PARALLELIZATION OF THE FAST MULTIPOLE SOLUTION OF THE ELECTROMAGNETIC SCATTERING PROBLEM

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSTY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Ali Ayub M.Kalufya

September 1997

PARALLELIZATION OF THE FAST MULTIPOLE SOLUTION OF THE ELECTROMAGNETIC SCATTERING PROBLEM

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

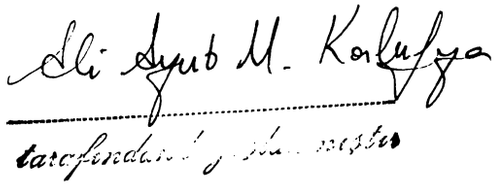
FOR THE DEGREE OF

MASTER OF SCIENCE

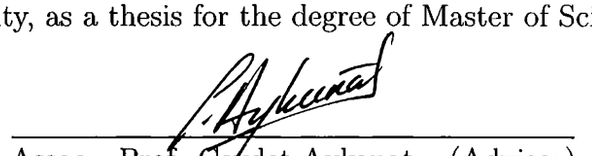
By

Ali Ayub M. Kalufya

September, 1997

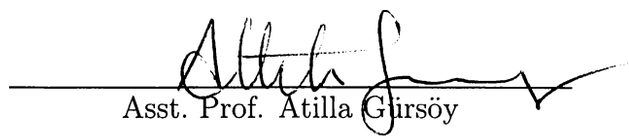


I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



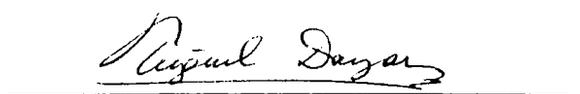
Assoc. Prof. Cevdet Aykanat (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Asst. Prof. Atilla Gürsöy

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Asst. Prof. Tuğrul Dayar

Approved for the Institute of Engineering and Sciences:

Prof. Mehmet Baray
Director of Institute of Engineering and Science

ABSTRACT

PARALLELIZATION OF THE FAST MULTIPOLE SOLUTION OF THE ELECTROMAGNETIC SCATTERING PROBLEM

Ali Ayub M. Kalufya
M.S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Cevdet Aykanat
September, 1997

The solution to the electromagnetic scattering problem may be modelled as an N-body problem. Using this model this work develops a solution that is based on a specific variant of the Fast Multipole algorithm that was proposed by V. Rokhlin[17] and modified further by Anderson[3], that is the Fast Multipole Method without multipoles. Because an iterative scheme is used, we also develop an preconditioning algorithm that is especially tailored for the solution of problems that may be modelled using N-body concept.

Moreover, in this work parallel computing is employed to improve the solution even further by developing a program that will utilize the above mentioned fast multipole method concept in parallel so as to be able to solve even larger and more interesting real-life problems in a reasonable amount of time and using minimum possible memory space.

A parallel version of the fast multipole method is developed and implemented on the Parystec Coignitive Computer 24 node multicomputer using the single program multiple data paradigm for solving the electromagnetic scattering problem in 2 dimensions.

Key words: N-body Concept, Fast Multipole Method, Blockwise Sparse Preconditioning

ÖZET

ELEKTROMANYETİK SAÇILIM PROBLEMİNİN HIZLI MULTIPOLE ÇÖZÜMÜ PARALELİSTİRİLMESİ

Ali Ayub M. Kalufya
Bilgisayar ve Enformatik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Doç. Cevdet Aykanat
Eylül, 1997

Elektromanyetik saçılım probleminin çözümü N-body problemi ile modellenebilir. Bu çalışmada bu modeli kullanarak V. Rokhlin [17] tarafından önerin ve Anderson [3] tarafından geliştirilen Hızlı Multipole Algoritmasının bir uyarlama olan bir çözüm üretir : Multipolesuz Hızlı Multipole Metodu. Inelemeli bir yöntem kullanıldığından, ozellikle N-body kavram kullanilarak modellenebilen problemlerin çözümü tüm üretilmiş bir ön-şartkandirma algoritması geliştridik.

Bu çalısmada, paralel hesaplama, daha enteresan ve büyük gerçek hayat problemlerini mâkûl bir zamanda ve olasi mınımum hafıza alanı kullanarak çözebilmek için yukarda bahsedilen hızlımultipole metodu paralelistiren bir program gelişterek çözümü daha da ilerletmek için kullanıldı.

Hızlı Multipole Metodunun paralel bir uyarlama iki boyutta elektromayetik saçılma problemini çözmek için, tekli program çoklu data yöntemi kullanilarak Parytec Coignitive Computer 24 dügümü da geliştirdi ve uygulandı.

Anahtar sözcükler: N-body kavramı, Hızlı Multipole Metodu, Seyrek Blok Ön-sartlandırması.

To my and my wife's parents.

ACKNOWLEDGEMENT

I am mostly grateful to Dr. Aykanat and Dr. Gürel for suggesting this interesting research topic. Sincerely, they have been supervising me with patience and everlasting interest all the way through the period of my graduate studies.

I am also indebted to Dr. Gürsöy and Dr. Kurç for showing keen interest to the subject matter. Their remarks and recommendations have been very constructive.

I have to express my gratitude to the technical and academic staff of Bilkent University. I am especially thankful to my classmates of my first graduate year in general, and my office mates in particular.

Lastly, albeit of most significance, I express my sincere recognition for maybe the most important role played by my wife throughout this work; without her warm motivation and concern maybe this would never be a reality.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Numerical simulation and analysis of the electromagnetic scattering phenomena is used in several engineering applications to gain important information about the systems prior to their actual implementation. However, due to the limitations in computing power and vast amounts of data and time necessary to acquire high simulation results, the size of the problem that can be solved using conventional methods is very much limited.

In this work the Fast Multipole Method is investigated, particularly its use in the above mentioned simulations to reduce both the space and time complexities of the simulations. A particular serial (sequential) implementation of the algorithm is examined aiming at improving it into producing even much better performance in terms of time and space requirements. Moreover, a sparse block matrix factorization is designed and implemented so as to exploit maximally the structural nature of the interaction matrix.

Finally, a parallel version of the improved algorithm is designed and implemented on the Parystec CC 24 node multicomputer to allow for even very much larger data sets, and to perform the calculations within reasonable time

scales.

1.2 The N-Body Problem

The electromagnetic scattering phenomena is a special case of a more generic N-body problem. An N-body is a collection of N particles each of which being acted upon by the remaining $N-1$ particles hence making the amount of computational effort required to evaluate the total force acting on each particle be of order $O(N^2)$. If we calculate the interactions using the pairwise law, then the amount of work becomes of order $\frac{1}{2}(N^2 - N)$ which is asymptotically the same $O(N^2)$ [18]. This computation is prohibitively expensive for large N which is common for real-life problems due to the amount of storage and time required to compute the interactions.

The basic notion behind the N-body is that a cluster of distant particles is replaced by a single pseudo-particle, and that as the distance to the cluster increases, the amount of particles that may together be considered as a single pseudo particle may increase. The quantity of interaction exerted by nearby particles is approximated by their interaction with this pseudo-particle.

In real-life there is a big variety of physical problems that can be modelled as a collection of interacting bodies or particles. Instances of this N-body problem can be perceived in computational fluid mechanics, molecular dynamics, plasma physics where the bodies may be ions and electrons, and astrophysics where the bodies may be stars [4].

Elsewhere [9], the N-body concept has been applied into speeding the matrix-vector multiplication which is a bottleneck in the iterative scheme that is used to solve the scattering solution. In chapter 2, a description of modeling of the scattering problem as an N-body, the model on which a particular variant of an N-body algorithm, the Fast Multipole Method, is used to speed up its computation. Moreover, the whole FMM based solution strategy is parallelized so as to be able to exploit its inherent parallel nature and utilize the

modern high performance computing parallel architectures made available by the current developments of technology.

1.3 Fast Algorithms for the N-body Problem

To speed up the N-body computation we can reduce either the frequency with which the force at an individual particle is calculated, or the computational cost of calculating the force per particle. There are several methods [12] [3] developed based on these strategies. These, including now historical algorithms such as the particle/mesh algorithms developed about 20 years ago, use approximations to calculate the interactions to reduce the execution time significantly. However, the performance of N-body computation may be improved even further by exploiting the inherent parallelism within the fast summation technique.

The Grengard-Rokhlin algorithm [10], termed the Fast Multipole Method, is the first N-Body algorithm in which the truncation error is controllable and could be fine tuned to produce a specified precision. In the FMM the pseudo particle is represented by an infinite multipole expansion centered on a sphere which contains the entire cluster. This expansion is truncated to a finite number of terms, where the number of terms taken in each expansion controls the precision. Forces may be evaluated to machine precision if required, and for a large number of particles this method can be even more accurate than the direct summation technique.

Due to its reduced computational complexity and memory requirement, the FMM can be used for problems demanding a high number of particles and a high controllable precision. The basis of this work [11] is the application of FMM in the Fast Radar Cross Section computation of large canonical two dimensional geometries. In this application, the electromagnetic scattering from two-dimensional canonical conducting strip geometries is analyzed. Although the original objective of this work was to parallelize the existing sequential

solution, prior to parallelization the existing sequential solution was improved for better performance.

A method of frequent choice for computing scattering cross sections and radiation patterns is to solve a matrix equation, $Z \times I = V$, derived from the discretization of an integral equation [18]. The number of unknowns required for accurate modelling of such problems can be very large, which severely limits the problem size. The system can be solved by factorizing the dense matrix Z, an $O(N^3)$ operation, or by using an iterative method which generally requires $O(N^2)$ operations per an iteration. In this work we use a combination of the two, that is, the dense matrix Z is considered to be a sum of two matrices, a sparse matrix Z_{NF} due to the nearfield interaction and a dense matrix Z_{FF} due to the interactions between particles in far clusters and Z_{NF} is factorized and used as a preconditioner in the iterations for which FMM is used twice to perform a fast matrix-vector calculation of the product of Z_{FF} and the current guess solution vector.

The initial work was done on the sequential code [11] to improve both its speed and memory constraints. Then to utilize the parallelism in the algorithm, the algorithm was parallelized and implemented on the Parsytec CC 24 node multicomputer. Moreover, to exploit the peculiar blockwise sparse property of the matrix resulting from the model, a blockwise sparse preconditioner was developed to speed up the convergence of the iterative process involved in the solution.

1.4 Organization of the Thesis

In this thesis the fast multipole method is investigated and used to solve a certain electromagnetic scattering problem in parallel. The thesis starts with a detailed description of the electromagnetic scattering problem in Chapter 2. The chapter throws light on the mathematical modelling of the scattering problem in such a way that the problem conforms to the characteristics of the

N-body computational paradigm.

In chapter 3 the fast multipole method is described. Here the computational model of the original FMM is explained together with the mathematical foundations of the generic multipole expansions. Moreover, the transition from the use of multipole expansions of the original Grengard-Rokhlin FMM algorithm to the use of Poisson expansions in Anderson's FMM without multipoles is discussed. Moreover the comparison between the two similar algorithms as far as computation and programming complexity is concerned is also discussed in this chapter.

Chapter 4 concentrates on the sequential solution to the electromagnetic scattering problem. A particular implementation [11] is discussed in detail to highlight various advantages and disadvantages of the Anderson's method and this particular application. For reasons explained in the chapter more emphasis is given to the matrix vector multiplication. In Chapter 4 the narration of the modifications made as part of this thesis work towards improving both time and memory efficiency of the basic implementation are given. Finally, a description is given of the modified sequential implementation that is developed in this work. Actually, this modified version is the actual basis of the parallelization efforts done in this work.

In Chapter 5 the possible strategies that can be used for distribution of data among processors in a parallel program are examined and the strategy used in this solution is described. Moreover, the method that is used in this work to create a load balance among the processors is explained. This method is based on a more generic graph partitioning algorithm.

Our parallelization strategy and its particular implementation is given in Chapter 6. Chapter 6 explains the methodologies that are used to reach the parallel algorithm and discuss the reasons for adopting such methods. Moreover, it describes the parallel $spM \times V$ and the blockwise sparse factorization algorithms that were developed as part of this work.

Chapter 7 is devoted to the experiments that were conducted to test the performance of both the original and modified sequential implementations together with the subsequent parallel implementation of the modified version. Moreover, it highlights various physical and computational aspects of the Parystec CC-24 and use them to explain the experimental results

In the closing chapter remarks and conclusions that are derivable from the experiments conducted as part of this work are discussed together with the vision about what is to be expected in the future as far the Fast Multipole Method is concerned especially on its application on solving the electromagnetic scattering problem and the more generic N-body problem.

Chapter 2

The Electromagnetic Scattering Problem

The electromagnetic scattering by two dimensional conducting cylinders is a classical electromagnetic problem and quite a number of algorithms have been developed to solve it. In most cases, the problem is modelled as a matrix equation where the unknown is the induced current. The matrix equation may be solved directly using the Method of Moments (MOM) which requires $O(N^3)$ if the Gaussian Elimination is used to invert the interaction matrix. If an iteration scheme such as the Conjugate Gradient Squared (CGS) method is used the complexity becomes that of the matrix-vector in the iteration, that is $O(N^2)$ per iteration. However, in the case of the iterative scheme it is of importance that the convergence rate be maintained as much as possible.

2.1 The Physical Model

In the electromagnetic scattering an individual metallic scatter can be decomposed into N subscatters. When an electromagnetic wave falls incident on a subscatter, the subscatter will carry a current distribution which is determined

by a discretized integral equation that is determined by the equation expressing interaction between subscatters. Therefore, for the numerical solution to obtain the field at a subscatter due to other subscatters one need N multiplications, and since there are N subscatters N^2 multiplications are needed to compute their total interactions between them. This is in complete conformity to the N-body problem and hence shows the electromagnetic scattering problem as a special case of the more general N-body problem.

2.2 The Mathematical Model

In this section we describe the procedure by which an electromagnetic scattering problem may be modelled into a matrix equation.

The surface integral that governs that scattering solution is given by

$$i\omega\mu_0 \int dS' g_o(\rho - \rho'_o) J_z(\rho') = -E_z^{inc}(\rho) \qquad \rho \in S \tag{2.1}$$

where

$$g_0(\rho - \rho_0) = \frac{i}{4} H_0^{(1)}(k\,|\rho - \rho_0|) \tag{2.2}$$

and $J_z(\rho')$ is the induced current on the surface of the scatter and $E_z^{inc}(\rho)$ is the incident field.

The equation (1) above is then discretized into [18]

$$\sum g_{ij} a_i = b_j \qquad {}_{j=1,\ldots,N} \tag{2.3}$$

where

$$g_{ij} = \left\{ \begin{array}{c} \frac{\omega\mu_0}{4}\left[1 - \frac{2i}{\pi}\ln\left(\frac{\gamma k \Delta_i}{4e}\right)\right]\Delta_i, i = j \\ \frac{\omega\mu_0}{4}\Delta_i H_0^{(1)}(\kappa\rho_{ji})\ i \neq j \end{array} \right\} \tag{2.4}$$

and

$$\begin{aligned} a_i &= J_z(\rho_i), \\ b_j &= E_z^{inc}(\rho_j) \end{aligned}$$

The result is a matrix equation

$$Z \times I = V \tag{2.5}$$

which may be solved either directly using Gaussian elimination at the cost of $O(N^3)$ operations, or iteratively at a cost of one or more matrix-vector operations which is order $O(N^2)$ operations per an iteration. In this work the Conjugate Gradient Squared iterative scheme is used to solve equation (2.5) and a fast summation algorithm that is based on the Fast Multipole Method is used to reduce the time complexity of the matrix vector multiplication operation from $O(N^2)$ to $O(N^{1.5})$. Moreover, a blockwise sparse factorization is used to supply a preconditioner matrix to improve the convergence rate of the iteration.

Chapter 3

The Fast Multipole Method

3.1 Overview

Several efforts are documented that aim at reducing the complexity of the N-body problem, among these being the particle-in-cell methods that has been used with some success in plasma physics [22]. These methods have a generic complexity of $O(N + M \log M)$, where N is the number of interacting particles and M is the number of mesh points of the N-body. The number of mesh points is normally chosen to be proportional to the number of particles, but with a small constant of proportionality so that $M \ll N$ so as to make the asymptotic complexity be of order $O(N \log N)$. Although in practice the calculations are observed to be $O(N)$, these methods have gradually and consistently lost popularity due to that they provide limited resolution and with more highly nonuniform source distributions their performance degrades significantly. Moreover, the 'fast Poisson solver' that these methods use to obtain potential values on the mesh introduce further errors by the necessity of numerical differentiation to obtain the force [4].

Particle-particle/particle-mesh methods are an improvement from the above mentioned particle-in-cell methods in the sense that the handle short-range interaction by direct computation while dealing with the far-field interaction

using the particle-mesh computation. These, in theory, do permit arbitrarily high accuracy be attained. However, when the required precision is relatively high for example in simulations of magnetic scattering from stealth vehicles the time complexity of such algorithms become prohibitively large [9].

'Gridless' methods for many-body simulation has an approximately $O(N \log N)$ complexity. They depend on the use of a monopole (centre-of-mass) approximation for computing forces over large distances and exceedingly complex data structures to keep track which among the particles are sufficiently clustered to make the approximation valid. Although the method achieves a dramatic speedup for some specific types of problems, it becomes much less efficient when the distribution of the particles is relatively uniform and the required precision is high [10].

