APPLICATION OF MASSIVELY PARALLEL ARCHITECTURE TO COMPUTATIONAL ELECTROMAGNETICS

by

XIANNENG SHEN

B.S., Chengdu Institute of Radio Engineering, 1982

M.S., Syracuse University, 1993

DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Electrical Engineering
in the Graduate School of Syracuse University

August, 1994

Approved ______________________
Professor Roger F. Harrington

Date ______________________

© Copyright 1994
Xianneng Shen

APPLICATION OF MASSIVELY PARALLEL ARCHITECTURE TO COMPUTATIONAL ELECTROMAGNETICS

by XIANNENG SHEN

ABSTRACT OF DISSERTATION

In this thesis, we discuss the development and implementation of computational electromagnetics simulations on massively parallel processing systems. The possibility of predicting radar cross section (RCS) for a full scale aircraft is discussed and demonstrated by combining the most advanced computational electromagnetics techniques and massively parallel processing technologies. Wilkes' and Cha's exact surface model and their basis function are used to develop numerical solutions for electromagnetic scattering problems involving arbitrarily shaped conducting bodies with and without lossy dielectric coatings.

The ParaMoM code—one of the most sophisticated and complicated software packages for electromagnetic scattering developed by Cha's group at Syracuse Research Corporation—is extended to treat arbitrarily shaped conducting bodies with lossy dielectric coatings. The parallel algorithms development of ParaMoM is discussed. The parallel ParaMoM, called ParaMoM-MPP, is implemented on three massively parallel architectures: the TMC CM-5, the Intel Paragon, and the IBM SP-1.

The accuracy of the ParaMoM-MPP code is discussed and demonstrated by comparing the numerical results with physical measurements. Efficiency of the parallel implementation is discussed. Portability of ParaMoM-MPP is discussed and tested by porting the ParaMoM-MPP code to different architectures. The scalability of the parallel implementation of each component of the ParaMoM-MPP code is discussed. The out-of-core algorithm is discussed as a method for solving large problems which require a large amount of memory exceeding that available in core.

This work demonstrates that parallel computing and advanced numerical techniques are equally important to successfully achieving full-scale aircraft RCS prediction. This thesis gives an example of the successful combining of state-of-the-art massively parallel processing technologies with state-of-the-art computational electromagnetic techniques.

Acknowledgements

I am most indebted to Professors Roger F. Harrington and Geoffery C. Fox, my academic and research advisors, for their encouragement, fruitful suggestions, invaluable and friendly guidance, and financial support. Without these, the success of the present work would not have been possible.

I would like to express my sincere gratitude to Dr. Chung-Chi Cha, Gerald E. Mortensen, and Debra L. Wilkes of Syracuse Research Corporation for their support and various useful discussions.

I am grateful to Dr. Joseph R. Mautz for his day-to-day consultation, taking the time to read this dissertation and to offer his suggestions. I greatly appreciate the courses taught by Dr. Arlon T. Adams, Dr. Jay K. Lee, and Dr. Ercument Arvas.

I am very grateful to Dr. Kim Mills and Gang Cheng for their useful suggestions on parallel implementation. I specially thank Mr. Gang Cheng for providing the data obtained on the Intel, IBM SP-1, and the network cluster. I thank Gerald E. Mortensen and Dr. Kenneth A. Hawick for their reading of the manuscript of the dissertation and offering suggestions. I am thankful for Mr. Jim Lauer providing EMCC testing results.

I thank the Northeast Parallel Architectures Center at Syracuse University, and the Numerical Aerodynamic Simulation (NAS) Facility at NASA Ames Research Center for providing extensive computer resources.

Finally, I would like to give my deepest thanks to my family. I am most grateful to my wife, Pingyan Zou, and my son, Kevin, for their patience and understanding of my frequent absence.

