10.1 Results for two dimensions

In the first experiment, the number of terms used in all expansions is 21, which is the number that results in double precision accuracy, and the maximum number of particles in a box (clustering parameter s) at the finest level is set in a way

so that it yields the optimal number of levels for all calculations. The results are reported in Table 10.1. The first column is the number N of particles in the simulation. The next three columns show the wall clock time $T_{new}, T_{old}, T_{direct}$ respectively for the FMM algorithm with new translation operators, the FMM algorithm with original translation operators, and the direct calculation. Each integer in the parenthesis next to the wall clock time is the number of levels in the data structure which results the fastest calculation. The fifth and the sixth columns present the relative errors E_{new}, E_{old} of the FMM algorithm with the new and old translation operators, with direct calculations are used to determine the relative error via the formula

$$E = \max_{i=1,\ldots N} \frac{\left|\phi_i - \tilde{\phi}_i\right|}{\left|\tilde{\phi}_i\right|}, \tag{10.1}$$

where ϕ_i is the value of the potential at the ith particle position obtained from the FMM algorithms, and $\tilde{\phi}_i$ is the value obtained from direct calculations. This error formula is much stricter than the l^2 norm.

From the table, we see that the algorithm using the new translation operators has the same magnitude of relative error as the one using the original translation operators. Since the CPU time for direct calculations are much higher than the other two methods, we only plotted the CPU times of the FMM with new and old translation operators and first few timings of the direct calculations. From the graph, we see that the algorithm with the new translation operators are about twice faster than the one with the original translation operators for a fixed number of terms of $p = 21$.

In the second experiment, the number N of particles is 8192, and the clustering parameter s is set to be 40 (Tables 10.2). The first column in the table is the number p of the terms in the multipole or local expansions. The second and the

third columns contain the CPU times T_{new}, T_{old} of the FMM using the new and old translation operators, respectively. The last two columns contain the relative error for these two calculations.

From the table, we can see that the two different translation operators introduce the same amount of error, and $p = 21$ is sufficient for double precision. Increasing the value of p beyond this does not raise the accuracy, since machine precision is already reached. For higher precision, we need not only to increase p, but also need to do all calculations in extended precision. Here, one purpose is to verify that the new efficient operators are capable of producing highly accurate results if we condition the operators properly even though there exist large numbers (e.g., of magnitude ~ 10^{62}) as well as small numbers ($O(1)$) in the decomposed matrices. From the second set of graphs (figure 10.3, 10.4), we can see that the curve of CPU time versus p for the old translation operators is a parabola. Indeed the curve can be well approximated by

$$T_{old} = 0.0369p^2 + 0.0126p + 4.0920. \tag{10.2}$$

$$T_{old} = 0.0402p^2 - 0.0007p + 12.0596. \tag{10.3}$$

The curve of CPU time versus p for the new translation operators can be approximated by a straight line as follows,

$$T_{new} = 0.1633p + 4.3553 \tag{10.4}$$

$$T_{new} = 0.1588p + 11.9252. \tag{10.5}$$

(The fit holds except at three values of p for the new operators. The reason for these three $p = 35, 39, 47$ is that the FFT lengths for the multipole to local translation operator are 68, 76, and 92, which contain the factors 17, 19, and 23,

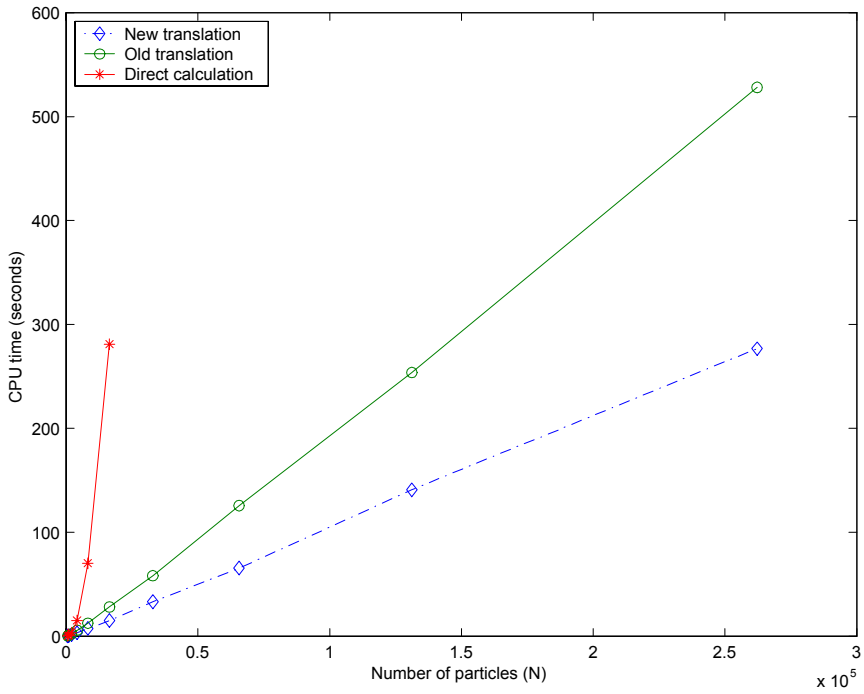

Figure 10.1: Timing results (wall clock time) in 2D for the direct calculations and the FMM using the old translation operators and the new translation operators with number of terms in all expansions is $p = 21$ and the optimal number of levels in each calculation is selected so that each calculation is the fastest possible. The data plotted in this figure are the same as the data given in table 10.1.

respectively, each of which is not optimal in the FFT calculation.). This implies that in the case of high precision computation, as the value of p gets larger, the FMM with the new translation operators is far more efficient than that with the old translation operators. In this experiment, when $p = 17$, the new algorithm uses only half of the time used by the original one; when $p = 25$, it uses one-third the time; and when $p = 29$, it uses one-fourth of the time; when $p = 49$, it uses one-eighth of the time.