The Fast Multipole Method (FMM) is a mathematical theory that is used to speed up summation in systems that can be modelled as N-body problems. The basis of the FMM is to combine large numbers of particles into single computational elements and then organize the resulting computations in such a way that the combining of particles is efficient while minimizing the error. In comparison to the above mentioned methods, the FMM, in its original form, uses multipole expansions to compute potentials or forces to a predetermined arbitrary precision, and with a time complexity of order $N^{1.5}$ [10]. In the sections of this chapter that follow we discuss the FMM in general with a special attention given to a particular variant of the algorithm, the fast multipole method without multipoles [3].

It is worthy noting that with time the FMM concept has evolved to the extent that, on our opinion, it should no longer be considered as an algorithm but rather a template or a parametric algorithm that can be applied into a wide range of physical problems by simply applying a change of parameters. In fact our work is based on a specific instance of the generic FMM which is actually single-stage rather than the multi-stage hierarchical FMM [21].

3.2 Informal Description of the FMM Algorithm

In this section an informal description of the FMM algorithm is given based on the original algorithm by Greengard and Rokhlin [10]. In this algorithm the domain of computation is subdivided into cells depending on the level of refinement of the domain. This lead into a hierachy of cells in which every cell x on a hierachical level is subdevided further into four cells at a lower hiearchical level. The four cells are known as the children of x and x is known as the parent. Two cells are termed nearest neighbors if they share a common boundary at the same level of refinement. Cells are defined as well separated if they are at the same level and are not nearest neighbors. The interaction set of a cell x is defined as those well separated cells which are the children of the nearest neighbors of x's parents.

The FMM algorithm consist of two major phases called passes, the upward pass and the downward pass. The upward pass begins at the finest hierachical level. The multipole expansion 3.3 is calculated for every cell. The multipole expansion associated with the parent is created from expansions of its four children by shfting the centres of its children's multipole expansions to the parent's centre. This occurs recursively from the level immediately above the finest level up to the highest level, that is the whole domain taken as a single cell.

The downward pass proceed in a direction opposite to that of the upward pass. It start with the highest hierachical level and proceed down to the finest level. Going down the hierachy, a cell at any level has a local expansion associated to it. This expansion is calculated from the multipole expansions belonging to the every cell in its interaction set. These multipole expansions describe the far-field interaction due to the well separated cells. In every cell a local expansion is formed from the coefficients of the multipole expansions associated with the cells in its interaction set then these local expansions are

summmed and shifted to the centres of the four children; this proceed recursively down the hierachy.

Finally, at the finest level, we have the description of the interaction in the finest cells due to all particles that lie in the well separated finest level cells. The amount of interaction at each particle due to the far field particles is approximated by evaluating the local expansion associated with the individual finest level cell. At this stage the only interaction not accounted for is the one due to near field particles. To complete the summing process this near field interaction is determined through normal pairwise interaction summation and added to the total interaction amount.

3.3 Computational Foundations of the FMM

The fundamental idea behind the N-body algorithms in general, and the FMM in particular is the process of combining large particles into single computational elements in such a way that if a cluster of particles is effectively distant from a particular point, then the potential of, or force due to the cluster is approximated by the potential induced by a single computational element located inside the cluster. In the original fast multipole method, the FMM by Greengard-Rokhlin , the computational element is the multipole expansion located at the centre of a disk containing the cluster of particles. However, there are other possible choices for the computation element. Novak [19] uses charge distributions over panels while Anderson [3] uses a computational element that is based on Poisson's formula for a circle in two dimensions and a sphere in three dimensions.

In the subsection that follow the mathematical model of the N-body interaction is described in detail and thereafter the computation foundations of the FMM by Greengard-Rokhlin and the Anderson FMM are given in the subsections that follow. The FMM by Greengard-Rokhlin is discussed in detail. Finally, an FMM variant that forms the actual basis of this work is examined

so as to establish the continuity between the N-body model of the electromagnetic scattering problem described in chapter 2 and the fast multipole method as explained in this section.

3.3.1 The N-body Model

An N-body is a system consisting of particles that interact with each other according to a specific physical law. An example of a system that may be modelled as an N-body is a 2-dimensional space populated by a multitude of electromagnetic charges. These charges act on each other in accordance to a well established Coulomb's Law. The detailed mathematical analysis of this model is to be found in Appendix A.1. For clarity here only the mathematics that is strictly necessary for the derivations of conclusions that follow shall be presented.

The potential due to an electromagnetic charge situated in a point z_0 in a $\mathbb{C}$ plane is given by

$$\phi_{z_0}(z) = q_0 \left(\log(z) - \sum_{k=1}^{\infty} \frac{1}{k} \left(\frac{z_0}{z} \right)^k \right) \tag{3.1}$$

Now consider a multitude of electromagnetic charges q_i located at points $z_i \; \forall z_i < r$, $i \in \{1, 2, \ldots, N\}$. The potential at an arbitrary point $z_j \in \mathbb{C}$ such that $z_j > r$ is given by

$$\Phi(z_j) = \sum_{i=1}^{N} \phi_i(z_j) \tag{3.2}$$

The calculation of the sum (3.2) above is the one that distinguish the Anderson's FMM variant from the original FMM by Greengard and Rokhlin. In the following subsections we describe a brief mathematical analysis of the two methods. Again, to avoid too much mathematical cluttering, the detailed derivations are left in the Appendix.

3.3.2 Original Fast Multipole Method

This is the version of FMM that was brought forward by Greengard and Rokhlin[10]. The use of the multipole theory to determine the sum $\Phi(z_j)$ in (3.2) is described in Appendix A.2

By applying the multipole theory the sum 3.2 becomes

$$\Phi(z_j) = Q\log(z_j) + \sum_{k=1}^{\infty} \frac{\alpha_k}{z_j^k} \tag{3.3}$$

Given that $Q = \sum_{i=1}^{N} q_i$ and $\alpha_k = \sum_{i=1}^{N} \frac{-q_i z_i^k}{k}$

The above expansion (3.3), termed the multipole expansion, gives the potential at point z_j accurately. However, this is just theoretically as it require an infinite of multipole terms to obtain the result. The number of terms of the multipole expansion is the key factor to the accuracy of the solution. In fact as described in greater detail in the Appendix A.3 that for a given precision, ε_p, the number of terms in the truncated series, p, is determined by

$$p = \lceil \log_c(\varepsilon_p) \rceil \tag{3.4}$$

However, in this FMM scheme not only is it necessary to form multipole expansions (3.3) but also there is a need to make a sequence of analytic transformations of the analytic expansions. The following are the analytical transformations that are used in the Fast Multipole algorithm. For clarity here we describe just the transformations, again detailed proofs may be obtained in [10].

The following transformation provides a mechanism for shifting the centre of a multipole expansion. Suppose that a set of charges of strength q_i, $i \in [1...N]$ all are located inside a disk D_0 of radius r with centre z_0 has the following multipole expansion

$$\phi(z) = a_0\log(z - z_0) + \sum_{k=1}^{\infty} \frac{a_k}{(z - z_0)^k} \tag{3.5}$$

it follows that an arbitrary z outside disk D_0, say in another disk D_1 of radius $r + z_0$ has the following multipole expansion

$$\phi(z) = a_0 \log(z) + \sum_{l=1}^{\infty} \frac{b_l}{z^l} \tag{3.6}$$

where

$$b_l = -a_0 z_0^l + \sum_{k=1}^{l} a_k z^{l-k} \binom{l-1}{k-1} \tag{3.7}$$

The above translation enable the conceptual consideration of the summing up of the potential of the charges within the vicinity of a disk at the logical centre of the disk so that the cluster of charges may be considered as a single supercharge located at the centre of the disk. The next analytical transformation enable the conversion of the multipole expansion (3.6) above into a local (Taylor) expansion inside a circular region of analyticity.

Again consider N charges of strength q_i, $i \in [1...N]$ all located inside a disk D_0 of radius r and centre z_0, and that $|z_0| > (c+1)r$, $c > 1$. Then the corresponding multipole expansion (3.5) converges inside another disk D_1 of radius r centered about the origin. Inside D_1, the potential due to the charges is described by the following power series

$$\phi(z) = \sum_{l=0}^{\infty} b_l \cdot z^l \tag{3.8}$$

where

$$b_0 = a_0 \log(-z_0) + \sum_{k=1}^{\infty} \frac{a_k}{z_0^k} (-1)^k \tag{3.9}$$

and

$$b_l = -\frac{a_0}{l \cdot z_l^k} + \frac{1}{z_l^k} \binom{l+k-1}{k-1} (-1)^k, \qquad \forall l \geq 1 \tag{3.10}$$

Unlike the two transformations above which have certain error bounds as discussed by length in [20], the following analytical transformation is exact and hence does not require any error bound analysis. This transformation realize the shifting of the centre of a local expansion within a region of analyticity. It is used in the FMM algorithm to enable the conceptual distribution of the

potential at the centre of a disk into the computational elements in the disk. The formula of this transformation is simple and it says

$$\sum_{k=1}^{N} a_k(z - z_0)^k = \sum_{l=1}^{N} b_l \cdot z^l \tag{3.11}$$

where

$$b_l = \sum_{k=l}^{N} a_k \binom{k}{l} (-z_0)^{k-l} \tag{3.12}$$

and $\forall z_0, z \in \mathbb{C}$ and $\{a_k\}$, $k \in [1 \dots N]$

3.3.3 The Anderson's Fast Multipole Method

Quite interestingly, the Anderson's FMM is often called the FMM without multipoles [3] due to the fact that it actually does not use multipoles as the element of computation. To avoid an obvious contradiction Anderson in his paper [3] suggested use of the term "hierarchical element methods" to the class of methods that use the basic computational structure of the fast multipole method but different computational elements. However, his suggestion could hold much significance probably due to existence of the Buttke's 3-dimensional algorithm which is actually a non-hierarchical version of the FMM [4].

As described in the Section 3.2.2., in the multipole method one forms an approximation to the potential at a point by summing the potential induced by multipole approximations for different sizes clusters of particles. In the following derivation the same derivation will be used with the computational elements based on Poisson's kernel. However, for the sake of clarity here the use is made of cylindrical coordinates rather than complex variable used in section 3.2.2.

Consider a cluster of N particles located at points $\mathbf{x}_i = (r_i, \theta_i)$ with magnitudes κ_i bounded by a disk of radius a centered at the origin. Let $\phi(r_j, \theta_j)$ be the potential in two dimensions induced by the collection. since $\phi(r_j, \theta_j)$ is a harmonic function outside the disk we can use Poisson's formula (with a log term to satisfy circulation requirements) to represent it. A detailed derivation of the expression for the potential $\phi(r_j, \theta_j)$ is given in the Appendix A.4.

$$\begin{aligned}\phi(r,\theta) \approx\ & \kappa\log(r) + \frac{1}{2\pi}\sum_{i=1}^{K} f(s_i) \times \\ & \left[\frac{1-\left(\frac{a}{r}\right)^2 - 2\left(\frac{a}{r}\right)^{M+1}\cos(M+1)(\theta+s_i)}{1-2\left(\frac{a}{r}\right)\cos(\theta-s_i)+\left(\frac{a}{r}\right)^2}\right]h + \\ & \left[\frac{2\left(\frac{a}{r}\right)^{M+2}\cos(M(\theta-s_i))}{1-2\left(\frac{a}{r}\right)\cos(\theta-s_i)+\left(\frac{a}{r}\right)^2}\right]h \end{aligned} \tag{3.13}$$

In Anderson's FMM the equation (3.13) above is used to approximate the potential induced by N particles of strengths κ_i with $h = \frac{2\pi a}{K}$, $\kappa = \sum_{i=1}^{N} \kappa_i$ and $f(s_i) = \phi(a, s_i) - \kappa\log(a)$. This approximation involves the summing of the strengths of the particles in a disk to get the weight of the log term just the same as in the original FMM then followed by the evaluation of the potential induced by the particles (minus the log term) at equispaced points on the disk of radius a. To obtain the potential due to the cluster of particles in the disk we just need to add the log term and the sum as in (3.13).

It should be noted that expression (3.13) is almost exactly similar in form to the corresponding expression (3.6) that is used for the same purpose in the original FMM computation. If the number of particles contained in a disk is large compared to K (or p in the Greengard-Rokhlin FMM), a considerable saving is realized in using such approximations. Moreover, if the exact Fourier coefficients were used insteady of the discrete ones then the resulting approximation would be identical to a multipole expansion.

To complete the mathematical model of the Anderson's FMM we need a representation of the potential inside a given region. This can be derived in the same logic as (3.13)

$$\begin{aligned}\phi(r,\theta) \approx\ & \frac{1}{2\pi}\sum_{i=1}^{K} f(s_i) \times \\ & \left[\frac{1-\left(\frac{r}{a}\right)^2 - 2\left(\frac{r}{a}\right)^{M+1}\cos(M+1)(\theta+s_i)}{1-2\left(\frac{r}{a}\right)\cos(\theta-s_i)+\left(\frac{r}{a}\right)^2}\right]h + \end{aligned}$$

$$\left[\frac{2\left(\frac{r}{a}\right)^{M+2}\cos(M(\theta - s_i))}{1 - 2\left(\frac{r}{a}\right)\cos(\theta - s_i) + \left(\frac{r}{a}\right)^2}\right]h \tag{3.14}$$

where (r, θ) is a point in a disk and $f(s_i)$ is the value of potential induced by particles outside the disk.

To realize the summing up of the contributions of elements inside a disk all needed is to evaluate the element's outer ring approximations(3.13) at the integration points of a single outer ring approximation for the point where the sum is to take place, in the original multipole method this would require shifting of the origins of the multipoles. This transformation is required in the upward pass of the FMM algorithm.

In the downward pass of the FMM we use the expression(3.14) in a similar way to the one for the upward pass. Maybe the most motivating feature of the Anderson's FMM is the close similarity and almost duality in the combining operations required in the FMM algorithm, moreover, this applies to the transition from 2-dimensional to 3-dimensional problems. This makes the programming of this method relatively easier than that of the original Greengard-Rokhlin FMM.

The error rate of the Anderson's FMM model using $K = 2M+1$ integration points can be expected be similar to that in the Greengard-Rokhlin FMM method using M terms of a multipole expansion. This prediction arise from the close relationship between the two representations[3].

3.3.4 The Fast Multipole Method for the Scattering Equation

This variant is the actual basis of our algorithm, and in fact it is more or less a direct descendant of the Anderson's FMM without multipoles. The element of computation for this model is subscatter that result when a scatter is devided. This subscatter is modelled as a pair of functions, the basis function $b_n(\mathbf{r})$ and the testing function $t_m(\mathbf{r})$ used to represent the induced current in the element

and strength of the electromagnetic field at the element respectively.

Given an object of surface S divided into elements $b_n(\mathbf{r})$ as described above and with a set of N unknowns a_n, we see that the total current induced on the object is given by

$$J(\mathbf{r}) = \sum_{i=1}^{N} b_n(\mathbf{r}_i) a_n \tag{3.15}$$

and hence, according to the Electromagnetic Scattering Theory when an E_x polarized wave is incident on the object the electric field scattered by the object, E_x^{Sc}, is given by

$$E_x^{Sc}(\mathbf{r}_j) = -\frac{k\eta}{4} \int_S G(\mathbf{r}_j, \mathbf{r}) J(\mathbf{r}) d\mathbf{r} \tag{3.16}$$

where $G(\mathbf{r}_j, \mathbf{r}) = H_0^{(1)}(k\,|\mathbf{r}_j - \mathbf{r}|)$. On the light of this substitution and change of summation and integration operations we arrive at the following equation

$$E_x^{Sc}(\mathbf{r}_j) = -\frac{k\eta}{4} \sum_{n=1}^{N} \int_S H_0^{(1)}(k\,|\mathbf{r}_j - \mathbf{r}|)\, b_n(\mathbf{r}) a_n d\mathbf{r} \tag{3.17}$$

Moreover, using the testing elements $t_j(\mathbf{r}_j)$ divided as above when an E_x polarized wave is incident on the object we have the electric field induced in the object E_x^{In} given by

$$e_m = \int_S E_x^{In}(\mathbf{r}_j) t_m(\mathbf{r}_j) d\mathbf{r}_j \tag{3.18}$$

However, whenever an E_x polarized wave is incident on the object, from the boundary conditions of the Laplace equation, the tangential components of the induced and scattered electric field on the objects surface must cancel each other, therefore we have

$$E_x^{Sc}(\mathbf{r}_j) = -E_x^{In}(\mathbf{r}_j) \qquad \forall \mathbf{r}_j \in S \tag{3.19}$$

Combining the above equations (3.15) through (3.19) we obtain the following relationship

$$e_m = \frac{k\eta}{4} \sum_{n=1}^{N} \int_S \left[\int_S H_0^{(1)}(k\,|\mathbf{r}_j - \mathbf{r}|)\, b_n(\mathbf{r}) a_n d\mathbf{r} \right] t_m(\mathbf{r}_j) d\mathbf{r}_j \tag{3.20}$$

$$m = 1, \ldots, N \tag{3.21}$$

Actually, the above equation is expression of the matrix equation $Z \times I = V$ as concluded in the mathematical modelling of the electromagnetic scattering problem in Chapter 2 where

$$z_{mn} = \int_S \left[\int_S H_0^{(1)} \left(k \left| \mathbf{r}_j - \mathbf{r} \right| \right) b_n(\mathbf{r}) d\mathbf{r} \right] t_m(\mathbf{r}_j) d\mathbf{r}_j \tag{3.22}$$

$$i_n = a_n \tag{3.23}$$

$$v_m = e_m \tag{3.24}$$

The expression (3.22) which is almost exactly similar in form to equation(A.11 in the appendix) used in derivation of the Anderson's FMM transactions motivated the use of the fast multipole for speeding up the sum (3.21). So far there seem no improvement on the sum as equation (3.22) actually indicates we shall have to determine the sum using $O(N^2)$ operations after all. However, the speedup is hidden in the fact that we do not perform these computations directly as shown on equation (3.22) insteady special addition theorems given in [11] and [20]are used for the purpose.