Contents

Acknowledgements iv

List of Tables viii

List of Figures ix

1 Introduction 1

1.1 Integral Equation Methods 4

1.2 Parallel Implementation of EM Scattering 9

2 EM Scattering from Conducting Bodies 11

2.1 Parametric Modeling and Basis Functions for a Parametric Surface . 11

2.1.1 Parametric Patch Model 12

2.1.2 Basis Functions 15

2.2 Electric Field Integral Equation Formulation 16

2.2.1 Derivation of the Electric Field Operator Equation 16

2.2.2 Numerical Formulation for Surface Current 18

2.3 Magnetic Field Integral Equation Formulation 25

2.4 Combined Field Integral Equation 30

3 EM Scattering from a Coated Conducting Body 33

3.1 Impedance Boundary Conditions for Scattering from a Conducting Body with a Thin Lossy Dielectric Coating 33

3.2 Numerical Solution of EFIE 37

4 Parallel Implementation 42

4.1 Introduction 42

4.1.1 Distributed Memory System 46

4.1.2 Parallelism and Performance Issues 47

4.2 Structure of the Code 50

4.2.1 The Parallel Programming Model and Approaches 54

4.3 Implementation 61

4.3.1 Parallel Setup 61

4.3.2 Precomputation 62

4.3.3 Moment Matrix Fill 63

4.3.4 Fill the Right-Hand Side Vectors 71

4.3.5 LU Factor and Solve 72

4.3.6 RCS Computation 74

4.3.7 Out-of-Core Algorithm 77

5 Performance and Numerical Results 81

5.1 Performance and Scalability Analysis 82

5.1.1 Performance Measurement for a Fixed Machine Size 82

5.1.2 Performance Measurement for a Fixed Problem Size 84

5.1.3 Scalability Analysis 95

5.2 Numerical Results 101

5.3 Discussion of Two Parallel Moment Method Codes 125

6 Conclusion 129

A A Normalized Local Area Coordinate System 137

B Definition of Scattering Angles 140

C Overview of Three MPP Architectures **143**
C.1 The CM-5 System Overview 143
C.2 The CM-5 at NAS 147
C.3 The Intel Paragon System at NAS 151
C.4 The IBM SP-1 System at ANL 154

Bibliography **158**

List of Tables

4.1 The elapsed time of writing the moment matrix to a SDA file using CMMD global write under CMMD synchronous sequential mode (recorded using the CMMD timer). 59

4.2 The elapsed time of reading the moment matrix from a SDA file using CM Fortran (SO mode) function (recoded by CM Fortran timer). . . 59

4.3 The time comparison between the out-of-core fill algorithm with $N_{out} = 10$ and the in-core fill algorithm on the CM-5. 80