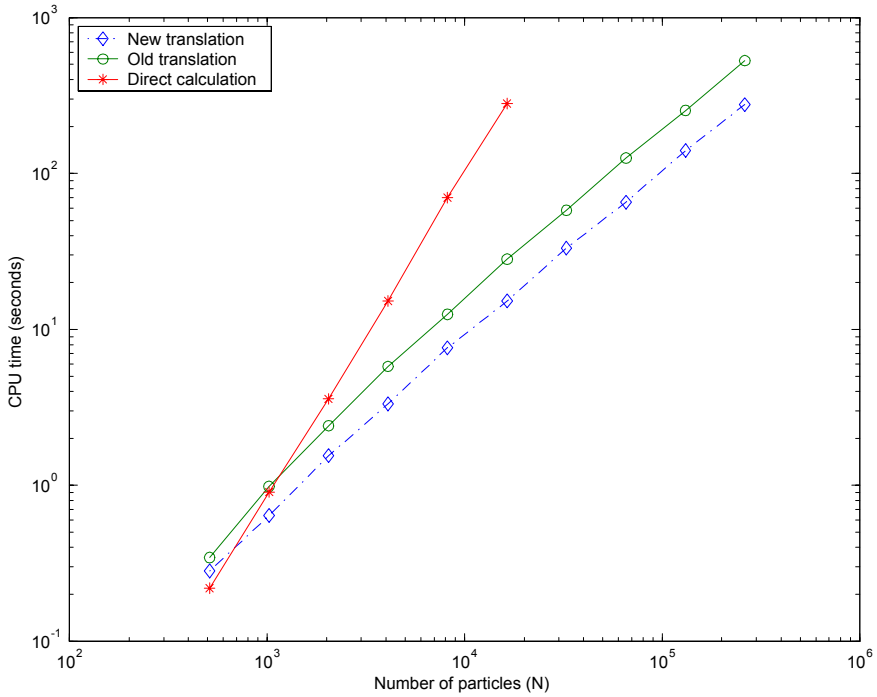

Figure 10.2: Log-log plot of the same data as in last figure. Timing results (wall clock time) in 2D for the direct calculations and the FMM using the old translation operators and the new translation operators with number of terms in all expansions is $p = 21$ and the optimal number of levels in each calculation is selected so that each calculation is the fastest possible. From this graph, it is clear that direct calculations are $O(N^2)$, while both FMM calculations are $O(N)$, with the FMM with the new translation operators has a smaller coefficient than that of the old translation operators.

N	$T_{new}(levels)$	$T_{old}(levels)$	T_{direct}	E_{new}	E_{old}
512	0.282 (3)	0.344 (2)	0.219	$1.94348e-015$	$2.45679e-015$
1024	0.641 (3)	0.984 (3)	0.906	$3.04228e-015$	$3.04228e-015$
2048	1.547 (4)	2.407 (3)	3.578	$4.65273e-015$	$4.34731e-015$
4096	3.328 (4)	5.781 (4)	15.219	$6.95426e-015$	$6.95426e-015$
8192	7.609 (5)	12.5 (4)	70.016	$8.96796e-015$	$9..41502e-015$
16384	15.265 (5)	28.25 (5)	281.172	$1.33988e-014$	$1.36313e-014$
32768	33.297 (6)	58.141 (5)	1130.34	$2.00279e-014$	$2.09988e-014$
65536	65.531 (6)	125.61 (6)	4501.7	$2.80928e-014$	$2.82561e-014$
131072	140.843 (7)	253.563 (6)	18002.4	$4.59491e-014$	$4.63124e-014$
262144	276.64 (7)	528.14 (7)	$*$	$*$	$*$

Table 10.1: Timing results (wall clock time) in 2D for the direct calculations and the FMM using the old translation operators and the new translation operators with number of terms in all expansions is $p = 21$ and the optimal number of levels in each calculation is selected so that each calculation is the fastest possible. The data given in this table are the same as the data plotted in figure 10.1 and 10.2.