Chapter 4

The Sequential Implementation of the Scattering Solution

4.1 Introduction

Generally, as is shown in Chapter 2, the mathematical modelling of the electromagnetic scattering problem result into a matrix equation of the form $Z \times I = V$. For a real life problem the number of unknowns, N, required for an accurate modelling can be prohibitively very large. This is because solving the equation directly by factorization of the dense matrix Z or by the use of the Gaussian theorem requires $O(N^3)$ operations while an iterative solution would require a matrix vector multiplication operation that requires $O(N^2)$ per iteration.

The solution of the electromagnetic scattering problem consists of three major parts. These parts are the iterative scheme, the preconditioning of the iterative scheme, and the matrix vector multiplication. Preconditioning is necessary for the iterative schemes that need it and normally consist of partial factorization of the interaction matrix and a triangular solver used within the iterations.

The solution developed in this work is an iterative strategy built around the

Conjugate Gradient Square (CGS) method. In iterative methods in general, the aim is to improve the complexity of the operations within a single iteration and speeding the convergence rate of the iterative routine. More specifically, the improvement efforts are usually targetted at making the matrix-vector operation more efficient as it is the most expensive of all operations and the bottleneck in an iteration. In fact it is the sole determinant of the complexity of an iteration. Moreover, especially when the matrix resulting from modelling of a particular physical problem is inherently poorly conditioned, various types of preconditioning has to be used to reduce the number of iterations required for the iterative routine to reach convergence.

This work started with a working Fortran 77 FMM code [11] that solves the electromagnetic scattering problem in sequential mode. The code was closely examined, its operations reorganized, and its data structures tuned for a better sequential performance. One version with the new modifications and another without was implemented on the Parystec CC 24. Based on the single program multiple data (SPMD) paradigm a parallel algorithm was designed and implemented again on the Parystec. The objectives of this parallelization was to make use of parallel computing capability to enhance the performance of the solution and hence enable solution of even larger problem sizes.

In the sections that follow a description of the original implementation is given and both the algorithm and its implementation are analysed for possible factors that could be improved to enhance them. Then the modified algorithm and its implementation is also described to highlight its variation from the original design.

Moreover, to ensure satisfactory convergence rate and at a reasonable memory and time complexity, a block version of matrix LU factorization together with a block version triangular solver was developed to deal with the preconditioning of the current solution between iterations.

4.2 Common Features of the FMM Implementations

In this work two sequential implementations and one parallel implementation of the fast multipole based solution to the electromagnetic scattering problem are discussed. The first sequential implementation is the original code that was adopted from a previous work [11] and the second is our modification of the first. The parallel implementation is the one that we developed in this work. In the paragraphs that follow the common features of all these implementations are explained as a basis for the discussions that follow.

1. They seek to solve the equation $Z \times I = V$ explained in the mathematical model of the electromagnetic scattering. From the nature of this particular application Z is complex and symmetric although not guaranteed to be positive definite.

2. They use an iterative strategy, is this case CGS although also GMRES could be used for the purpose. Moreover, if a particular problem set is found to yield a positive definite Z, then there is even a wide range of possible iterative schemes that could be used [7] that has less complexity than CGS in terms of number of sp$M \times V$, preconditioner and other operations per a single iteration.

3. The matrix Z is divided into a sum of two matrices $Z = Z_{FF} + Z_{NF}$ depending on the separations between interacting elements. Z_{FF} contains the far field interactions between elements that can be summed using the FMM with the required accuracy while Z_{NF} contain the nearfield interactions of matrix Z. Z_{NF} is generally sparse and is summed using a direct summing method.

4. To improve the convergence rate of the iterative strategy the Z_{NF} is factorized and the resulting factors are used in preconditioning the intermediate vectors resulting in the iterative process. For preconditioning a

triangular solver is used which, in case of the CGS routine, is called twice per iteration.

5. The matrix vector multiplication routine owns the Z matrix and takes as input the multiplicant vector $\mathbf{x}$ and returns the product vector $\mathbf{b} = Z \times \mathbf{x}$. However, since Z is actually a sum of Z_{FF} and Z_{NF} the actually computation is

$$\mathbf{b} = Z_{FF} \times \mathbf{x} + Z_{NF} \times \mathbf{x} \tag{4.1}$$

and $\mathbf{b}_{NF} = Z_{NF} \times \mathbf{x}$ is computed directly while $\mathbf{b}_{FF} = Z_{FF} \times \mathbf{x}$ is determined using a fast summation method based on the FMM.

4.3 Speeding up the Matrix Vector Multiplication

The matrix vector multiplication, $b_j = \sum g_{ji} a_i$, is the bottleneck in the speed of any iterative strategy and in particular the Conjugate Gradient Squared (CGS) method that is used in this solution. In the CGS the matrix vector multiplication is used twice. Because of the special property of the matrix-vector multiplication mentioned above, a large number of methods have been developed that aim at improving the complexity of the matrix-vector multiplication operation. Some of these efforts aim to exploit the mathematical properties of the matrix such as sparsity, positive definiteness, shape and symmetry; while others concentrate on exploiting the physical nature of the process that is modelled by the matrix. However, in some instances, the problem is examined for conformity to a well-known mathematical model and hence uses the established faster solutions already documented for the model. This work progresses along the latter alternative.

The algorithm used in this work is a modification of the original Greengard-Rokhlin Fast Multipole Method by Anderson [3] and it relies on the translation of scattering centers to speed up their summations. First a scatter is divided into many subscatters and instead of trivially computing the N^2 floating point

operations the strategy reduces the count down to asymptotic complexity of $N^{1.5}$.

In [21], the electromagnetic scattering problem has been shown to conform to the N-body problem. The Fast Multipole Method that has been shown to work efficiently in fast summing interaction elements of an N-body is adopted, modified and applied to speed up the matrix-vector multiplication operation which is the most expensive operation of the iterative solution [11]. Actually, in the first phase of this work a working sequential code of this solution from a previous work was examined and improved to better memory and time complexity by tuning the data structures used and rearranging the operations involved.

The sequential fast matrix-vector multiplication is then modified further for more parallelism so that a multiplicity of concurrently working processing nodes may be used to improve further the complexity of the operation. The final parallel algorithm is implemented in the Fortran language on a Parystec CC 24 node multicomputer.

4.4 The Original Sequential Implementation

The original sequential algorithm is based on the Fast Multipole Method algorithm adapted to the electromagnetic scattering problem. It uses an iterative scheme, the Conjugate Gradient Method (CGS) to solve the problem. In fact it concentrates on improving the matrix vector multiplication operation that is called twice in each loop in the iterative routine. Otherwise, for the part of preconditioning it uses the Sparse Library routines for factorization and triangular solution of the sparse nearfield matrix. Moreover, it uses a template of a preconditioned CGS (Figure 4.1) from the LAPACK collection of templates[7].

Important to note in the algorithm of Figure 4.1 is that the matrix-vector multiplication and the preconditioning solve are both called twice in one iteration. Since preconditioning was realized using a library function we will dwell

Compute $r^{(0)} = b - Ax^{(0)}$ using some initial guess $x^{(0)}$ and choose $\tilde{r} = r^{(0)}$
for i=1, 2, ...
$\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
if $\rho_{i-1} = 0$ **return** FAILURE
if $i = 1$
$u^{(1)} = r^{(0)}$
$p^{(1)} = u^{(1)}$
else
$\beta_{i-1} = \frac{\rho_{i-1}}{\rho_{i-2}}$
$u^{(i)} = r^{(i-1)} + \beta_{i-1} q^{(i-1)}$
$p^{(i)} = u^{(1)} + \beta_{i-1}(q^{(i-1)} + \beta_{i-1} p^{(i-1)})$

solve $M\hat{p} = p^{(i)}$
$\hat{v} = A\hat{p}$
$\alpha_i = \frac{\rho_{i-1}}{\tilde{r}^T \hat{v}}$
$q^{(i)} = u^{(i)} - \alpha_i \hat{v}$
solve $M\hat{u} = u^{(i)} + q^{(i)}$
$x^{(i)} = x^{(i-1)} + \alpha_i \hat{u}$
$\hat{q} = A\hat{u}$
$r^{(i)} = r^{(i-1)} - \alpha_i \hat{q}$
check convergence and continue if necessary
end

Figure 4.1: The Conjugate Gradient Squared Algorithm.

most in discussing the implementation of the matrix vector multiplication routine as actually this is where the Fast Multipole Method concept is utilized to speedup the operation.

4.4.1 Features of the Original Implementation

The following are highlights of the original solution

1. Factorization and the triangular solution for preconditioning the iterative routine to improve the iterative strategy convergence properties are realized through the use of the Sparse library routines spFactor and spSolve respectively

2. Since the elements are indexed locally in the clusters, a global indexing scheme is used that gives a numbering to all elements using the element's cluster number and its local, index in the cluster

3. Z_{NF} is stored in a classical sparse matrix representation, that is, using three one dimensional arrays *value*, *row* and *column*. Albeit transparent to the user, the library routine spFactor allocates memory space to store the factors of Z_{NF} that it uses in preconditioning.

4. Z_{FF} is represented as a quadruplet of arrays that stand for the transformations required for the proceeding of the fast summation by the FMM. The basis and test functions are represented by two dimensional $N \times N_F$ arrays *Basis* and *Test*. The upward and downward passes transformations are represented by a two dimensional $N \times N_F$ array *Prop*. These two transformations require only one representation because they are conjugates of each other. The dimension N_F represents the discretization of the transformations for numerical computation. The translation of field effects between clusters is represented by a three dimensional $M \times M \times N_F$ array *Trans*. These bring up the amount of memory required to $(32N+16M^2)\times N_F$ bytes where M is the number of the clusters in the problem.

5. The $b_{NF} = Z_{NF} \times \mathbf{x}$ is made to progress the same order as the Z_{NF} is filled.

6. The sp$M \times V$ $\mathbf{b}_{FF} = Z_{FF} \times \mathbf{x}$ is performed according to the formulation of the FMM.

The original algorithm is given in Figure 4.2.

4.4.2 The Original Sequential Matrix Vector Multiplication Algorithm

The objective of this algorithm is that given a matrix Z and vector I, determine the product $v = Z \cdot i$, where Z is an $N \times N$ matrix and both v and i are size N vectors. Trivially, this involve O(N^2) operations and hence the FMM concept is used to reduce the complexity to O($N^{1.5}$) as shown in Chapter 3.

The input to the program is data about a certain set of subscatters (or equivalently strips) which is arranged into clusters and the geometrical properties and relationships about the cluster data. Before calling the matrix-vector multiplication routine the data is converted into a matrix consisting of interactions between the elements in the clusters. These interactions are classified into two forms depending on the amount of separation between them, the nearfield interaction and the far field interaction. The nearfield interaction is the one between elements in the same clusters and elements within clusters for which the separation is not enough to give the required accuracy if the FMM is used for their summation. The far field interaction is between elements in clusters that are separated enough for the FMM to be used to sum them up with the required accuracy.

In the matrix-vector multiplication the nearfield interaction is stored in a sparse matrix format and a direct sparse matrix - vector multiplication is performed between the near field matrix and the coefficient vector. Actually, the FMM algorithm is applied only to the far field interaction matrix.

```
for each cluster L do
    for each discretization point ip do
        initialize Q(ip, L) to zero
    for each element l in cluster L do
        get global index from array GI(L,l) let it be ig
        for each discretization point ip do
            Q(ip, L)=Q(ip, L)+Prop(ip,ig)*Basis(ip,ig)*x(ig)
{Here is the end of the upward pass}
for each cluster K do
    for each dicretization point ip do
        initialize R(ip, K) to zero
    for each cluster L do
        if K and L are near clusters then
            for each discretization point ip do
                R(ip, K)=R(ip, K)+Trans(ip, K, L)×Q(ip,L)
    for each element k in cluster K do
        for each discretization point ip do
            get global index from array GI(K, k) let it be ig
            initialize S(ip, ig) to zero
    for each element k in cluster K do
        get global index from array GI(K, k) let it be ig
        for each discretization point ip do
            S(ip, ig)=S(ip, ig) +conjg(Prop(ip, ig))×R(ip,K)
    for each element k in cluster K do
        get global index from array GI(K, k) let it be ig
        initialize sum to zero
        for each discretization point ip do
            sum = sum + Test(ip,ig)×S(ip, ig)×weight(ip)
```

Figure 4.2: The Original Sequential Matrix Vector Multiplication ($spM\times V$) Algorithm

4.5 Modifications to the Original Sequential Algorithm

Albeit the original purpose of our modifications to the sequential code was to have a much easily parallelizable algorithm, on examining the existing code carefully we realized several features of the existing implementation that could be improved for better performance in terms of time and space complexities of the algorithm. The main strategies that we used for improvement of the sequential code was to remove as much as possible floating point operations from the iteration loop body while utilizing the available cache in the best way possible. Theoretically, this would result into a constant change on the ultimate asymptotic complexity; practically, as exhibited from our experimental results, it gave rise to quite astonishing improvements in performance.

4.5.1 From Static to Dynamic Memory Management

An immediate limitation that we removed was lack of flexibility of space usage of the program. In the original version a large chunk was necessary to be allocated to the program before use, although only a small chunk of that space was actually used in the program. The main reason for this was absence of dynamic memory allocation in Fortran 77 standard. Initially when we were working on the Sun SparcStations we designed a C routine for this purpose and linked it with the other Fortran modules but later when we ported the implementation to the Parystec we converted the program into using the few Fortran 90 standard dynamic memory allocation routines that were available.

4.5.2 Cache Utilization

Cache utilization refers to reorganization of an algorithm so as to maximize running of the program at the top of the memory hierachy (Figure 6.3). This is realized by arranging the matrix data in form of interaction blocks. This was

used mainly in nearfield computation in which matrix blocks and input vector slices were read wholly at a time.

4.5.3 Reduction of Loop Computation

In the original implementation both the base/test function and the translation from cluster element to the cluster centre were represented by arrays. This necessitated two computations, first transforming the input vector by the base/test function and second transforming the result by the translation function to the centre. In the modified version, the basis function and the translation are fused into one transformation that takes the input vector directly to the cluster centre. Obviously, this led to good gains in perfomance of the loop.

4.5.4 Reduction of Indexing

Moreover, we noted multi-indexing of data, that is, use of structures that need more indexes for access such as arrays was also a significant factor in the speed of the algorithm. Changing the data representations from multidimensional arrays to single dimensional arrays; or in some cases actually designing an equivalent algorithm that used less indexing by utilizing temporary buffers also contributed to our improved efficiency. This is quite at par with well established theories about the working of computing machinery.

4.5.5 Changes on the Preconditioner

An interesting point that we noted was that it is not enough just to say that the nearfield matrix is sparse. Actually it is blockwise sparse while the blocks themselves were completely dense. Exploiting this observation also led to significant improvements both in the memory requirements and speed of the matrix vector multiplication operation. To utilize this fact we changed the data structures

that are used to represent the nearfield matrix in the computer from simply classical sparse matrix representation to the blockwise sparse representation. In this way we removed the need of excessive indexing of the elements and it became possible to use more optimized algorithms for dense matrix operations with the blocks

4.6 The Modified Sequential Implementation

4.6.1 Features of the Modified Implementation

The following are distictive features of the modified implementation

1. Preconditioning is realized using a blockwise sparse factorization routine and a triangular solver both developed in this work.

2. Elements are indexed globally using only one index compared from the original design that uses two indices for the purpose.

3. The nearfield interaction Z_{NF} is stored as a list of interaction blocks, indexed by a block number that is directly derived from the cluster numbers of the interacting clusters. Although the blocks are sparse, each block constitute a dense matrix, so within the block standard dense matrix algorithms can be used. Actually, Z_{NF} is represented in the algorithm by a set of three lists : the list of NF blocks Blk(i) indexed by block number, the list stK(k) that contain starting index number of each row block indexed by block rows, and the list jL(i) that contain the block column number of a block indexed by block number. This representation was decided so as to simplify the complexity of multiplication and factorization of the Z_{NF}.

4. Z_{FF} matrix is represented by a trio of sparse matrices representing the three basic transformations involved in the Fast Multipole Method summation. These are the transformation at the base cluster represented by

the B matrix, the transformation between cluster centres represented by the H matrix and the transformation at the testing cluster represented by the T matrix. Actually, the far field interaction determination is motivated by the physical conceptualization of the electromagnetic scattering problem and its conformity to the generic N-body model.

4.6.2 The Modified Sequential $spM\times V$ Algorithm

Since this algorithm has to work within an iterative routine its interface has to be designed so as to conform to the interface requirements of the iterative routine. The algorithm seeks to determine the product $\vec{\mathbf{b}} = Z \times \vec{\mathbf{x}}$. It owns the matrix Z and takes as input the multiplicand $\vec{\mathbf{x}}$ vector and returns the product vector $\vec{\mathbf{b}}$. Actually, the Z matrix exist as a sum of two matrices, that is $Z = Z_{FF} + Z_{NF}$, and this has influence to the multiplication process as described below.

The matrix in Figure 4.1 is an example of the matrix resulting from a set of linear strips that are grouped into 12 clusters. In the Z matrix of Figure 4.1, the blocks marked NF contain nearfield interaction submatrices. It should be noted that although these blocks are themselves sparse, that is three to five blocks per row, they are in fact dense blocks. An interesting point to note is that for a particular problem the number of nearfield blocks in a particular block row is limited to a small number which makes the rowwise distribution of blocks a good candidate for the purpose of load balancing. The rest of the blocks constitute the Z_{FF} matrix.