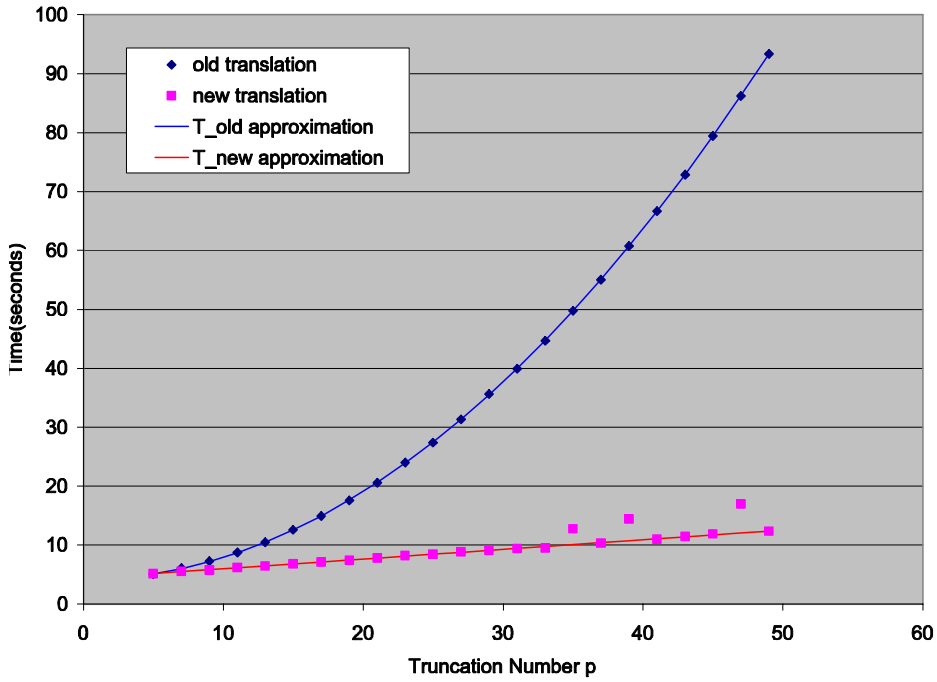

Figure 10.3: Timing results (wall clock time) for 2D with fixed number of particles $N = 8192$ and cluster parameter $s = 40$. The data plotted in this figure are the same as the data given in Table 10.2

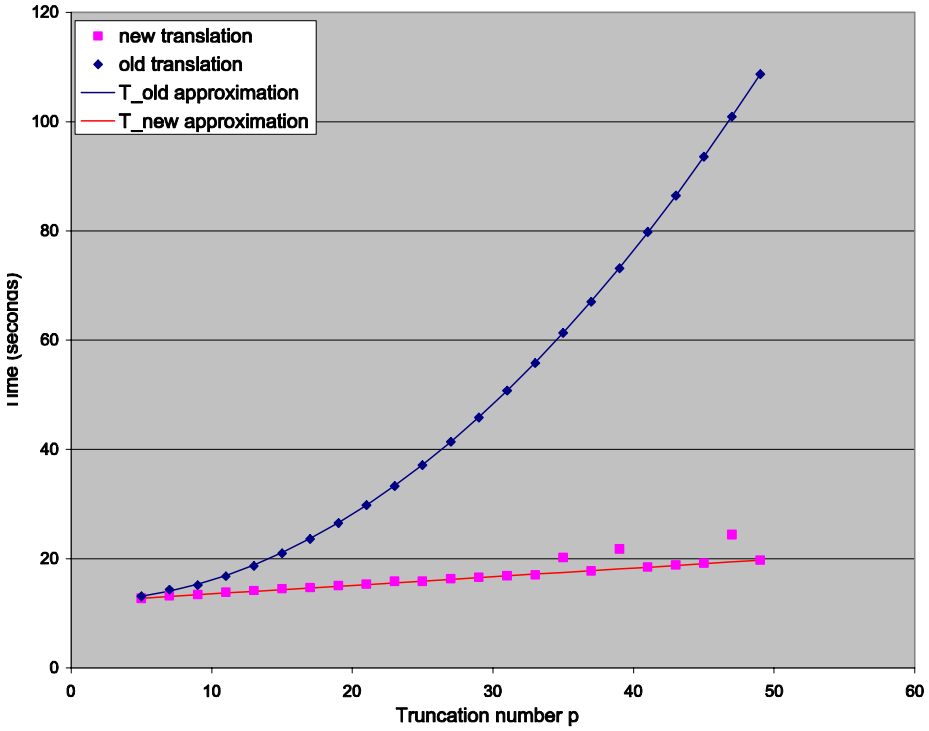

Figure 10.4: Timing results (wall clock time) for 2D with fixed number of particles $N = 16384$ and cluster parameter $s = 40$. The data plotted in this figure are the same as the data given in Table 10.3 .

p	T_{new}	T_{old}	E_{new}	E_{old}
5	5.172	5.078	$5.3701E-05$	$5.3701E-05$
7	5.516	6.063	$2.3504E-07$	$2.3504E-07$
9	5.781	7.25	$3.6499E-08$	$3.6499E-08$
11	6.218	8.719	$1.1580E-09$	$1.1580E-09$
13	6.437	10.454	$3.1867E-11$	$3.1867E-11$
15	6.828	12.579	$3.0385E-12$	$3.0388E-12$
17	7.093	14.954	$5.6928E-14$	$5.6928E-14$
19	7.438	17.61	$1.0647E-14$	$9.7717E-15$
21	7.796	20.593	$8.9680E-15$	$9.3052E-15$
23	8.219	23.968	$1.0145E-14$	$9.2772E-15$
25	8.406	27.391	$9.1240E-15$	$9.2772E-15$
27	8.813	31.344	$9.1226E-15$	$9.2772E-15$
29	9.094	35.657	$1.0731E-14$	$9.2772E-15$
31	9.391	39.937	$9.1226E-15$	$9.2772E-15$
33	9.5	44.656	$8.8133E-15$	$9.2772E-15$
35	12.75	49.719	$9.5864E-15$	$9.2772E-15$
37	10.312	55.047	$2.0832E-14$	$9.2772E-15$
39	14.406	60.734	$1.0398E-14$	$9.2772E-15$
41	11	66.641	$1.7394E-14$	$9.2772E-15$
43	11.422	72.859	$4.0149E-14$	$9.2772E-15$
45	11.86	79.421	$1.5406E-14$	$9.2772E-15$
47	17	86.157	$9.1142E-14$	$9.2772E-15$
49	12.359	93.344	$5.0979E-14$	$9.2772E-15$

Table 10.2: Timing results (wall clock time) for 2D with fixed number of particles $N = 8192$ and cluster parameter $s = 40$. The timing given in this table are plotted in Figure 10.3

p	T_{new}	T_{old}	E_{new}	E_{old}
5	12.719	13.062	$5.3933E-05$	$5.3933E-05$
7	13.188	14.312	$2.1549E-07$	$2.1549E-07$
9	13.406	15.125	$3.5302E-08$	$3.5302E-08$
11	13.828	16.765	$1.2355E-09$	$1.2355E-09$
13	14.156	18.672	$2.8376E-11$	$2.8377E-11$
15	14.5	20.969	$3.2839E-12$	$3.2842E-12$
17	14.672	23.578	$6.8581E-14$	$6.8321E-14$
19	15.016	26.484	$1.4722E-14$	$1.3631E-14$
21	15.313	29.718	$1.3399E-14$	$1.3631E-14$
23	15.875	33.25	$1.3963E-14$	$1.3631E-14$
25	15.859	37.141	$1.3631E-14$	$1.3631E-14$
27	16.344	41.375	$1.3995E-14$	$1.3631E-14$
29	16.593	45.828	$1.3918E-14$	$1.3631E-14$
31	16.86	50.703	$1.4182E-14$	$1.3631E-14$
33	17.032	55.782	$1.5135E-14$	$1.3631E-14$
35	20.188	61.344	$1.4399E-14$	$1.3631E-14$
37	17.734	67.016	$2.1599E-14$	$1.3631E-14$
39	21.766	73.094	$2.0007E-14$	$1.3631E-14$
41	18.469	79.797	$2.0181E-14$	$1.3631E-14$
43	18.86	86.422	$3.9793E-14$	$1.3631E-14$
45	19.188	93.516	$1.8615E-14$	$1.3631E-14$
47	24.375	100.907	$1.3358E-13$	$1.3631E-14$
49	19.704	108.64	$4.8065E-14$	$1.3631E-14$

Table 10.3: Timing results (wall clock time) for 2D with fixed number of particles $N = 16384$ and cluster parameter $s = 40$. The timing given in this table are plotted in Figure 10.4

10.2 Results for three dimensions

The settings for the experiments in 3D are similar to that in two dimensions. The same relative error formula is used.

From table 10.4 and figure 10.5, we see that the FMM with the old translation operators or the new translation operators have the same good accuracy with 121 terms retained in the expansion. The FMM in 3D is much slower than that in 2D for two reasons: a) the interaction list in 3D most time consists of 189 boxes while in 2D it has only 27 boxes; b) the number of terms used in the expansion in 3D is much more than that in 2D. The FMM with the new and old translation operators is faster that the direct calculation for large number of particles. The FMM with old translation operators often can not compete with the direct calculation when the number of particles are relatively small. The FMM with the new translation operators often are two times faster than that with the old translation operators; It breaks even with the direct calculation for a reasonable number of particles.

From figure 10.7 and table 10.5, it is obvious that the CPU time for the FMM with the old translation operators is a second order polynomial in p^2, that is $O(p^4)$, while the CPU time for the FMM with the new translation operators is $O(p^2 \log p)$, where p^2 is the number of terms in all multipole/local expansions in 3D. Indeed the curves can be well approximated by

$$T_{old} = 0.0014p^4 - 0.0519p^2 + 15.8037. \tag{10.6}$$

$$T_{new} = 0.0336p^2 + 15.7909. \tag{10.7}$$

It is easy to see that both have the same order of accuracy. It confirms that the break-even p is 4, and for higher accuracy, the number of terms grows larger,