The input to the algorithm is a mathematical model of the scattering elements grouped into clusters and the output is the sum of effects of all other elements on each element. This model is a triangular function for each scattering element. Computationally, the input is modelled as a list of 3-tuples representing the three vertices for each scattering element. In our implementation the list in represented by a dynamically allocated $N \times 3$ array. Therefore

NF		NF		NF							NF
	NF	NF	NF				NF			NF	
NF	NF	NF	NF	NF							
	NF	NF	NF	NF	NF						
NF		NF	NF	NF	NF	NF					
			NF	NF	NF	NF					NF
				NF	NF	NF	NF	NF			
	NF					NF	NF	NF	NF		
						NF	NF	NF	NF	NF	
							NF	NF	NF	NF	NF
	NF							NF	NF	NF	NF
NF					NF				NF	NF	NF

Table 4.1: The Structure of Interaction Matrix for a 12 Cluster Problem

actually what this algorithm does is

$$\begin{aligned} \vec{\mathbf{b}} &= (Z_{FF} + Z_{NF}) \times \vec{\mathbf{x}} \\ &= Z_{FF} \times \vec{\mathbf{x}} + Z_{NF} \times \vec{\mathbf{x}} \\ &= \vec{\mathbf{b}}_{FF} + \vec{\mathbf{b}}_{NF}. \end{aligned}$$

The algorithm uses direct sparse matrix vector multiplication to determine $\vec{\mathbf{b}}_{NF}$ and the FMM to determine $\vec{\mathbf{b}}_{FF}$. In the algorithm the matrix block B_{KL} represent interaction between cluster K and cluster L. The concise form of the algorithm is given in Figure 4.6.2

; Determine $\vec{\mathbf{b}}_{NF}$ using sparse matrix multiplication method
; $\vec{\mathbf{b}}_K$ and $\vec{\mathbf{x}}_K$ are slices of $\vec{\mathbf{b}}$ and $\vec{\mathbf{x}}$ respectively.
for each cluster K **do**
for each NF block $Z_{[I,J]}$ such that $I = K$ **do**
$\vec{\mathbf{b}}_K$ = Blas3Mult($Z_{[I,J]}, \vec{\mathbf{x}}_K$)
; Determine $\vec{\mathbf{b}}_{FF}$ using the Fast Multipole Method
for each cluster L **do**
initialize to $Q(iQ)$ zero for all iQ
for every element l in L **do**
get the global index of l into ig
$Q(iQ) = Q(iQ) + B(iB) \times x(ig)$
for each cluster K **do**
for each discretization point ip **do**
initialize $R(ip)$ to zero
for each cluster L such that Far(K,L) **do**
for each discretization point ip **do**
$R(ip) = R(ip) + H(iH) \times Q(iQ)$
for each discretization point ip **do**
for each element k in K **do**
$\vec{\mathbf{b}}(\mathbf{ig}) = \mathbf{b}(\mathbf{ig}) + T(iT) * R(ip)$

Figure 4.3: The Modified Sequential Matrix Vector Multiplication ($spM\times V$) Algorithm

Chapter 5

Data Organization for the $spM\times V$ Algorithm

This chapter is about the organization of data required by the $spM\times V$ algorithm. In the sections that follow, we shall investigate the various data partitioning possibilities that are generally used in parallel algorithms and describe the data distribution strategy that we use in this application. Moreover, the advantages and disadvantages of using these partitioning strategy are examined, both in general context and for our particular application, to explain our data partitioning option.

5.1 Data Parallelism of the $spM\times V$ Algorithm

Much more than for the computation of $\vec{\mathbf{b}}_{FF}$, the computation of $\vec{\mathbf{b}}_{NF}$ depends very much on the distribution of data, and this will be clear after we examine various options of data distributions and their effects on the parallelization. In the following subsections we shall highlight various options that are possible for the data organization and distribution of the Z_{NF} among the participating processors.

5.1.1 All nearfield blocks kept on a single processor

At first sight this may seem to be a nice option, that is, because NF blocks are almost insignificantly small in size that the FF blocks for a general set of strips problem it may seem attractive to have them just in a single processor and let it quietly determine the $\vec{\mathbf{b}}_{NF}$ without communicating with any other processor. However, this option has several ugly setbacks as explained briefly below

1. Prior to determining $\vec{\mathbf{b}}_{NF}$ it needs to receive $\vec{\mathbf{x}}$ held by all other processors, that is, it will have to execute a global collect communication operation every time the $spM\times V$ is called. In this way it will be a bottleneck, something highly undesirable for a parallel computation system.

2. Since at the end, all the processors need to determine the sum $\vec{\mathbf{b}} = \vec{\mathbf{b}}_{NF} + \vec{\mathbf{b}}_{FF}$ it will be necessary for the processor that determine $\vec{\mathbf{b}}_{NF}$ to execute a global broadcast communication operation so as to deliver its partial product. This also will result in a bottleneck.

3. Other lesser important disadvantages include asymmetry that will result in the algorithm.

5.1.2 The Block Checkerboard Partitioning

Checkerboard partitioning refers to the case when a matrix is conceptually divided into smaller square or rectangular blocks or submatrices that are distributed among the processors [23]. A checkerboard partitioning splits both the rows and the columns of the matrix so that no processor is assigned a complete row or column. This partitioning is interesting because the matrix computation can be divided into more processors than in stripe partitioning and hence it is more scalable [17]. For example, theoretically, we can partition an $N \times N$ matrix among a maximum of N^2 processors using checkerboard partitioning while we cannot use more that N processors with striping.

P_1	P_1	P_3	P_3
P_1	P_1	P_3	P_3
P_1	P_1	P_3	P_3
P_2	P_2	P_4	P_4
P_2	P_2	P_4	P_4
P_2	P_2	P_4	P_4

Table 5.1: The Checkerboard Partitioning

However, the idea of allocating more processors than there is clusters has its serious setbacks. The clustering process constrains the number of elements to vary proportional to the square of the number of clusters. Because of this, if the number of clusters is small enough to be less than the number of processors the number of elements in a cluster will be much less to be shared by more than one processor. Moreover, the farfield computation is naturally parallel when the actual cluster, rather than matrix block data is allocated to processors. If we allocate more processors than the number of clusters, we either have to have some processors with nil clusters and hence difficulty in load balancing; or we have to devise another scheme that will involve more communication.

In this partitioning strategy every processor also own a slice of the $\vec{\mathbf{x}}$ vector that it needs for computing its partial $\vec{\mathbf{b}}$ vector slice. Every processor concurrently computes a partial slice of the product vector, that is the processor that contains matrix block $\mathbf{A}_{ij}$ and vector slice $\vec{\mathbf{x}}_j$ performs the following computation

$$\vec{\mathbf{b}}_{ij} = \mathbf{A}_{ij}\vec{\mathbf{x}}_j.$$

However, after this computation, a single processor in every row has to perform a global gather communication operation so as to sum up the partial contributions of the slice that it owns, assuming a ring topology this amounts to a requirement of $O(\sqrt{\mathcal{P}})$ where $\mathcal{P}$ is the number of participating processors. Moreover, a single processor in every column has to execute a columnwide multicast communication operation at the asymptotic cost of $O(\sqrt{\mathcal{P}})$ on a ring topology.

Unfortunately the attractively low communication overhead offered by this

P_1	P_2	P_3	P_3
P_1	P_2	P_3	P_3
P_1	P_2	P_3	P_3
P_1	P_2	P_3	P_3
P_1	P_2	P_3	P_3
P_1	P_2	P_3	P_3

Table 5.2: The Columnwise Partitioning

data partitioning strategy is generally applicable to dense matrices. Since in the $spM\times V$ the actual matrix for which we do direct matrix vector multiplication is the nearfield interaction submatrix Z_{NF}, which is strictly sparse. This partitioning scheme has a serious shortback in the sense that load balancing becomes complicated because most of the checkerboard blocks are actually non-existent.

5.1.3 The Block Columnwise Partitioning

This refers to the distribution of data in which all the blocks $Z_{[IJ]}$ for which $J = L$ are kept in the same processor as in Figure 5.2. This has advantage that if we let the processor also own the slice of $\vec{\mathbf{x}}$ that is related with cluster L, $\vec{\mathbf{x}}_L$ then the algorithm does not need precommunication since every processor will have the slice of $\vec{\mathbf{x}}$ that it needs for its multiplication. However, post communication will be necessary since what a processor will determine will not be actually the $\vec{\mathbf{b}}_{NF_K}$ but a partial contribution of it, that is $\vec{\mathbf{b}}_{NF_{KL}}$. Hence every processor will have to call its personalized global gather to get all the contributions of other processors for the slice of $\vec{\mathbf{b}}_{NF}$ that it owns and then perform the following sum

$$\vec{\mathbf{b}}_{NF_K} = \sum_{\forall L, near(K,L)} \vec{\mathbf{b}}_{NF_{KL}}. \tag{5.1}$$

However, due to the fact that the Z_{NF} is sparse, if we use the standard gather for a particular row, most of the processors shall have to communicate zero arrays leading to unnecessary communication overheads.

P_1	P_1	P_1	P_1
P_1	P_1	P_1	P_1
P_2	P_2	P_2	P_2
P_2	P_2	P_2	P_2
P_3	P_3	P_3	P_3
P_4	P_4	P_4	P_4

Table 5.3: The Rowwise Partitioning

5.1.4 The Block Rowwise Partitioning

This refers to the distribution of data in which all the blocks $Z_{[IJ]}$ for which $I = K$ are kept in the same processor as in figure 5.3. This has advantage that if we let the processor also own the slice of $\vec{\mathbf{b}}$ that is related with cluster K, $\vec{\mathbf{b}}_K$ then the algorithm does not need postcommunication since every processor will be able to determine the slice of $\vec{\mathbf{b}}$ that it own. Generally, precommunication will be necessary because a processor will need the slices of $\vec{\mathbf{x}}$ from some of other processors.

In a general context, the rowwise and columnwise partitioning schemes offer similar communication requirements. Postcommunication and precommunication for columnwise and rowwise partitioning respectively. The similarity is stressed even more by the fact that the matrix is symmetry. Moreover, since for a particular problem the number of nearfield cluster to a particular is a constant these forms of partitioning offer a easier load balancing strategy than for the checkerboard partitioning for the nearfield computation.

5.2 Data Partitioning

Data partitioning refers to the distribution of data required for a parallel computation into the processing nodes that take part in the computation. This is one the most important problems that one must solve in order to use parallel computers. The objectives of data partitioning are to minimize communication requirement and ensure good load balance across the participating processors.

In the parallel solution of the electromagnetic scattering problem developed in this work there are two independent computations, the nearfield and the far field computation. Various solutions are found in the parallel computing literature that seek to solve the data partitioning problem. The solution used here for the nearfield computation makes use of the graph theory to model the computation. The computation is viewed as a graph in which vertices represent computation and edges represent communication. Mapping means assigning each node to a processors which is equivalent to partitioning the graph nodes and referring to vertices in a partition as being allocated to the same processor.

In the case of far field computation the basic data structure involved is the set of clusters. This is because the FF computation is realized through a continual application of transformations two of them needing exclusively local cluster data without any need of communication; and the third transformation, that is the summing of contributions from other cluster centers at the center of a testing cluster requiring a global summing which is done for each processors in parallel.

Nearfield computation involves a sparse matrix multiplication to a dense vector. The sparse matrix is made of nearfield interaction blocks $Z_{[KL]}$ between near clusters K and L and the dense vector is a concatenation of the testing clusters' vectors. In case of NF computation data partitioning results into distribution of the testing vectors and nearfield interacting blocks into participating processors. However, for the NF computation, a processors having the NF block $Z_{[KL]}$ need testing vector $\vec{\mathbf{x}}_{\mathbf{L}}$ from its owning processor to perform the computation

$$\vec{\mathbf{b}}_{\mathbf{K}} = \mathbf{Z}_{[\mathbf{KL}]} \times \vec{\mathbf{x}}_{\mathbf{L}} \tag{5.2}$$

and on completion it need to send the product vector $\vec{\mathbf{b}}_{\mathbf{K}}$ to its owning processor.

In rowwise decomposition scheme, processors need to get non-local cluster data $\vec{\mathbf{x}}_{\mathbf{L}}$ prior to the computation 5.2 above. After receiving these $\vec{\mathbf{x}}$ slices the processors can proceed with the computation concurrently. Hence load balancing problem can be reduced to a simple division of the cost of computation 5.2

above and therefore be modelled as a number partitioning problem. However, the objective of data partitioning is not limited to load balancing alone, the communication volume that results from the partitioning also have to taken into consideration. Unfortunately, the communication requirements scales up with increasing problem size and the minimization of the communication cost while at the same time maintaining load balance is a domain decomposition problem [5].

The key to a successful domain decomposition is to maximize data locality with a balanced load across the processors taking part in the computation. In the domain decomposition applied here the Z_{NF} matrix is modelled as a graph which is passed to the METIS graph partitioning tool [16] for partitioning. In the following section we discuss the graph model that is used by METIS for decomposition of the Z_{NF} matrix.

5.2.1 Graph Theoretic Modelling of the Z_{NF} Matrix

In this partitioning scheme the sparse matrix Z_{NF} is represented as an undirected graph $G = (V, E)$. The vertices V are the rows of matrix Z_{NF} and there exist an edge $e_{KL} \in E$ between any two vertices v_K and v_L if and only if there is a pair cluster K and cluster L that interact with each other in a nearfield way. Therefore, the adjacency list of a vertex v_K represent all the clusters that interact with cluster K in a nearfield way.

The weight w_{KL} of a node v_L in an adjacency list of node v_K represent the cost involved in computing the expression 5.2 given availability of $\vec{\mathbf{x}}_\mathbf{L}$. The load that is sought to be balanced by the METIS graph partition tool is the sum $\sum_{L\,:Near(K,L)} w_{KL}$ where v_K is allocated to p. The vertices are mapped to processors in such a way that each processor's sum is approximately the same.

The weight c_{KL} of an edge $e_{KL} \in E$ is the cost of communication required to send the vector $\vec{\mathbf{x}}_\mathbf{L}$ from the processor owning cluster L to the one owning cluster K if they are different. After partitioning an edge is said to be cut if

its pair of vertices are in two different partitions, otherwise it is uncut. The cut size refers to the sum of costs of cut edges. As a result, the objective of partitioning is to divide a graph into a multitude of parts so that the cutsize is minimized.

5.2.2 Implementation Considerations

To make use of the METIS graph partitioning tool, an algorithm was designed and implemented in Fortran. This program take as input the cluster data and by using the electromagnetic relationships between the cluster constructs the blocked matrix. The program then models the matrix as a graph that represent the nearfield block matrix resulting from cluster data. The resulting graph is then output in a plain text file in the format required by METIS as described below.

The METIS graph partitioning tool takes a graph stored in a plain text file and the number of required partitions as an input. A graph $G = (V, E)$ with n vertices and m edges is stored in a plain text file that contains $n+1$ lines. The first line contains information about the size and the type of the graph while the remaining n lines contain information for each vertex of G. The first line contains two or three integers. The first integer is the number of vertices, the second integer is the number of edges and the third integer describes the type of the input graph. This third integer is nonexistent, 1, or 11 if the input graph has all its nodes and edges having the same weight, only edges with unequal weights, or both edges and vertices have unequal weights respectively.

In this implementation both the edges and vertices may have different weights. A vertex represents a set of blocks that contribute together as basis function of a single testing cluster. Moreover, the vertex represents ownership of the testing cluster data. The weight of a vertex is the amount of computation of type (5.2) to be performed by the processor that contains the blocks. An edge exists between a pair vertices if ni the construction of a block in one set cluster data from the other set is required. Because of the symmetrical

properties of the matrix this property is reflexive and hence the edges are unidirected. The weight of an edge is the sum of the cluster data that need to be exchanged.

Chapter 6

The Parallel Solution to the Scattering Problem

In this chapter a discussion is given of the parallel algorithms that were developed in this work to solve the electromagnetic scattering problem. The algorithms developed as part of this solution are designed based on the generic data parallel strategy. Data paralellism is a model of parallel computing in which a single operation is applied to all elements of a data structure simultaneously [25]. Actually, the main outcome of this work consists of two major algorithms, that is, the parallel matrix vector multiplication algorithm based on the Fast Multipole Method; and the blockwise sparse factorization algorithm that is suited to problems which can be modelled using the N-body concept.

This work was done in three stages. In the first stage an existing sequential matrix vector multiplication algorithm was explored and modified so as to have better timing and space performance. To exploit the symmetrical and blockwise sparse properties of the nearfield matrix involved in the second stage a blockwise sparse factorization was developed to be used in preconditioning of the solution. In the third stage the parallel matrix vector multiplication algorithm was designed based on the modified sequential algorithm.

The modified sequential $spM\times V$ algorithm is described in Chapter 4 together with the original sequential algorithm. The modified sequential algorithm is the actual basis of the parallel $spM\times V$ algorithm developed in this work. Moreover, although the parallel algorithm is described here, the data distribution strategy used is described separately in Chapter 5 to avoid overcrowding in this chapter.