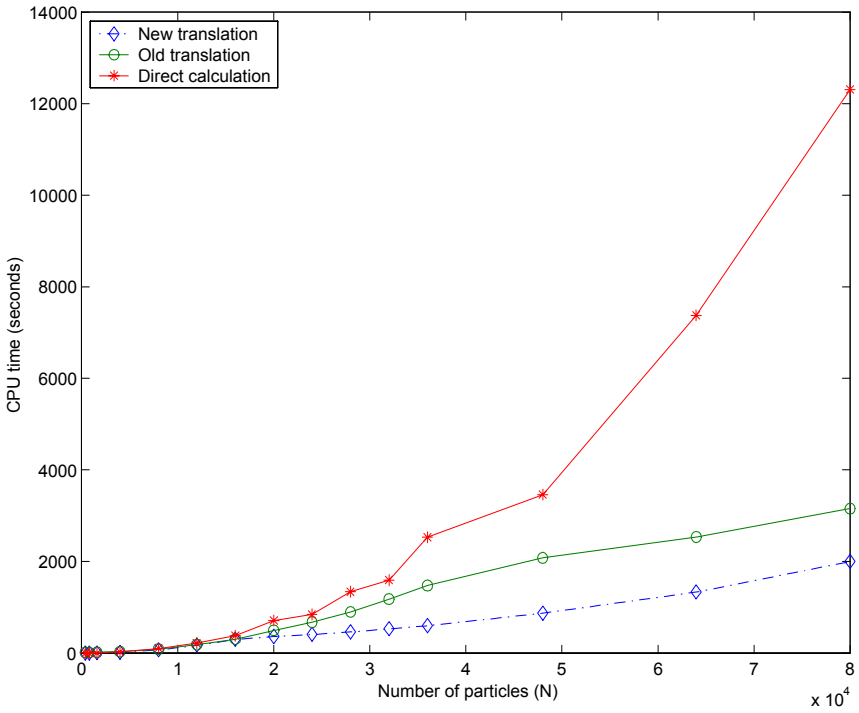

Figure 10.5: Timing results (wall clock time) in 3D for the direct calculations and the FMM using the old translation operators and the new translation operators with number of terms in all expansions is $p^2 = 121$ and the optimal number of levels in each calculation is selected so that each calculation is the fastest possible. The data plotted in this figure are the same as the data given in table 10.4.

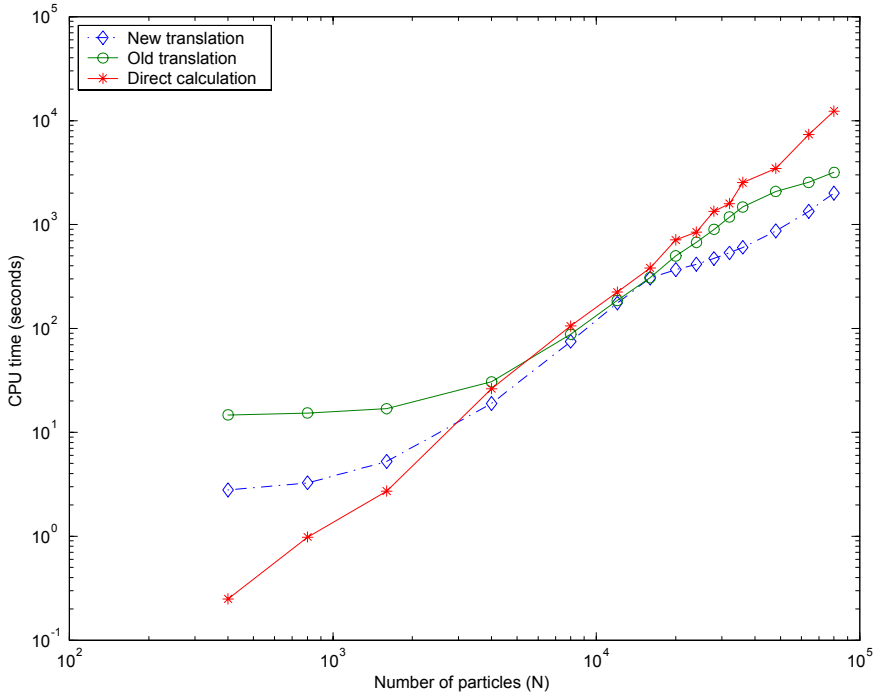

Figure 10.6: Log-log plot of the same data as in last figure. Timing results (wall clock time) in 3D for the direct calculations and the FMM using the old translation operators and the new translation operators with number of terms in all expansions is $p^2 = 121$ and the optimal number of levels in each calculation is selected so that each calculation is the fastest possible. From this graph, it is clear that direct calculations are $O(N^2)$, while both FMM calculations are $O(N)$, with the FMM with the new translation operators has a smaller coefficient than that of the old translation operators.

N	$T_{new}(levels)$	$T_{old}(levels)$	T_{direct}	E_{new}	E_{old}
400	2.797 (2)	14.719 (2)	0.25	$4.39601e-009$	$4.39566e-009$
800	3.266 (2)	15.375 (2)	0.984	$3.95576e-09$	$3.95769e-009$
1600	5.235 (2)	16.875 (2)	2.719	$1.91777e-09$	$1.91865e-009$
4000	18.906 (2)	30.5 (2)	26.25	$3.54763e-09$	$3.5483e-009$
8000	75.234 (2)	88.14 (2)	105.641	$1.75868e-09$	$1.75752e-009$
12000	175.922 (2)	186.094 (2)	223.203	$1.52882e-09$	$1.52884e-009$
16000	305.469 (2)	308.141 (2)	382.672	$1.59152e-09$	$1.59146e-009$
20000	367.953 (3)	498.5 (2)	713.688	$2.06895e-09$	$1.86678e-009$
24000	414.172 (3)	674.922 (2)	845.078	$2.25412e-09$	$1.90272e-009$
28000	469.906 (3)	898.156 (2)	1339.41	$1.57744e-08$	$3.4595e-009$
32000	534.031 (3)	1179.33 (2)	1591.27	$1.14218e-08$	$3.44941e-009$
36000	604.203 (3)	1476.73 (2)	2531.98	$8.24206e-09$	$3.43159e-009$
48000	872 (3)	2078.97 (3)	3461.5	$5.37855e-09$	$5.3784e-009$
64000	1336.13 (3)	2537.84 (3)	7376.8	$4.21805e-09$	$4.21818e-009$
80000	2002.81 (3)	3155.61 (3)	12308.6	$4.16503e-09$	$4.16509E-09$

Table 10.4: Timing results (wall clock time) in 3D for the direct calculations and the FMM using the old translation operators and the new translation operators with number of terms in all expansions is $p^2 = 121$ and the optimal number of levels in each calculation is selected so that each calculation is the fastest possible. The data given in this table are the same as the data plotted in figure 10.5 and 10.6.

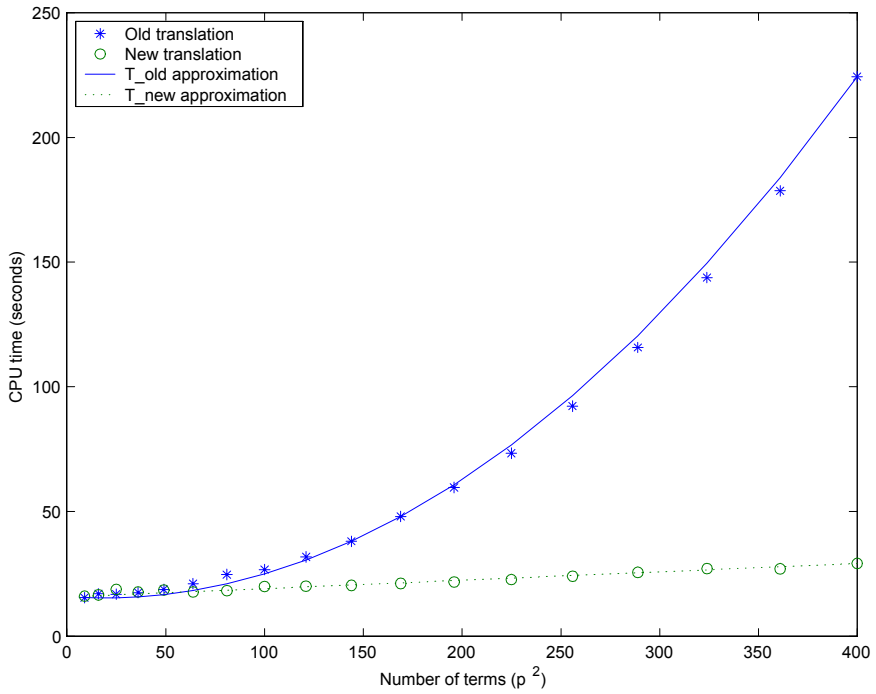

Figure 10.7: Timing results (wall clock time) in 3D for the FMM using the old translation operators and the new translation operators with $N = 4000$ and $s = 120$. The data plotted in this figure are the same as the data given in Table 10.5

the new translation operators become increasingly more efficient than the old translation operators.

p^2	T_{new}	T_{old}	E_{new}	E_{old}
9	16.093	15.453	$7.01992e-04$	$7.01992e-004$
16	16.547	17.031	$1.78884e-04$	$1.78884e-004$
25	18.797	16.828	$4.13579e-05$	$4.13579e-005$
36	17.75	17.484	$5.61547e-06$	$5.61547e-006$
49	18.469	18.703	$9.08662e-07$	$9.08662e-007$
64	17.703	21.031	$2.29129e-07$	$2.29129e-007$
81	18.265	24.703	$2.1003e-08$	$2.1003e-008$
100	19.937	26.641	$1.22377e-08$	$1.22384e-008$
121	20.016	31.75	$3.54973e-09$	$3.5483e-009$
144	20.375	38.047	$2.9289e-10$	$2.94224e-010$
169	21.125	48.016	$2.35364e-10$	$2.34372e-010$
196	21.719	59.625	$7.25083e-11$	$7.16183e-011$
225	22.687	73.313	$8.8203e-12$	$7.51667e-012$
256	24.047	92.203	$3.48925e-12$	$3.3386e-012$
289	25.594	115.75	$3.48706e-12$	$1.76288e-012$
324	27.078	143.781	$7.25795e-12$	$7.20734e-012$
361	26.984	178.531	$4.0068e-11$	$1.56376e-011$
400	29.218	224.312	$1.98628e-10$	$1.73987e-011$

Table 10.5: Timing results in 3D with fixed number of particles $N = 4000$ and cluster parameter $s = 120$. The data given in this table are the same as the data plotted in Figure 10.7

Chapter 11

Conclusion and Future Work

The efficient translation operators for Coulombic potentials in two or three dimensions have been developed. The complexity of the FMM is improved to $O(p \log pN)$, where p is the number of terms retained in the multipole/local expansions. Numerical experiments indicate that for 2D, when optimal clustering parameter for both algorithms are selected, the new translation operators speed up the whole algorithm by a factor of two to three. For 3D, the new translation operators speed up the algorithm even more. From the complexity analysis, for calculations requiring higher precision, the truncation number p has to be larger, the new translation operators will speed up the algorithm even more significantly. In the process of developing new translation operators, two type of fast algorithms has been developed. One is for the Pascal matrix, its transpose, and the product of these two to perform the matrix-vector multiplications. The other is to efficiently compute the new coefficients of a field given as a spherical harmonic expansion under rotation transformation.