6.1 Parallelization of the $spM\times V$ Routine

Essentially, a parallel computing system constitutes a set of processing units joined together into a particular graphical formation by communication links. The objective of parallel computation is to distribute the data structures in a way that maximizes the number of tasks that can be done concurrently while at the same time minimizing the communication required for the completion of the task. Load balancing, that is, data distribution into processors evenly in such a way that every processor has exactly the same amount of work to do is also of paramount importance in a parallel computing scheme. In our particular application, since the cluster sizes are more or less uniform in the sense that the number of scatter elements in a cluster are roughly equal, the load balance criteria can be realized through balanced allocation of clusters to processors. The description of the data distribution strategy adopted in designing this solution is found in Chapter 5. However, the organization of data for the above purpose depends heavily on the nature of the algorithm to be implemented and topology of the underlying parallel machine on which the algorithm is to be implemented. In the following subsection an examination of the system upon which the algorithm is implemented, that is the Parystec CC 24 multicomputer running Extended Parallel Unix operating system, is examined in sufficient detail to provide the reasons for our choice of the parallelization strategy. Then the parallel $spM\times V$ algorithm developed in this work is described in detail to show why we should expect our parallelization scheme to produce good results.

6.2 The Parystec Cognitive Computer CC-24

6.2.1 Physical Topology of the Parystec CC-24

The Parystec CC-24 is a 24 node multicomputer system with each of its processors connected to all other processors by an interconnection network. The linking element consists of a hardware router which routes the data streams to target processors. The transfer time per data is constant and independent of the respective source or target processor [1]. It follows that physical topologies are not of significance, in fact there exists a virtual topology library package that offers various primitives so that by using them a user can create different topologies by simply calling the routine to do so.

The underlying operating system in the Parystec is called the Extended Parallel UNIX (EPX). The EPX supports the normal functions offered by the UNIX operating system and also some message passing facilities that may be used by a parallel program. The system supports creation of virtual topologies. The virtual topology is a set of nodes joined in a particular graphical format through virtual links. Virtual links enable most efficient possible communication between a pair of processors using one or more physical links of the machine. In this application two types of topologies are created. A ring topology is created for the global sum operation used in computing the dot products and second norm of a vector distributed into the participating processors. Another random topology is created to faciltate efficient swapping of vectors required for the nearfield computation.

The ability of the EPX, that is the operating system that runs on the Parystec, of creating topologies 'on the fly' has been a major factor in design of this program. Using this facility we could create any virtual topology that we think to be best suited to our task. We chose the ring topology because of its efficient expand communication.

6.2.2 Communication Primitives of the Parystec CC-24

The Parystec CC-24 system support the Single Program Multiple Data (SPMD) paradigm of parallel computing. The SPMD programming type refers the scenario in which a single program is loaded onto each of processors that participate in the algorithm, and according to the processor number, defined by its conceptual location in the system, determines which particular computation should be done by each individual processor. In this way the SPMD paradigm shares both of the features of both the SIMD and MIMD paradigms. For a rather more detailed discussion of the SIMD and MIMD paradigms, the reader is advised to refer to [25], and [17].

Generally, there are two methods of message passing in the MIMD distributed memory machines, these are called the *Blocking* and *Non-Blocking* communications. *Blocking* describes the action of **send** or **receive** which waits until the function has been executed successfully. The process is suspended until the operation is completed and this forms an implicit synchronization for the involved processors since they have to coincide for a finite time for the operation. On the other hand *Non-Blocking* send initiates the **sending** of data and continues with its computation and a *Non-Blocking* **receive** just provide a memory location where the incoming data will be stored and proceed with its computation, they both leave the underlying operating system to take care of communication. The EPX supports both of these through function calls `PX_Send` and `PX_Recv` for *Blocking* **send** and **receive** respectively. For the *Non-Blocking* **send** and **receive** there are `PX_ASend` `PX_ARecv` function calls respectively [1]. In another much used terminology *Blocking* and *Non-Blocking* are termed *Synchronous* and *Asynchronous* Communication respectively.

6.3 The Parallel $spM\times V$ Algorithm

In essence, a parallel program is a mixture of computation and communication. The communication is necessary so as to avail data wherever and whenever

required for a specific computation to take place. However, communication involves use of time which is an overhead to the computation time and hence contribute negatively into the efficiency of the parallel program as a whole. Therefore minimization of the amount of communication requirement is a key factor to be considered in the design for any parallel algorithm. For minimization of communication time both the topology of the parallel system and the mode of distribution of data among the processing nodes of the system is important. Topology of a parallel system refers to the graph which indicate how the processing elements of the system are linked. In this section we present the kind of computation and the type of communication required by the $spM\times V$ algorithm.

The $spM\times V$ algorithm is divided into two main parts, the $\vec{\mathbf{b}}_{NF}$ and the $\vec{\mathbf{b}}_{FF}$ determination. These two tasks can be achieved in any order as the result is the sum of the two which is a commutative operation. Our modified fast field computation algorithm is motivated by the physical interpretation of the N-body summation using the FMM while the nearfield computation involves a conventional sparse matrix vector multiplication.

Computation of $\vec{\mathbf{b}}_{FF}$ is based around the concept of FMM [11] [14]. As observable from the algorithm 6.1, the FMM algorithm has two stages. In the first stage, called the upward pass, the contributions of all the elements in each cluster are gathered summed at the cluster centre using a translation transaction applied on every element of the cluster. This sum is Q as shown in algorithm 6.1. Obviously, this needs only the data within a cluster, and, assuming that no single cluster may be shared by two processors, this can go on concurrently at every processor taking part in the solution.

However, every processor that contains a far field interaction block for which one of the cluster is in another processor will need Q from that processor. In general we expect to need an expand communication so that every processor may have access to Q from all other processors. This is where the processor topology may have significance on the performance of the algorithm. As explained in a former section (Section 6.2.2) about physical topology of the

Parystec CC-24, the underlying topology is generic and ensure a maximum of three links between any two processors [2]. Moreover, due to the functionality of the underlying EPX operating system that enable creation of virtual topologies we can create any standard topology that is more suited to our communication requirements.

The second stage of the Fast Multipole Method, called the downward pass, involves accumulation at cluster centers of the contributions from all other FF clusters. This stage, considering that a processor own the $\vec{\mathbf{x}}$ slice corresponding to the cluster that it owns, does not involve any communication whatsoever. Then the accumulated effect at the centre in distributed to all the elements in the cluster again with no communication involved.

As hinted earlier, the computation of $\vec{\mathbf{b}}_{NF}$ involves communication and is very much dependent on the distribution and organization of data among the processors that are participating in the $spM\times V$ algorithm.

In the nearfield computation every processor needs to get some slices of $\vec{\mathbf{x}}$ from other processors so as to accomplish determination of its $\vec{\mathbf{b}}_{NF}$ slice. It should be mentioned here also that, from the nature of the Conjugate Gradient Squared method (CGS), the $\vec{\mathbf{b}}$ and $\vec{\mathbf{x}}$ vectors are actually the same arrays as the latter becomes the former at the final stage of every iterative loop. Moreover, due to the symmetric nature of the matrix Z_{NF} if processor P_α need slice $\vec{\mathbf{x}}_i$ from processor P_β to compute $Z_{ji}\times\ \vec{\mathbf{x}}_i$ then it will have to send $\vec{\mathbf{x}}_j$ to P_β so that P_β can compute $Z_{ij}\times\ \vec{\mathbf{x}}_j$. This imply that the communication that is actually needed for nearfield computations is an efficient demand-driven swapping operation between peer processors.

Actually the parallel algorithm developed in this work is a combination of the modified version of the sequential algorithm with a demand-driven swapping and a Ring Expand algorithm called once. The swapping algorithm is used in precommunication and Ring Expand algorithm is used during the FMM based fast summation of the far field interaction. Needless to say this was because we designed the modified version keeping parallelism as the main objective of our design.

In the Figure 6.1 there is the pseudocode of our parallel algorithm. It is worthy noting that this is a Single Program Multiple Data (SPMD) type of algorithm in the sense that the same algorithm is run on all processors and data. The Z matrix and $\vec{\mathbf{b}}$ and $\vec{\mathbf{x}}$ vectors are partitioned so that every processor gets a number approximately $\frac{M}{\mathcal{P}}$ block rows and a slice of same size of x and b vectors. M and $\mathcal{P}$ mentioned here are the number of clusters and processors respectively. For consistence of terminology, we say processor p owns $\vec{\mathbf{x}}_p$ and $\vec{\mathbf{b}}_p$ slice of vectors $\vec{\mathbf{x}}$ and $\vec{\mathbf{b}}$ respectively.

The `Ring_expand` algorithm used in our parallel $spM\times V$ algorithm is given in Figure 6.2

6.4 Preconditioning the Solution Strategy

The numerical solution of electromagnetic scattering problems formulated as integral equations generally involves the solution of the solution of a system of linear equations. Such systems may be solved by either direct or iterative methods. For the iterative methods to be efficient, the convergence must be rapid, the system of equations must be well conditioned. Unfortunately, a number of convenient integral equation formulations for electromagnetic scattering are plagued by resonance problems [24]. Elsewhere, a technique based on the principle of limiting absorption that involves introducing an imaginary part to the wave number (equivalent to introducing a physical loss) is used [24]. However, The convergence rate of iterative methods depends on the spectral properties of the coefficient matrix. Hence, sometimes a transformation of the linear system can be done in such a way that the transformed matrix has the same solution as the original one but with more favourable spectral characteristics [7]. A preconditioner is a matrix that can realize the above mentioned transformation.

A broad class of preconditioners is based on incomplete factorizations of the coefficient matrix [7]. These are called incomplete because during the

factorization process certain positions that would be nonzero in an exact factorization are being ignored or taken as zero. Such a preconditioner is then given in factored form $Z = LU$ where L and U are lower and upper factors of Z respectively. The efficiency of preconditioners of this class depends on how well Z approximates the original matrix. Actually, the preconditioner adopted in this work belong to this class. A matrix resulting from only the nearfield interaction is taken as Z ignoring all the farfield interactions. In the sequential solution all the nearfield interactions were included in Z however in the parallel implementation some of the nearfield interaction had to be left out since they could have resulted into unwanted communication overhead and serialization complexity. However, this did not pass without a price as it will be demonstrated in the experimental results, generally the number of iteration to convergence tend to increase with finer data decomposition.

The nearfield interaction matrix Z_{NF} contain the interaction between the elements that are not distant enough to be computed by the FMM in the given accuracy. In the electromagnetic scattering problem this is dependent on the actual physical separation of the interacting elements. Z_{NF} has a special structure that it consist of sparse blockwise while the blocks themselves are dense. This rather peculiar structure had to be taken into consideration so as to come up with an implementation that will not only minimize the memory space requirement but also be able to apply faster dense matrix manipulation algorithm for the blocks themselves.

An important consideration for incomplete factorization preconditioners is the cost of the factorization process. Generally, the asymptotic cost of factorization is the same as solving a matrix directly, that is, $O(N^3)$, and even in the case of incomplete factorization the cost may equal to that of one or more iterations of the iterative method. On parallel computers this problem is aggravated by the generally poor parallel efficiency of the factorization [7]. However, such factorization costs can be amortized if the iterative method takes many iterations, or if the same preconditioner is used for a number of linear system such as in the successive time steps or Newton iterations.

Our incomplete factorization is based on the scheme called a "modified incomplete factorization". The basic idea of this scheme is that if the product $a_{i,k}a_{k,k}^{-1}a_{k,j}$ is nonzero and a fill is not allowed in position (i,j), instead of simply discarding this fill quantity substract it from diagonal element $a_{i,i}$. This modified incomplete factorizations are interesting because, in combination with small pertrubations, the spectral condition number of the preconditioned system can be of a lower order. For us this had a further attraction in the sense that it relaxes the need of fill-ins which are quite cumbersome to deal with when implementing in a naturally static memory programming language such as the one based the Fortran77 standard.

A block algorithm in a matrix computation is one that is defined in terms of operations on submatrices rather than matrix elements. Such matrices are well suited to many high performance computers because their data locality properties lead to efficient usage of memory hierachies[15] [6]. However, many of the current popular "block algorithms" are scalar algorithms in which the operations have been grouped and reordered into matrix operations. Our block factorization algorithm is also based on these block algorithms for more generality of application. A genuine block LU factorization is stable for a matrix that is symmetric positive definite or point diagonally dominant by rows or columns as long as it is well-conditioned. The nearfield interaction matrix that we seek to factorize is quite more general than that prescribed above.

6.4.1 Development of the Factorization Algorithm

The nearfield matrix that result from N-body modelling naturally exist in a form of dense blocks that are sparsely scattered accross the domain of computation. Although the whole matrix can be considered as a sparse matrix, wich it actually is, it can also be treated as a sparse collection of dense blocks. In order to make good use of the memory hierachy (Figure 6.3), this design is based on the latter consideration.

Useful work, such as floating point operations can only be done at the top

of the hierachy. So to work on data lower in the hierachy, it must first be transferred to the registers. This data movement is much slower than the rate at which processing take place in the registers. In fact in many computations more time is spent moving data in the hierachy than doing intended work.

The objective of adopting a block algorithm was to keep data near the top of the hierachy as long as possible while minimizing movement between levels. This affected by moving about blocks of data rather than elementary data elements. This has observed to be very effective as is shown by the experimental results in the Chapter 7.

A pseudocode of the blockwise sparse factorization developed in this work is is given in Figure 6.5. The areas referred to in the pseudocode are as indicated in the Figure 6.4.

; Perform demand-driven swapping $\vec{\mathbf{x}}$ required for multiplication
for each NF blocks $Z_{[I,J]}$ such that $I = K$ **do**
  if (owner(J) $\neq p$ **then**
    Let q = (owner(J)
    Send $\vec{\mathbf{x}}_i$ to q
    Receive $\vec{\mathbf{x}}_j$ from q
; Determine $\vec{\mathbf{b}}_{pNF}$ using sparse matrix multiplication method
; $\vec{\mathbf{b}}_K$ and $\vec{\mathbf{x}}_K$ are slices of $\vec{\mathbf{b}}$ and $\vec{\mathbf{x}}$ respectively.
for each clusters K in p's block row **do**
  for each NF blocks $Z_{[I,J]}$ such that $I = K$ **do**
    $\vec{\mathbf{b}}_K$ = Blas3Mult($Z_{[I,J]}, \vec{\mathbf{x}}_K$)
; Determine $\vec{\mathbf{b}}_{FF}$ using the Fast Multipole Method
; Perform the downward pass
for each clusters L **do**
  initialize to $Q(iQ)$ zero for all iQ
  for all elements l in L **do**
    get the global index of l into ig
    $Q(iQ) = Q(iQ) + B(iB) \times x(ig)$
; Perform an expand of the Q vector required for the upward pass
Call `Ring_Expand`(Q, mysize, $\mathcal{P}$)
; Perform the upward pass
for each clusters K **do**
  for all discretization ip **do**
    initialize $R(ip)$ to zero
  for each clusters L such that clusters Far(K,L) **do**
    for all discretization ip **do**
      $R(ip) = R(ip) + H(iH) \times Q(iQ)$
  for all discretization ip **do**
    for all elements k in K **do**
      $\vec{\mathbf{b}}$**(ig)** = **b(ig)** + $T(iT) * R(ip)$

Figure 6.1: The Parallel Matrix Vector Multiplication ($spM\times V$) Algorithm

```
p=0
frompid = mypid+1
if frompid=P+1 then frompid=0
m=size(mypid)
loc=location(mypid)
while p is less than P
    Send to Left Neighbor M(loc:loc+m-1)
    loc=location(frompid)
    m=size(frompid)
    Receive from Right Neighbor M(loc:loc+m-1)
    frompid = mypid+1
    if frompid=P+1 then frompid=0
    increment p
```

Figure 6.2: The Ring Expand Algorithm

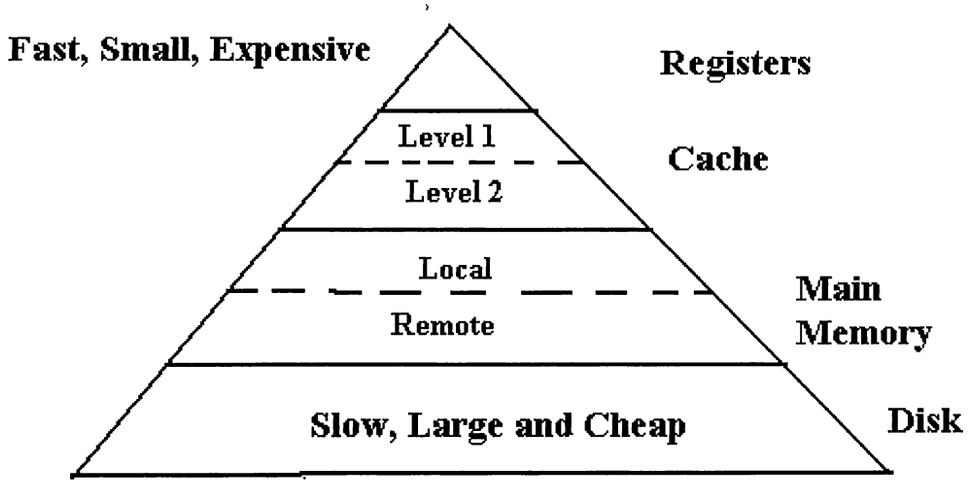


Figure 6.3: The Memory Hierachy

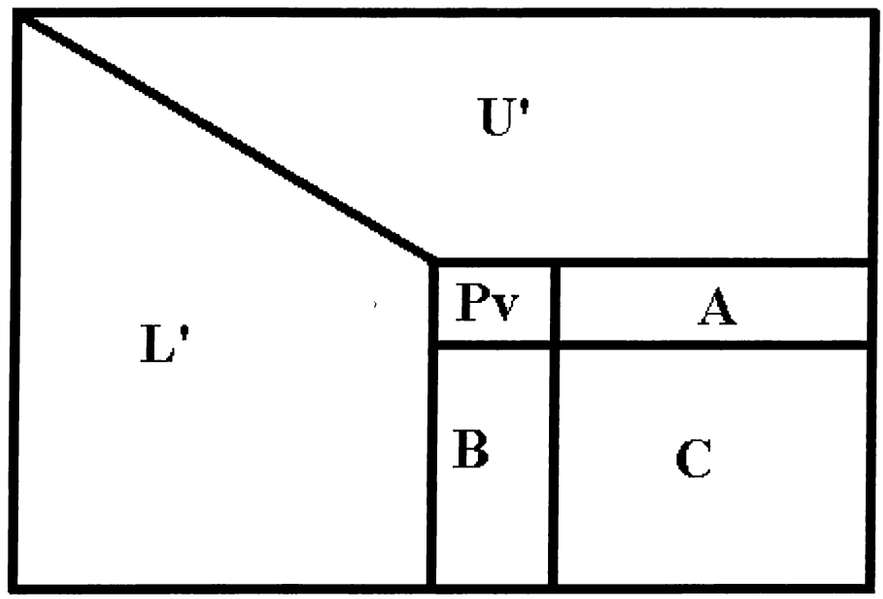


Figure 6.4: The Proceeding of Blockwise Sparse Matrix Factorization

```
for index_cluster=1 to number_of_clusters do
    for all blocks do
        if (pivot_block) then
            factorize block A_p = L_p U_p
        if (block A_i in area A) then
            solve for U_i in L_p U_i = A_i
            A_i = U_i
        if (block B_j in area B) then
            solve for L_j in L_j U_p = B_j
            B_j = L_j
    for all blocks U_i in area A do
        let col be the column index of U_i
        for all blocks L_j whose row index is index_cluster do
        ;Update blocks in area C
        if block Z_ij exist then
            Z_ij = Z_ij - L_j U_i
        else
            A_p = A_p - L_j U_i
```

Figure 6.5: The Blockwise Sparse Factorization Algorithm

Chapter 7

Analysis of the Experimental Results

The parallel algorithm designed in this work was implemented on a Parystec CC 24 multicomputer using the AIX Fortran that conform completely with the Fortran 77 standard and partially to the Fortran 90 standard. This implementation was tested using an even number of processors starting with two up to 24 and various aspects of the results were recorded. The results were then compared with the modified sequential program. The data collected was for time per a single iteration, number of iterations to convergence, and time for factorization of the NF matrix.