We have identified constant matrices in the translation operators. Accelerating the matrix-vector product for these constant matrices will speed up the translation operators, thus improving the complexity of the whole algorithm.

Some future work can be devoted to this.

The fast rotation transform can be applied to the potential governed by the three dimensional Helmholtz equation as well. We need only to develop similar coaxial translation operators for it to achieve $O(p \log p)$ complexity for the corresponding translation operators.

BIBLIOGRAPHY

[Abramowitz65] Abramowitz, M. and I. A. Stegun, *Handbook of Mathematical Functions*, Dover Publications, 1965.

[Bai2000] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe and H. van der Vorst, editors, *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.

[Barnes86] Josh Barnes and Piet Hut, *A hierarchical $O(N \log N)$ force-calculation algorithm*, Nature, 324, Dec. 1986, pp. 446-449.

[Call93] Gregory S. Call and Daniel J. Velleman, *Pascal's matrices*, American Math. Monthly, 100, April, 1993, pp. 372-376.

[Carrier88] J. Carrier L. Greengard and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statist. Comput., 9, 1988, pp.669-686.

[Cheng99] H. Cheng, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm in three dimensions*, J. Comput. Phys., 155, 1999, pp. 468-498.

[Choi99] Cheol Ho Choi, Joseph Ivanic, Mark S. Gordan, and Klaus Ruedeberg, *Rapid and stable determination of rotation matrices between spherical harmonics by direct recursion*, J. Chem. Phys., 111(19), Nov. 1999, pp. 8825-8831.

[Coifman93] R. Coifman, V. Rokhlin, S. Wandzura, *The fast multipole method for the wave equation: a pedestrian prescription*, IEEE Antennas and Propagation Mag., 35(3), June 1993, pp.7-12.

[Cooley65] J. W. Cooley and J. W. Tukey *An algorithm for the machine calculation of complex Fourier series*, Math Comp. 19, 1965, pp. 297-301.

[Driscoll94] James R. Driscoll and Dennis M. Healy,Jr., *Computing Fourier transforms and convolutions on the 2-sphere*, Advances in Applied Mathematics, 15, 1994, pp.202-250.

[Driscoll97] J. R. Driscoll, D. M. Healy, JR., and D. N. Rockmore, *Fast discrete polynomial transforms with applications to data analysis for distance transitive graphs*, SIAM J. COMPUT. 26(4), 1997, pp1066-1099.

[Edelman] Alan Edelman and Gilbert Strang, *Pascal matrices*, MIT.

[Edmonds60] A.R. Edmonds, *Angular Momentum in Quantum Mechanics,* Princeton University Press, Princeton, 1960.

[Elliott96] William D. Elliott And John A. Board,JR., *Fast fourier transform accelerated fast multipole algorithm*, SIAM J. Sci. Comput., 17(2), March 1996, pp. 398-415.

[Epton95] M. A. Epton and B. Dembart, *Multipole translation theory for the three-dimensional Laplace and Helmholtz equations*, SIAM J. Sci. Comput., 16(4), July 1995, pp. 865-897.

[Frigo2003] Matteo Frigo and Steven G. Johnson, *FFTW User's Manual*, MIT, 2003.

[GohbergL94] I. Gohberg and V. Olshevsky, *Complexity of multiplication with vectors for structured matrices*, Linear Algebra Appl., 202(1994), pp. 163-192.

[GohbergC94] I. Gohberg and V. Olshevsky, *Fast algorithms with preprocessing for matrix-vector multiplication problems*, J. Complexity, 10(4)(1994), pp. 411-427.

[Golub96] G. Golub and C. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, third edition, 1996.

[Greengard88] L. Greengard, *The Rapid Evaluation Of Potential Fields in Particle Systems,* MIT Press, Cambridge, 1988.

[Greengard94] Leslie Greengard, *Fast algorithms for classical physics*, Science, 265, Aug. 1994, pp. 909-914.

[Greengard98] Leslie Greengard, Jingfang Huang, Vladimir Rokhlin, Stephen Wandzura, *Accelerating fast multipole methods for Helmholtz equation at low frequencies*, IEEE

Computational Science and Engineering, 5(3),July-September 1998.

[Greengard87] L. Greengard, and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comput. Phys., 73, 1987, pp. 325-348.

[GreengardL88] L. Greengard, and V. Rokhlin, *Rapid evaluation of potential fields in three dimensions*, in Vortex Methods, edited by C. Anderson and C. Greengard, Lecture Notes in Mathematics, Springer-Verlag, Berlin, vol. 1360, 1988, pp.121-141.