In Table 7.1 there is a short descripition of the problems solved during these experiments.

In the sections that follow we shall demonstrate and discuss the various

No. of Clusters	No. of elements	Sparsity of NF
36	1201	0.08178
50	2402	0.0592
111	12010	0.02686

Table 7.1: The characteristics of the electromagnetic scattering problems solved in these experiments

P	*iter*	*t_soln*	*s_iter*	*s_up*	*eff*	*t_fact*
1	19	12.12	0.6379	1.0000	1.000	1.45
2	23	7.70	0.3348	1.9053	0.9527	1.04
4	23	3.99	0.1735	3.6766	0.9192	0.49
6	23	2.82	0.1226	5.2031	0.8672	0.33
8	28	3.05	0.1089	5.8576	0.7322	0.26
10	30	2.82	0.094	6.7862	0.6786	0.20
12	28	2.40	0.0857	7.4434	0.6203	0.15
14	28	2.47	0.0882	7.2324	0.5166	0.15
16	28	2.59	0.0925	6.8962	0.4310	0.14
18	30	2.59	0.0863	7.3916	0.4106	0.08
20	38	3.70	0.0974	6.5492	0.3275	0.09
22	34	3.74	0.11	5.799	0.2636	0.10
24	31	2.97	0.0958	6.6586	0.2774	0.09

Table 7.2: Timing results for the 36 clusters problem

aspects of the results that were obtained from the experiments.

7.1 Timing Results

In Tables 7.2, 7.3, and 7.4, P refers to number of processors used, *iter* refers to number of iterations required to solution, *s_up* refers to speed up attained, *eff* refers to the parallel efficiency of the solution, and *t_iter*, *t_soln*, *t_factor* refers to one iteration, full solution, and factorization times in seconds respectively. It should be noted that the accuracy of the Parystec timing is up to the millisecond. Importance is given to the time for one iteration since it reflects the improvements on the $spM\times V$ algorithm which is the primary objective of this work.

7.2 The time for a single iteration

This is the time that a program uses to execute one loop of the iterative routine. This is found by synchronizing all the processors and recording time

P	*iter*	*t_soln*	*s_iter*	*s_up*	*eff*	*t_fact*
1	20	34.07	1.7035	1.0000	1.0000	6.16
2	25	21.48	0.8592	1.9827	0.9913	4.46
4	25	11.81	0.4724	3.6061	0.9015	2.25
6	27	8.62	0.3192	5.3368	0.8895	1.54
8	27	6.90	0.2552	6.6752	0.8344	1.16
10	31	6.29	0.2029	8.3958	0.8396	0.80
12	29	5.92	0.2041	8.3464	0.6955	0.80
14	32	5.41	0.1691	10.074	0.7196	0.62
16	31	5.61	0.181	9.4116	0.5882	0.62
18	32	4.46	0.1485	11.471	0.6373	0.45
20	33	4.94	0.1497	11.379	0.5690	0.44
22	31	4.94	0.1594	10.687	0.4858	0.44
24	31	5.02	0.1619	10.522	0.4384	0.45

Table 7.3: The timing results for the 50 cluster problem

P	*iter*	*t_soln*	*s_iter*	*s_up*	*eff*	*t_fact*
4	33	145.73	4.4161	4.0000	1.0000	57.13
6	31	93.11	3.0035	5.8813	0.9802	38.62
8	34	74.36	2.1871	8.0766	1.0096	27.90
10	32	60.16	1.8800	9.3960	0.9395	23.69
12	32	50.62	1.5819	11.166	0.9305	19.48
14	35	44.63	1.2751	13.853	0.9895	15.27
16	32	35.63	1.1134	15.865	0.9916	13.19
18	35	39.25	1.1214	15.752	0.8751	13.34
20	37	35.41	0.9570	18.458	0.9229	11.02
22	34	31.41	0.9238	19.121	0.8692	11.27
24	39	30.92	0.7928	22.281	0.9284	9.27

Table 7.4: Timing results for the 111 cluster problem

immediately the iterations starts and again taking the time after all processors have finished their iterations. The time difference is divided by the number of iterations that were done before convergence. It should be noted that because the accuracy of the Parystec timing is limited to the millisecond small reading times such as for the 36 and 50 cluster problem are quite not reliable.

This time is a sum of computation and communication time. As seen in Figures 7.1 and 7.2 this time diminishes gradually as the number of processors is increased until it reaches a certain number of processors, then it stabilizes and after a short range the time values start to increase. The reason behind this behavior is that with an increased number of processors we both reduce the computational load per processor and increase the communication overhead required to complete the computation. The number of processors for which time stop diminishing depends on the granularity of the computation which in turn depends on the problem size. As it can be observed from the figures, this critical number increases from 10 in the 36 clusters problem to 14 in the 50 clusters problem; as for the 111 cluster problem this number is beyond the available 24 processors of the Parystec.

However, when the problem size is large enough to replenish the available physical memory yet another factor affecting this time arises. This is due to virtual memory allocation. As seen from Figure 7.3, the reduction of time is more than expected using the given number of processors. This behavior is attributed to the fact that there is much more time spent in swapping at larger granularity using less number of processors than otherwise.

Yet another interesting feature can be observed from the timing results collected in these experiments. This is due to the asymmetry of the processors of the Parystec CC 24 system. As the processor number is extended beyond 16 there is a need to include the 4 IO-Nodes of the system. These IO processors have different properties from the others e.g. running some more operating system routines and having more physical memory. A careful look at the graphs in Figures 7.1, 7.2, and 7.3 at the point where processor number becomes 18 reveals this interesting feature. This can be seen in the efficiency, and total

solution times graphs as well.

Moreover, granularity has been a reason for performance degradation that is observed especially for small sized problems. Considering the case of the 12 cluster problem with 121 elements as shown in Figure 7.1, we see as the number of processors is increased the granularity gets smaller and hence also the efficiency.

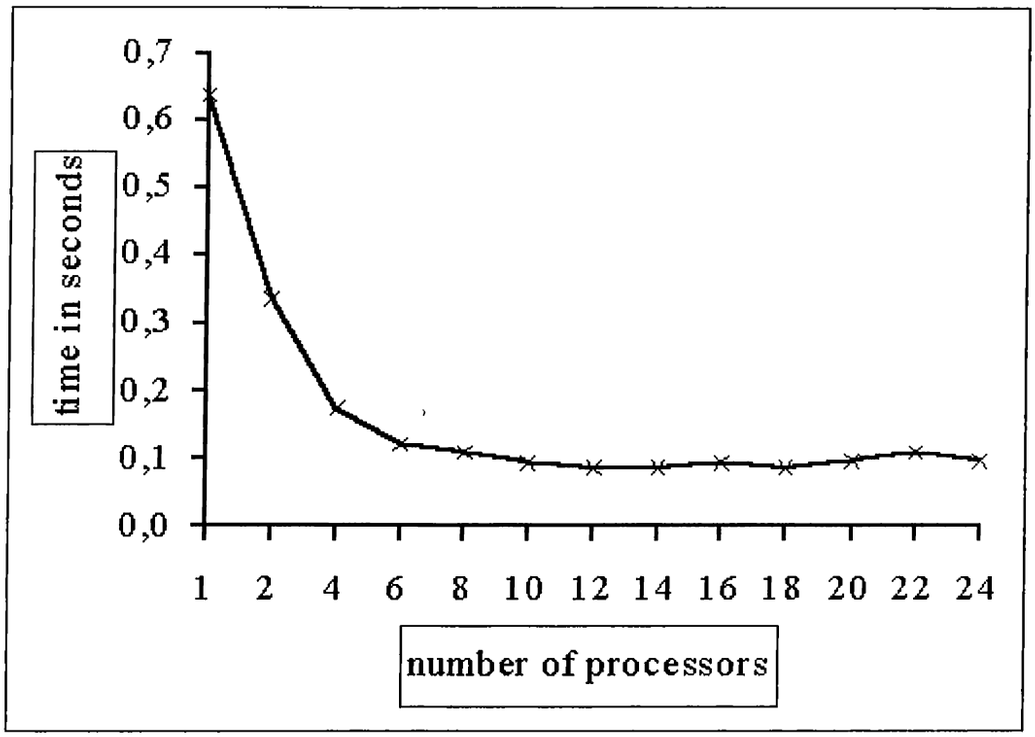


Figure 7.1: Time for one iteration in the solution of the 36 clusters problem

7.3 Number of Iterations

As the number of processors used is increased, the number of iterations also increases. This is attributed to less accuracy of the preconditioner used resulting from matrix partitioning. The preconditioner used is based on an incomplete

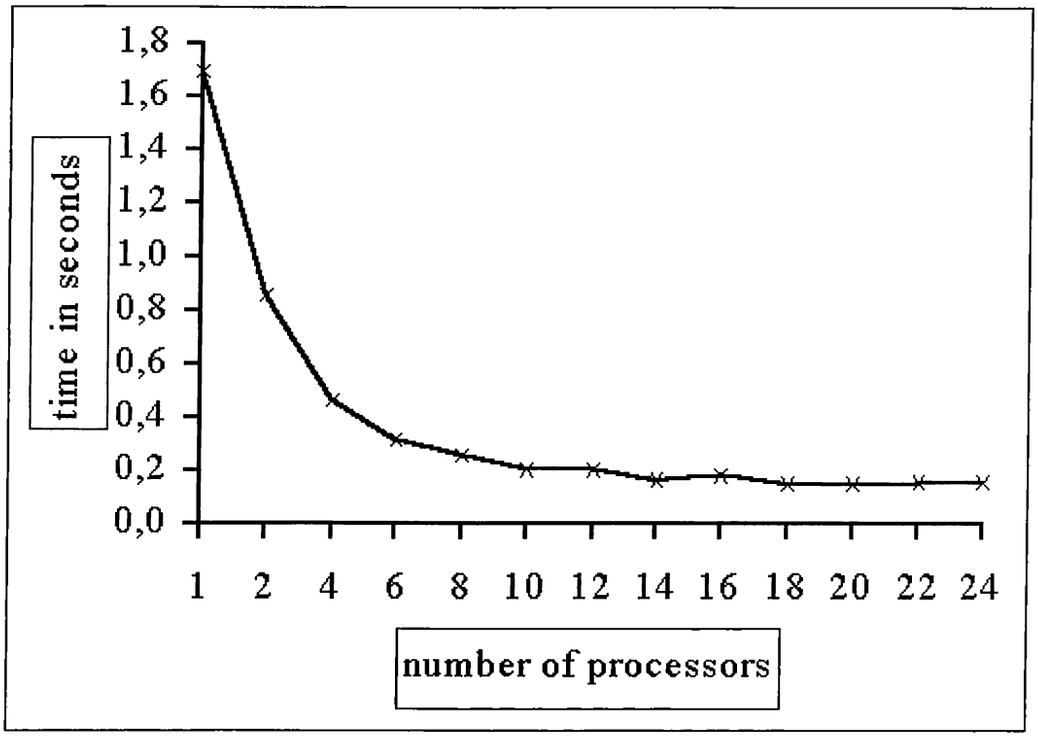


Figure 7.2: Time for one iteration in the solution of the 50 cluster problem

factorization of the nearfield interaction matrix. Because of the data partitioning used, having a complete factorization would have lead to unreasonably large communication volume overhead, so a trade-off was struck and a no-communication parallel factorization routine was used. It is interesting to note that as the problem size increases, the granularity increases and hence the accuracy of the preconditioner is improved accordingly. This fact explain the less dependency on the number of processors of the convergence rate of larger problems as seen in Chart 7.4.

The number of iteration have a direct effect on the complete solution of the electromagnetic scattering problem. The complete solution time is the sum of factorization time and the time for convergence of the iterative scheme. The complete solution times are influenced by both the factors affecting the time for a single iteration explained in Section 7.2 and the factorization times. The

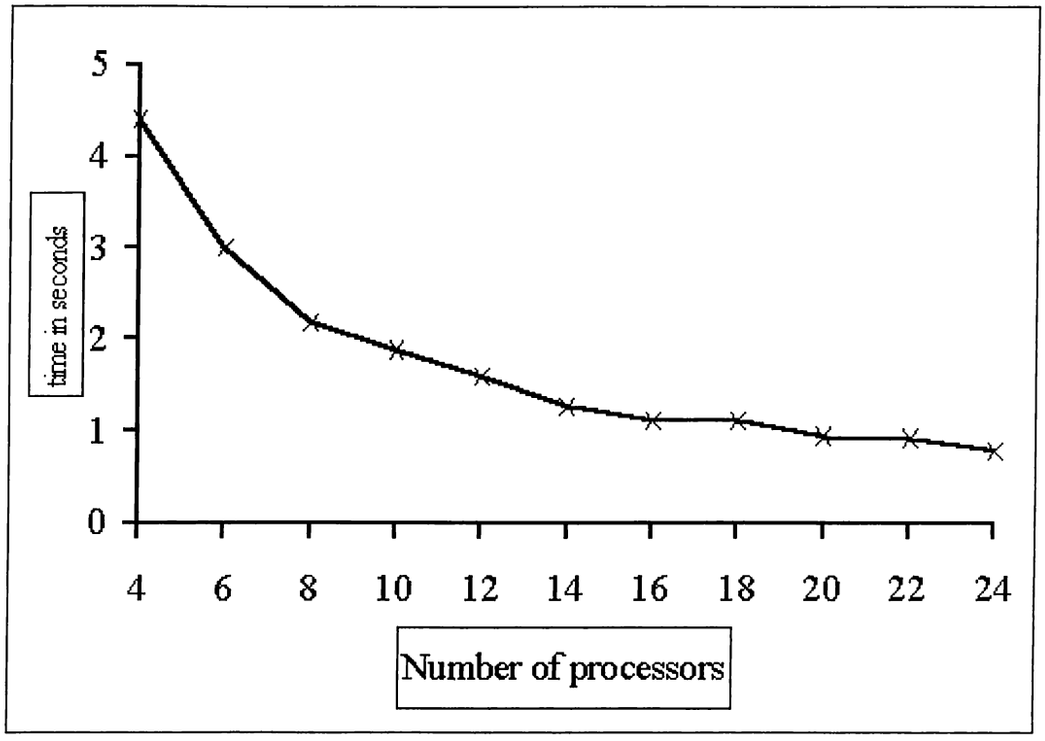


Figure 7.3: Time for one iteration in the solution of the 111 cluster problem

Charts 7.5, 7.6 and 7.7 exhibit the behavior of complete solution times as the number of processors used is increased from 1 to 24.