[Greengard97] L. Greengard, V.Rokhlin, *A new version of the fast multipole method for the Laplace equation in three dimensions*, Acta Numerica, 1997, pp.229-269.

[Gumerov2001] Nail A. Gumerov and Ramani Duraiswami, *Fast, exact, and stable computation of multipole translation and rotation coefficients for the 3-D Helmholtz equation*, UMIACS TR 2001-4. University of Maryland, College Park, 2001. (accepted to *SIAM J. Sci. Stat. Comput.*)

[Gumerov2002] Nail A. Gumerov and Ramani Duraiswami, *Multiple scattering from N spheres using multipole reexpansion*, J. Acoust. Soc. Am.,112 (6), pp. 2688-2701, Dec. 2002.

[Gumerov2003] Nail A. Gumerov, Ramani Duraiswami and Eugene A. Borovikov, *Data structure, optimal choice of parameters, and complexity results for generalized multilevel fast multipole meth-*

ods in d dimensions, UMIACS TR 2003-28, University of Maryland, College Park, 2003.

[Hrycak98] Tomasz Hrycak and Vladimir Rokhlin, *An improved fast multipole algorithm for potential fields*, SIAM J.Sci. Comput., 19(6), Nov. 1998, pp1804-1826.

[Kailath99] T. Kailath and A.H. Sayed, editors, *Fast Reliable Algorithms for Matrices with Structure*,SIAM, Philadelphia, 1999.

[Kellogg53] O. D. Kellogg, *Foundations of Potential Theory*, Dover, New York, 1953.

[Lu98] Hao Lu, *A generalized Hilbert matrix problem and confluent Chebyshev-Vandermonde system*, SIAM, J. MATRIX ANAL. APPL. 19(1), Jan. 1998, pp. 253-276.

[Makadia2003] Ameesh Makadia and Kostas Daniilidis, *Direct 3D-rotation estimation from spherical images via a generalized shift theorem*, IEEE Conf. Computer Vision and Pattern Recognition, June 2003.

[Mohlenkamp99] Martin L. Mohlenkamp, *A fast transform for spherical harmonics*, The Journal of Fourier Analysis and Applications, 5, 1999.

[Moore93] Sean S. B. Moore, Dennis M. Healy, Jr., and Daniel N. Rockmore, *Symmetry stabilization for fast discrete monomial transforms and polynomial evaluation*, Linear Algebra Appl., 192, 1993, pp. 249-299.

[Newton2002] Roger G. Newton, *Quantum Physics*, Springer, New York, 2002.

[Pan92] Victor Pan, *Complexity of computations with matrices and polynomials*, SIAM Review, 34(2), June 1992, pp.225-262.

[Perez96] Jose M. Perez-Jorda and Weitao Yang, *A concise redefinition of the solid spherical harmonics and its use in fast multipole methods*, J. Chem. Phys. 104(20), May 1996, pp.8003-8006.

[Petersen95] H.G. Petersen, E.R. Smith and D. Soelvason, *Error estimates for the fast multipole method. II. the three-dimensional case*, Proc. R. Soc. London, Series A, 448, 1995, pp. 401-418.

[Petersen94] H.G. Petersen, D. Soelvason, and J. W. Perram, *The very fast multipole method*, J. Chem. Phys. 101(10), Nov. 1994, pp. 8870-8876.

[Risbo96] T. Risbo, *Fourier transform summation of Legendre series and D-functions,* Journal of Geodesy, 70, 1996, pp. 383-396.

[Rokhlin83] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, Journal of Computational Physics, 60, 1983, pp. 187-207.

[Rokhlin93] V. Rokhlin, *Diagonal forms of translation operators for the Helmholtz equation in three dimensions*, Appl. Comput. Harm. Anal., 1, 1993, pp. 82-93.

[Rokhlin98] V. Rokhlin, *Sparse diagonal forms of translation operators for the Helmholtz equation in two dimensions*, Appl. Comput. Harm. Anal., 5, 1998, pp. 36-67.

[RokhlinS98] V. Rokhlin and N. Yarvin, *Generalized Gaussian quadratures and singular value decomposition of integral operators*, SIAM J.Sci. Comput., 20(2), 1998, pp699-718.

[Stein71] E. Stein and G. Weiss, *Fourier analysis on Euclidean Spaces*, Princeton University Press, Princeton, NJ, 1971.

[Stein61] Seymour Stein, *Addition theorems for spherical wave functions*, Quarterly Of Applied Mathematics, 19, 1961, pp. 15–24.

[Sun2001] Xiaobai Sun and Nikos P. Pitsianis, *A matrix version of the fast multipole method*, SIAM Review, 41(2), 2001, pp.289-300.

[Van92] Charles Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.

[White96] C. A. White and M. Head-Gordon, *Rotation around the quartic angular momentum barrier in fast multipole method calculations*, J. Chem. Phys., 105(12), Sept. 1996, pp. 5061-5067.

[Whittaker44] E. T. Whittaker and G. N. Watson, *A Course of Modern Analysis,* The university Press, Cambridge, Macmillan, New York, 1944.

[Wigner59] EUGENE P. WIGNER, *Group Theory and its Application to the Quantum Mechanics of Atomic Spectra*, Academic Press, New York, 1959