7.4 Parallel Efficiency and Speedup

In a parallel program use of a multitude of processing nodes is employed to fasten the solution of a problem. How faster the program become is termed the speedup of the parallel program. Mathematically, the speedup is the ratio of the time requirements of a fastest sequential program to that of the parallel program using a multitude of processors. The chart in Figure 7.8 shows the speedup attained using even numbers of processors in the range from 2 to 24 to solve 36, 50, and 111 clusters problems. It should be noted that because of problem size the 111 cluster problem could not be solved neither sequentially

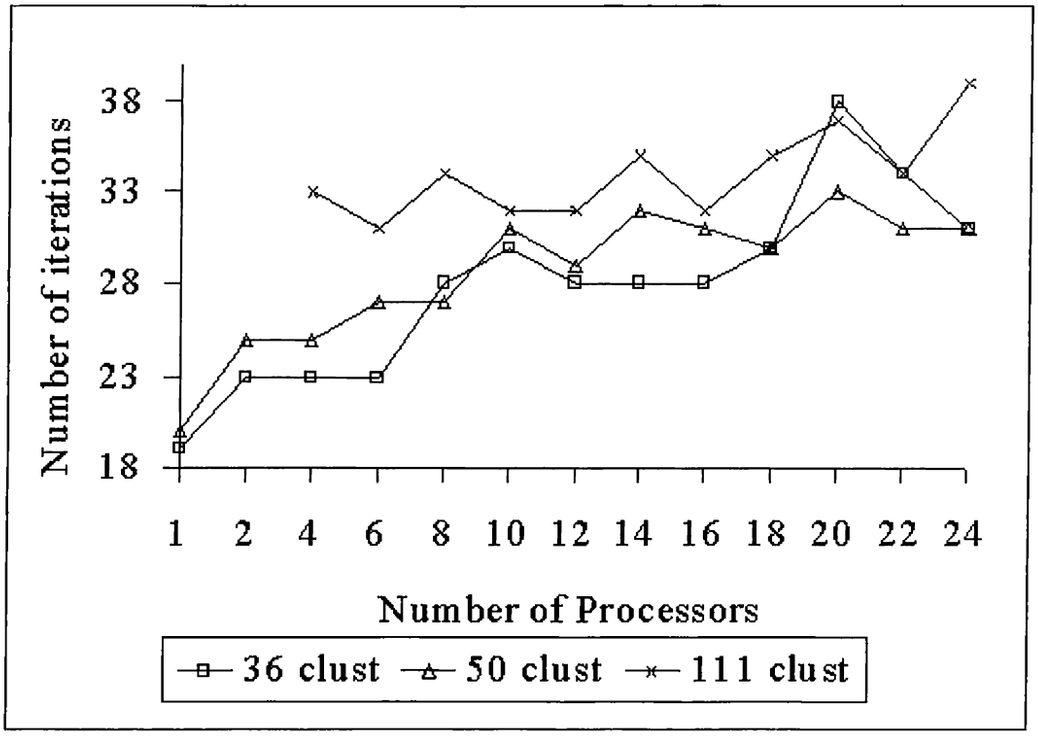


Figure 7.4: Number of iterations required for solutions of the 36, 50 and 111 clusters problems

nor using less that 4 processors so the speedups are considered using 4 processors as the base reference. The anomalies observed in the chart are explained in Section 7.2.

Efficiency of a parallel program indicates to what extend we are utilizing the added computing resources. It provides us with a clue so as to what number of processors should be employed to realize the best speedup in solving a problem of certain size. It is directly derived from the speedup attained and the number of processors used, actually it is a ratio of the speedup attained to the number of processors used. Generally, efficiency is expected to decrease with increase in number of processors used due to increased communication overhead involved. The chart in Figure 7.9 demonstrate how the efficiency of our program changes with increase of the processors in even steps from 2 to 24. However, for the reasons explained in Section 7.2 the efficiency curve depict some anomalies

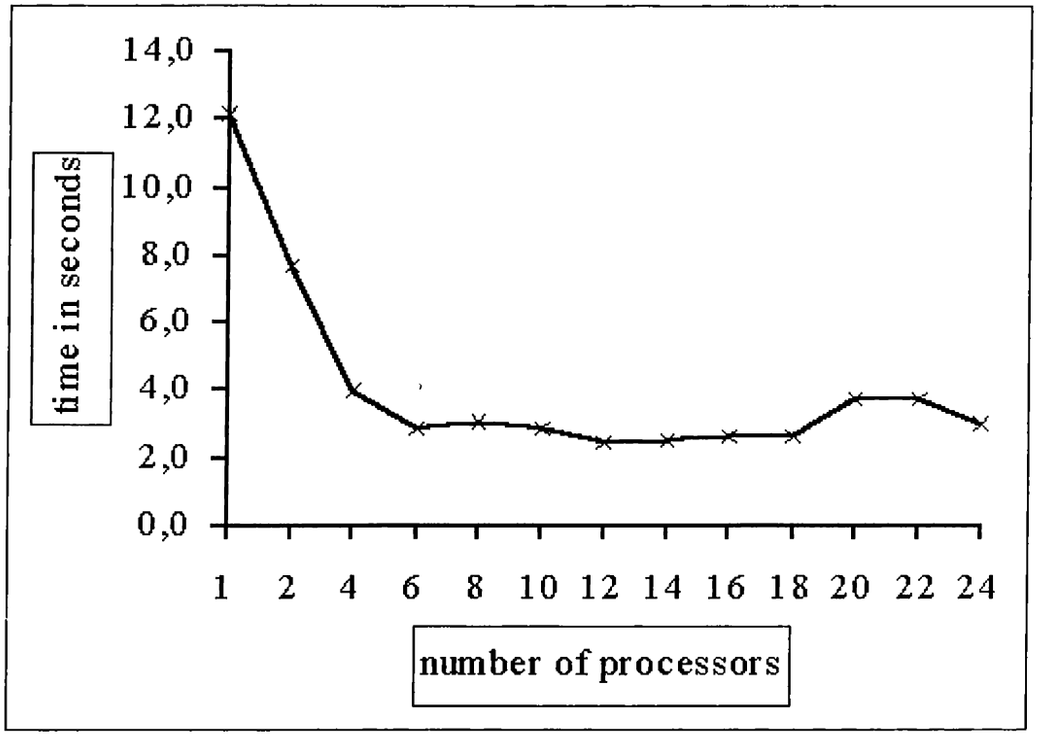


Figure 7.5: The complete solution time for the 36 cluster problem

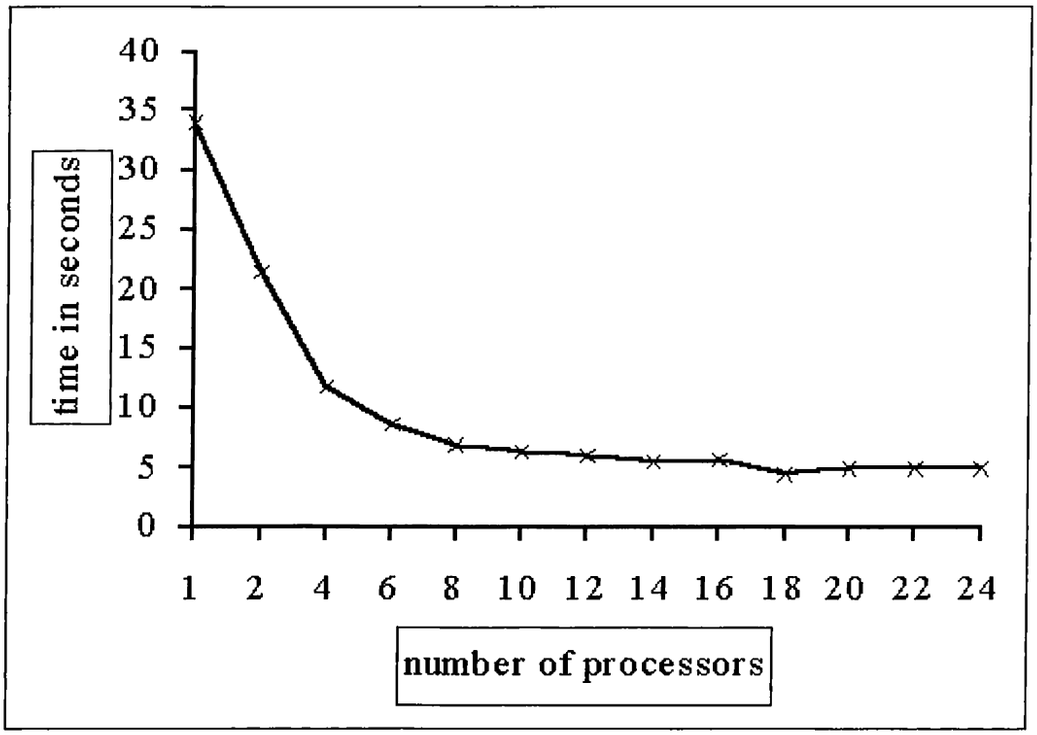


Figure 7.6: Time for complete solution of the 50 clusters problem.

starting at 18 processors use for all problem sizes and throughout the processor range for the 111 clusters problem.

7.5 Preconditioning

In this work a blockwise sparse factorization was developed and used for preconditioning. The original implementation was using a routine spFactor from the Sparse library package and in the problems that were used in these experiments as described in Section 1 of this chapter. The rate of convergence when using the new blockwise factorization was always exactly the same as using the routines from the Sparse library. However, the new routine by far outperformed the library routines as the Table 7.5 shows.

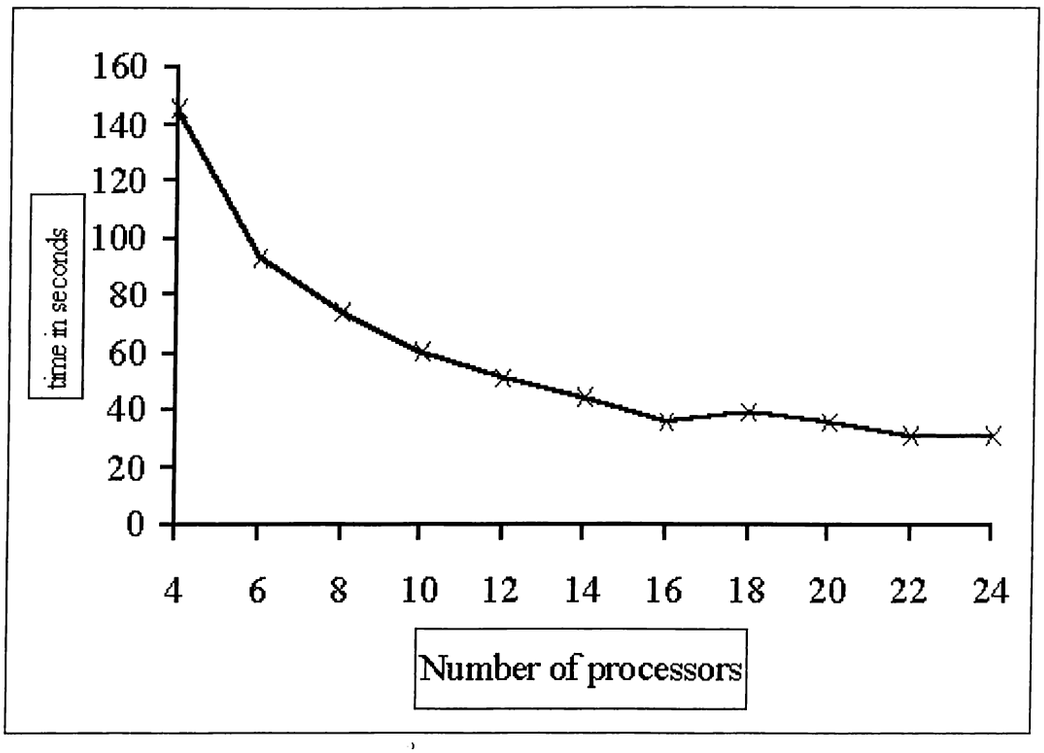


Figure 7.7: Time for complete solution of the 111 clusters problem

Moreover, since the original implementation was using a triangular solver that was compatible with spFactor for our blockwise design we implemented a blockwise sparse triangular solver which slightly outperformed the original one and added up to improvement of the one iteration loop runtime. This improvement can be observed on the Table 7.6. The comparison of the running time per a single iteration for the two versions is given in the Table 7.7

Cluster Size	Lib	New
121	0.19	0.04
1201	14.02	1.45
2402	90.50	6.16

Table 7.5: The comparison between factorization time of the Sparse library spFactor and our new Blockwise sparse factorization algorithm

Cluster Size	Lib	New
121	1.1	0.54
1201	37.2	13.91
2402	148.52	40.84

Table 7.6: The problem solution time (factorization+looping to convergence) of the Sparse library spFactor compared to our new Blockwise sparse factorization algorithm

$\frac{Algo}{ClustSize}$	Lib	New
6	0.004	0.002
121	0.07	0.0425
1201	1.22	0.6379
2402	2.901	1.7035

Table 7.7: The comparison between running time per a single loop of the original implementation and the new modified implementation

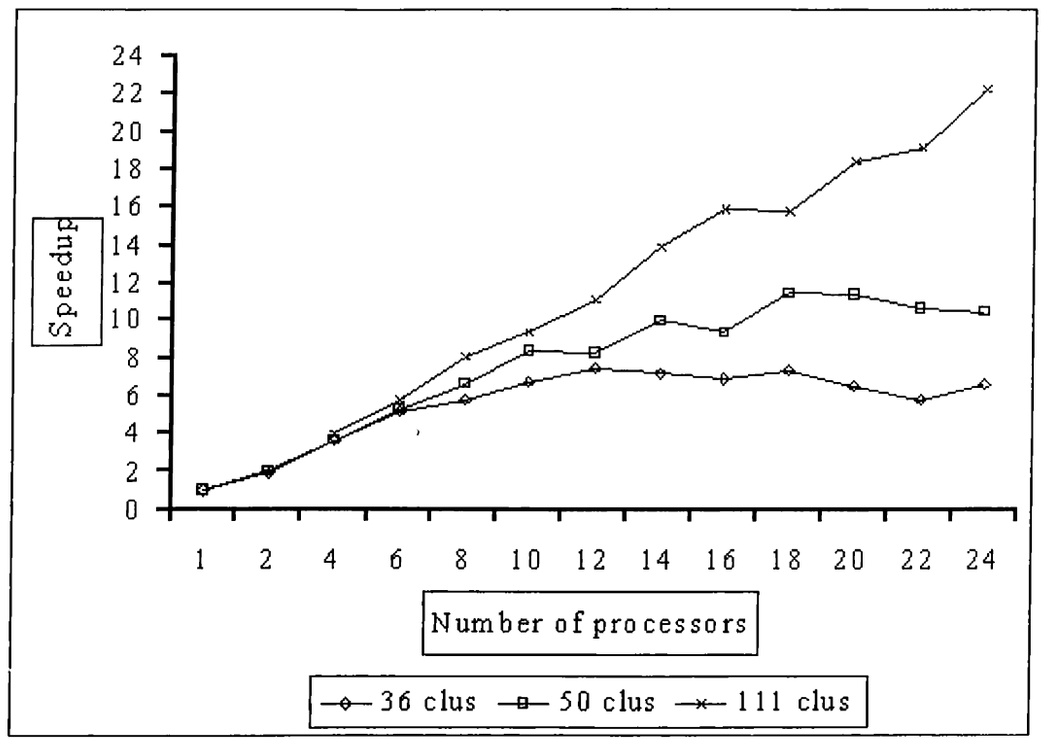


Figure 7.8: The speedups attained in solving the 36, 50, and 111 clusters problems

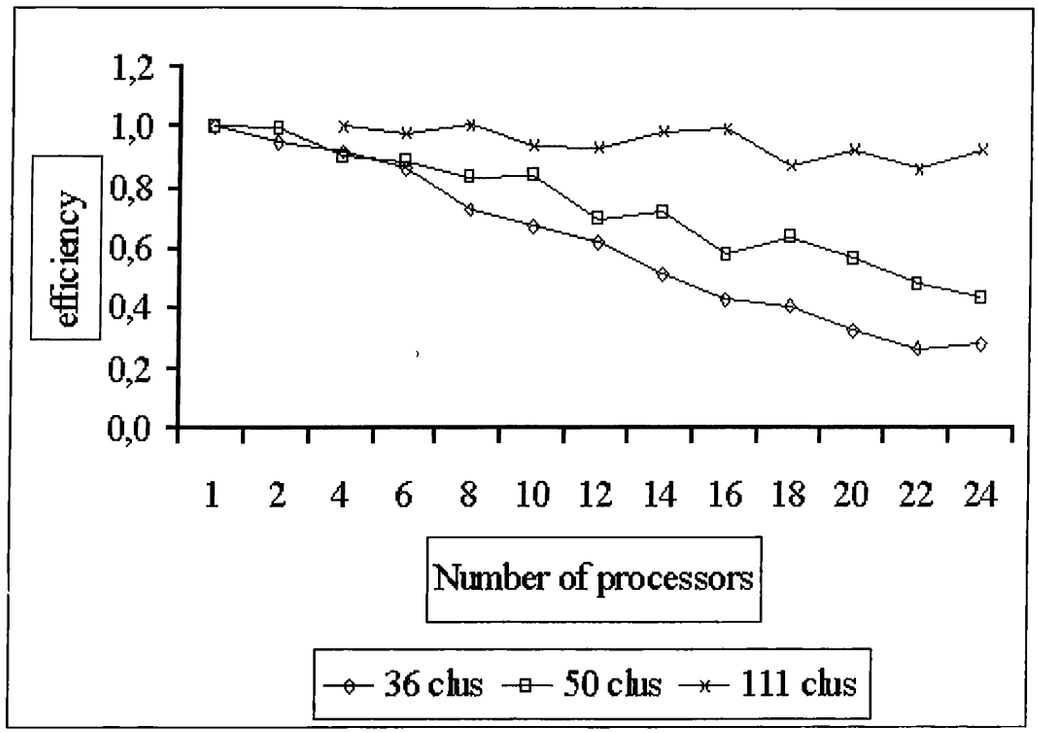


Figure 7.9: The variation of the efficiency of the parallel scattering solution.

Chapter 8

Conclusion

8.1 Final Remarks

In this work a parallel algorithm was developed that used the FMM concept to solve a certain electromagnetic scattering problem. This algorithm is implemented on the Parystec CC-24 multicomputer and the program was run to solve various test problems of varying sizes. The results of these experimental runs exposed the various peculiar features of the electromagnetic scattering problems such as its data locality that can be exploited to make good use of caching and memory hierachy. What is more interesting is that some of these features apply to the more generic N-body problem as well.

Essentially, the fast multipole method can be considered as a template or rather a framework that may be used for solving a wide range of problems that can be modelled as an N-body. Depending on a particular implementation the computation elements can be modelled as interacting particles and the analytical transformations are used to represent the abstract upward passes and downward passes of the generic FMM algorithm.

This is observed from the use of multipoles as fundamental computation elements in the original Greengard-Rokhlin FMM; use of the discretizations of

the Poisson equation in Anderson's FMM and discretization with trapezoidal numerical scheme in this work.

This calls for an implementation strategy that will be general and act as a parametric program for which the user only needs to set some specific parameters for the program to be able to solve a particular problem.

8.2 Similar Work

Since its first publication by Rokhlin, the FMM has proven quite indispensable to the scientific and engineering computation community. In his PhD dissertation [13], James F. Leathrum applied a serial version of FMM into solving an integrated molecular dynamics solver and in the solution of a Gaussian distribution of bodies such as might occur in gravitational force studies.

In another doctorate dissertation [8], Gavin J. Pringle designed a parallel version of the FMM in 2 and 3 dimensions and implemented it on a Meiko Computing Surface, CS-1. That FMM version was designed to solve a particular fluid dynamics problem based on the turbulent flow model. In that work, it was observed that the break even point between FMM and direct method, i.e., the point at which both methods take same time to execute, is $N \approx 180$ and $N \approx 5000$ vortex particles for the sequential 2 and 3 dimensional FMM.

8.3 Future Work

For the future I foresee work done to develop a parallel N-body generic solver that would enable every problem that is compatible with the N-body model be solved efficiently by just supplying a few parameters that specify the current problem.

My vision is that such a framework sort of a solver will need to use artificial intelligent agent clusters organized into different nodes of a multiprocessing

machine and geared to solve a common problem. My expectations are that in the future the existing parallel processing paradigms such as the SIMD, MIMD and SPMD may be replaced by a paradigm based on artificial agents. This can be preceived clearly if one follows the trend of the successful use of multiagent systems in a range of applications as wide as from economics to mainstream software engineering systems.

Appendix A

Mathematical Modelling of the Fast Multipole Method

A.1 The N-body Model

Consider an N-body consisting of particles that interact according to Coulomb's law. A charge of magnitude q_0 located at point $\mathbf{x}_0 = (x_0, y_0) \in \mathbb{R}^2$ has a corresponding electrostatic force expressed by

$$F_{\mathbf{x}_0}(x, y) = q_0 \frac{(\mathbf{x} - \mathbf{x}_0)}{\|\mathbf{x} - \mathbf{x}_0\|^2} \tag{A.1}$$

and, hence it also has a potential given by

$$\phi_{\mathbf{x}_0}(x, y) = -q_0 \log\left(\|\mathbf{x} - \mathbf{x}_0\|^2\right) \tag{A.2}$$

at an arbitrary point $\mathbf{x} = (x, y) \in \mathbb{R}^2$ as dictated by Coulomb's law [10]. Since $\phi_{\mathbf{x}_0}$ is harmonic in any region not containing the point $\mathbf{x}_0$ and that for every harmonic function f there exist an analytic function g such that $u(x, y) = Re(g(x, y))$ where g is unique except for an additive constant, in the context of this derivation no distinction is made between a point $(x, y) \in \mathbb{R}^2$ and $z = x + iy \in \mathbb{C}$. As a consequence of the above analysis, the potential due to a charge is modelled as the analytic function $\log(z)$ and the equation (A.2)

above becomes

$$\phi_{z_0}(z) = q_0 \log(z - z_0), \tag{A.3}$$

but

$$\begin{aligned} \log(z - z_0) &= \log(z - z_0) - \log(z) + \log(z) \\ &= \log\left(\frac{z - z_0}{z}\right) + \log(z) \\ &= \log(1 - \frac{z_0}{z}) + \log(z). \end{aligned}$$

Moreover, since $\left|\frac{z_0}{z}\right| < 1$ and that

$$\log(1 - \tau) = (-1) \sum_{k=1}^{\infty} \frac{\tau^k}{k} \qquad \forall \tau, |\tau| < 1,$$

we have

$$\log(z - z_0) = (-1) \left[\sum_{k=1}^{\infty} \frac{1}{k} \left(\frac{z_0}{z}\right)^k\right] + \log(z)$$

and the expression (A.3) becomes

$$\phi_{z_0}(z) = q_0 \left(\log(z) - \sum_{k=1}^{\infty} \frac{1}{k} \left(\frac{z_0}{z}\right)^k\right). \tag{A.4}$$

A.2 Derivation of the Multipole Expansion

Substituting (A.4) into (3.2) we get

$$\begin{aligned} \Phi(z_j) &= \sum_{i=1}^{N} q_i \left(\log(z_j) - \sum_{k=1}^{\infty} \frac{1}{k} \left(\frac{z_i}{z_j}\right)^k\right) \\ &= \sum_{i=1}^{N} q_i \log(z_j) - \sum_{i=1}^{N} q_i \left(\sum_{k=1}^{\infty} \frac{1}{k} \left(\frac{z_i}{z_j}\right)^k\right) \\ &= \log(z_j) \sum_{i=1}^{N} q_i - \sum_{k=1}^{\infty} \sum_{i=1}^{N} \frac{q_i}{k} \left(\frac{z_i}{z_j}\right)^k \\ &= \log(z_j) \sum_{i=1}^{N} q_i + \sum_{k=1}^{\infty} \frac{1}{z_j^k} \sum_{i=1}^{N} \frac{-q_i z_i^k}{k}. \end{aligned}$$

Therefore, given $Q = \sum_{i=1}^{N} q_i$ and $\alpha_k = \sum_{i=1}^{N} \frac{-q_i z_i^k}{k}$, we have

$$\Phi(z_j) = Q \log(z_j) + \sum_{k=1}^{\infty} \frac{\alpha_k}{z_j^k}. \tag{A.5}$$

A.3 Error Bounds of a Multipole Expansion

The above expansion (A.5), termed the multipole expansion, gives the potential at point z_j accurately. However, it is just theoretical as it require an infinite number of multipole terms to obtain the result. The number of terms of the multipole expansion is the key factor to the accuracy of the solution as shown in the following expression [12]

$$\varepsilon_p = \left|\Phi(z_j) - Q\log(z_j) - \sum_{k=1}^{p} \frac{\alpha_k}{z_j^k}\right| = \left|\sum_{k=p+1}^{\infty} \frac{\alpha_k}{z_j^k}\right|.$$

Considering again $\alpha_k = \sum_{i=1}^{N} \frac{-q_i z_i^k}{k}$ and letting $A = \sum_{i=1}^{N} |q_i|$ we get

$$\varepsilon_p = \left|\sum_{k=p+1}^{\infty} \frac{\alpha_k}{z_j^k}\right| \leq A \sum_{k=p+1}^{\infty} \frac{r^k}{k\,|z_j|^k} \leq A \sum_{k=p+1}^{\infty} \left|\frac{r}{z_j}\right|^k = \gamma \left|\frac{r}{z_j}\right|^{p+1}, \quad \text{(A.6)}$$

where $\gamma = \frac{A}{1-\left|\frac{R}{z_j}\right|}$ and r is the radius of the disk containing z_i. This shows that the error due to truncation of the multipole expansion to p terms is bounded above by ε_p as given in expression (A.6) above. This enable the accuracy of the solution to be determined by the user by adjusting the number of terms p of the multipole expansion to be included in the potential field determination. Moreover, suppose we insist $|z_j| > cr$, for some $c > 1$, and since $z_i < r$ then $\left|\frac{z_i}{z_j}\right| < \frac{1}{c}$. It is shown in [10] that for a given precision, ε_p, the number of terms in the truncated series, p, is determined by

$$p = \lceil \log_c(\varepsilon_p) \rceil \quad \text{(A.7)}$$

A.4 Derivation of the potential sum using Anderson's FMM

In the following we seek to derive the expression for the potential sum in equation (3.2) using the FMM given by Anderson [3].

Let $\phi(r,\theta)$ be a solution of Laplace's exterior to the disk of radius a and circulation $\kappa = \sum_{i=1}^{N} \kappa_i$, then if $f(\theta) = \phi(r,\theta) - \kappa\log(r)$ we have

$$\begin{aligned}\phi(r,\theta) &= \kappa\log(r) + f(\theta) \\ &= \kappa\log(r) + \sum_{k=-\infty}^{\infty} c_k \left(\frac{a}{r}\right)^{|k|} e^{-ik\theta}, \end{aligned} \tag{A.8}$$

where c_k is the kth Fourier coefficient of the func+tion $f(\theta)$ given by

$$c_k = \frac{1}{2\pi}\int_0^{2\pi} f(s)e^{-iks}ds. \tag{A.9}$$

From (A.8) and (A.9) we get

$$\phi(r,\theta) = \kappa\log(r) + \sum_{k=-\infty}^{\infty} \left[\frac{1}{2\pi}\int_0^{2\pi} f(s)e^{-iks}ds\right] \left(\frac{a}{r}\right)^{|k|} e^{-ik\theta}. \tag{A.10}$$

When we interchange the summation and integration in (A.10), we get

$$\phi(r,\theta) = \kappa\log(r) + \frac{1}{2\pi}\int_0^{2\pi} f(s) \left[\sum_{k=-\infty}^{\infty} e^{-ik(\theta-s)} \left(\frac{a}{r}\right)^{|k|}\right] ds \tag{A.11}$$

$$= \kappa\log(r) + \frac{1}{2\pi}\int_0^{2\pi} f(s) \left[\frac{1-\left(\frac{a}{r}\right)^2}{1-2\left(\frac{a}{r}\right)\cos(\theta-s)+\left(\frac{a}{r}\right)^2}\right] ds \tag{A.12}$$

Moreover, if we use the trapezoidal rule to approximate the integral (A.12) above setting $h = 2\pi/k$ and $s_i = (a\cos(ih), a\sin(ih)), i \in [1,\ldots,K]$, then we have

$$\phi(r,\theta) = \kappa\log(r) + \frac{1}{2\pi}\sum_{i=1}^{K} f(s_i) \left[\frac{1-\left(\frac{a}{r}\right)^2}{1-2\left(\frac{a}{r}\right)\cos(\theta-s_i)+\left(\frac{a}{r}\right)^2}\right] h \tag{A.13}$$

This (A.13) approximation is equivalent to implicitly using the trapezoidal rule to approximate the Fourier coefficients in (A.9). If only $K = 2M+1$ nodes are used in the quadrature rule, then we should not use any coefficients $c_k, |k| > M$; that is, we have to use only the Fourier modes that can be reliably estimated using K equispaced points [3]. On the light of this conclusion, using the only first M modes in (A.8), we obtain

$$\phi(r,\theta) = \kappa\log(r) + \frac{1}{2\pi}\int_0^{2\pi} \left[\sum_{k=-M}^{M} e^{-ik(\theta-s)} \left(\frac{a}{r}\right)^{|k|}\right] f(s)$$

$$
\begin{aligned}
&= \kappa \log(r) + \frac{1}{2\pi} \int_0^{2\pi} f(s) \times \\
&\quad \left[\frac{1 - \left(\frac{a}{r}\right)^2 - 2\left(\frac{a}{r}\right)^{M+1} \cos(M+1)(\theta + s)}{1 - 2\left(\frac{a}{r}\right)\cos(\theta - s) + \left(\frac{a}{r}\right)^2} \right] ds + \\
&\quad \left[\frac{2\left(\frac{a}{r}\right)^{M+2} \cos(M(\theta - s))}{1 - 2\left(\frac{a}{r}\right)\cos(\theta - s) + \left(\frac{a}{r}\right)^2} \right]. \qquad \text{(A.14)}
\end{aligned}
$$

When the above integral (A.14) is approximated by the trapezoidal rule we obtain our required approximation

$$
\begin{aligned}
\phi(r,\theta) &\approx \kappa \log(r) + \frac{1}{2\pi} \sum_{i=1}^{K} f(s_i) \times \\
&\quad \left[\frac{1 - \left(\frac{a}{r}\right)^2 - 2\left(\frac{a}{r}\right)^{M+1} \cos(M+1)(\theta + s_i)}{1 - 2\left(\frac{a}{r}\right)\cos(\theta - s_i) + \left(\frac{a}{r}\right)^2} \right] h + \\
&\quad \left[\frac{2\left(\frac{a}{r}\right)^{M+2} \cos(M(\theta - s_i))}{1 - 2\left(\frac{a}{r}\right)\cos(\theta - s_i) + \left(\frac{a}{r}\right)^2} \right] h. \qquad \text{(A.15)}
\end{aligned}
$$

Bibliography

[1] *Embedded PARIX Software Documentation.* PARYSTEC Eastern Europe GmbH, 1996

[2] *Logical CC Systemview : Linktopology Bilkent.* PARYSTEC Eastern Europe GmbH, 1996

[3] Anderson Christopher R., "An Implementation of the Fast Multipole Method without Multipoles", *SIAM Journal of Statistical Computation*, 13(4):923–947, July 1992.

[4] Buttke T. F., "Fast Vortex Methods in Three Dimensions", Unpublished report, Courant Institute of Mathematical Sciences Sept, 1990.

[5] Çatalyürek Ümit V. and Aykanat Cevdet, "Decomposing irregularly sparse matrices for parallel matrix-vector multiplication", Technical report, Dept. Of Computer Eng. and Informatic Sciences, Bilkent University, 1996.

[6] Demmel James W. and Higham Nicholas J., "Stability of Block Algorithms with Fast Level 3 BLAS", Unpublished Report, July 1990.

[7] Dongarra J., Barret R., and Berry M., Demmel J., Donato J., Eijkkhout V., Pozo R., Romine C., and van der Vorst H., *Templates for Solution of Linear Systems : Building Blocks for Iterative methods.* Philadelphia, PA : SIAM, 1994 also available at http://netlib2.cs.utl.edu/templates/index.html

[8] Gavin Pringle J., "Numerical Study of Three-Dimensional Flow using Fast Parallel Particle Algorithms", PhD thesis, Napier University, 1994.

[9] Greengard, L. and Groop, W. D., "A Parallel Version of the Fast Multipole Method", *Parallel Processing for Scientific Computing,* SIAM Conference Proceedings, pages 213–222, 1988.

[10] Greengard, L. and Rokhlin, V., "A Fast Algorithm for Particle Simulations", *Journal of Computational Physics*, 73:325–348, 1987.

[11] Gürel, Levend, "Fast Radar Cross Section (RCS) Computation via the Fast Multipole Method", Technical Report BILUN/EEE/LG-9602, Bilkent University, October1996.

[12] Greengard, L., Carrier, J. and Rokhlin V., "A Fast Adaptive Multipole Algorithm for Particle Simulations" *SIAM Scientific and Statistical Computing*, 9(4):669–686, July 1988.

[13] James Leathrum F., "Parallelization of the Fast Multipole Algorithm: Algorithm and Architecture Design", PhD thesis, Duke University, 1993.

[14] James Leathrum F., and Board, A., "Mapping the Adaptive Fast Multipole Algorithm onto MIMD Systems", Technical Report, Duke University, April, 1992.

[15] James Demmel W., Higham Nicholas J., and Robert S. Schreiber. "Block LU Factorization", Unpublished Report February 1992.

[16] Karypis G and Kumar V., "A Fast and High Quality Multilevel Scheme For Partitioning Graphs", Technical report tr 95-035, Dept. of Computer Science, University of Minnesota, 1995. also available at http://www.cs.umn.edu/karypis/metis/metis.html

[17] Leighton F. T., *Introduction to Parallel Algorithms and Architectures : Arrays, Trees and Hypercubes*, Morgan Kaufmann Publishers, 1992.

[18] Lu, C.C. and Chew, W.C., "Fast Algorithm for Solving Hybrid Integral Equations", *IEEE Proceedings-II*, 140(6):455–460, December 1993.

[19] Nowak, Z. P., "Panel Clustering Technique for Lifting Potential Flows in the Three Dimensions", page 1360. Viewg-Verlag, Braunschewig, 1987.

[20] Rokhlin V., "Rapid Solution of Integral Equations of Scattering Theory in Two Dimensions" *Journal of Computational Physics*, 86(2):414–439, February 1989.

[21] Rohklin Vladimir, Coiffman Ronald and Wandzura Stephen, "The Fast Multipole Method for the Wave Equation : A Pedestrian Prescription", *IEEE Antennas and Propagation Magazine*, 35(3):7–12, June 1993.

[22] Stalzer Mark A., "Parallelizing the Fast Multipole Method for the Helmholz Equation", *Parallel Processing Letters*, vol. 5, pages 263–274, 1995

[23] Vipin Kumar, Gupta Anshul, Grama Anath and Karypis George, *Introduction to Parallel Computing, Desin and Analysis of Algorithms.* The Benjamin/Cummings Publishing Co. Inc., 1994.

[24] Vassiliou M. S., Rokhlin V., and Murphy W. D., "Acceleration methods for the iterative solution of electromagnetic scattering problem", *Radio Science*, 28(1):1–12, January-February 1993.

[25] Wilson, Gregory V., *Practical Parallel Programming Dictionary,* Unpublished book, 1992.