5.1 The running time in seconds on a 32-node partition CM-5 83

5.2 The running time in seconds on a 512-node partition CM-5 83

5.3 The running time in seconds on an Intel 32-node Paragon, 512-node and 64-node Touchstone Delta. 85

5.4 The running time in seconds on the IBM 58-node and 32-node SP-1. . 85

5.5 N: matrix size, fill: matrix filling time in seconds, LU: matrix factor time in seconds 86

List of Figures

2.1 A special class of parametric surface 13

2.2 Triangulation of a parametric surface 14

2.3 Domain for a parametric surface basis function 16

2.4 Illustration of the relationship of $\vec{J}_n$ and $\vec{r} - \vec{r}\,'$ 28

2.5 Three sub-triangles generated from a testing point to avoid a singularity in the integration 31

3.1 Original problem of EM scattering from a conducting body with a dielectric coating 34

3.2 An external equivalent to the original problem 35

4.1 Flynn's classification of computer architectures 44

4.2 An example of a shared memory system 46

4.3 Distributed-memory architectures. 48

4.4 The sequential code structure 53

4.5 The pseudo code for the precomputation algorithm 64

4.6 Data decomposition for matrix fill implementation 68

4.7 The pseudo code of the parallel fill algorithm 69

4.8 Varying amounts of data can be written by nodes in *CMMD synchronous sequential* mode 70

4.9 The pseudo code for the fill RHS vectors algorithm 72

4.10 Divide and conquer algorithm for global summation 75

4.11 The pseudo code for the RCS computation algorithm 76

4.12 The pseudo code of the parallel out-of-core fill algorithm 79

5.1 Performance of matrix fill portion of code implemented on small CM-5 partitions for fixed small problems 88
5.2 Performance of matrix fill portion of code implemented on large CM-5 partitions for fixed problem sizes 89
5.3 Performance of matrix factor portion of code implemented on large CM-5 partitions for fixed problem sizes 90
5.4 Performance of matrix fill portion of code implemented on the Intel Paragon system 91
5.5 Performance of matrix factor portion of code implemented on the Intel Paragon System 92
5.6 Performance of matrix fill portion of code implemented on the Intel Touchstone Delta system 93
5.7 Performance of matrix factor portion of code implemented on the Intel Touchstone Delta system 94
5.8 Performance of matrix fill portion of code implemented on the IBM SP-1 system 96
5.9 Performance of matrix factor portion of code implemented on the IBM SP-1 system 97
5.10 The wedge cylinder with gap geometric model 102
5.11 The RCS with HH polarization of a wedge cylinder with gap 103
5.12 The NASA's half almond parametric geometry model 105
5.13 The RCS with HH polarization of a NAS almond 106
5.14 The RCS with VV polarization of a NASA almond 107
5.15 The single ogive parametric geometry model with symmetry 108
5.16 The RCS with HH polarization of a single ogive 109
5.17 The RCS with VV polarization of a single ogive 110
5.18 The double ogive parametric geometry model 111
5.19 The RCS with HH polarization of a double ogive 112
5.20 The RCS with VV polarization of a double ogive 113
5.21 The cone-sphere parametric geometry model 115

5.22 The RCS with HH polarization of a cone-sphere 116
5.23 The RCS with VV polarization of a cone-sphere 117
5.24 The cone-sphere with gap parametric geometry model 118
5.25 The RCS with HH polarization of a cone-sphere with gap 119
5.26 The RCS with VV polarization of a cone-sphere with gap 120
5.27 The quarter of rectangular parallelpiped parametric geometry model . 121
5.28 The RCS with HH polarization of a rectangular parallelpiped 122
5.29 The RCS with VV polarization of a rectangular parallelpiped 123
5.30 The RCS with HH polarization of a coated conducting sphere 124
5.31 Comparison of accuracy on a small sphere 127
5.32 Comparison of matrix size for a specified (0.5 dB) error tolerance . . 128
A.1 Definitions of areas used in defining area coordinates. 138
A.2 The local area system of coordinates. 139
B.1 Definition of azimuth, elevation, and RLOS angles 141
B.2 Definition of monostatic and bistatic angles for different RCS modes . 142
C.1 CM-5 network interface. 145
C.2 CM-5 data network with 16 nodes. 146

Chapter 1

Introduction

Electromagnetic radiation and scattering problems are old problems in the sense that they have been research topics for over 100 years. They are also new and challenging problems since there are still a lot of unsolved problems. Since the development of digital computer technology, computing electromagnetic scattering and radiation involving a complicated geometry became possible. Computational electromagnetics (CEM) based on advanced numerical technology and state-of-the-art computer technology becomes a very active research area in electromagnetic fields. Today, the major numerical methods in computational electromagnetics are the differential equation solver and the integral equation solver in both frequency and time domains. The finite element method is the most widely used method for differential equations in the frequency domain. The finite difference time domain method is a popular one for differential equations in the time domain. A popular numerical method in the past 30 years is the Method of Moments (MoM), which was proposed for electromagnetics by Harrington [1]. The method of moments is an integral equation solver. There are thousands papers published on these methods.

The problem we are going to tackle is how to predict the full-scale aircraft radar cross section (RCS) combining the state-of-the-art of CEM techniques and the state-of-the-art of massively parallel processing technologies. Practical RCS prediction using numerical methods has long been thought of as unrealistic. This is because numerical solutions, while exact in concept, demanded amounts of computation too large to accomplish in the past. The RCS prediction of a fighter-sized aircraft using MoM, for example, requires the solution of a matrix equation whose dimension can easily exceed 100,000. The impossibility of such computations also discouraged efforts to improve other aspects of CEM techniques.

Successful developments of massively parallel processing (MPP) technologes have moved us into a position from which the opportunity now looks much better for solving the above-mentioned problem. Parallel computing not only drastically improves speed, and promises much more, it also prompts new developments in CEM techniques by bettering the prospects of real problem solutions.

In this thesis, we discuss the development and implementation of computational electromagnetics techniques on massively parallel architectures. We focus on advanced numerical formulations and parallel implementation for electromagnetic scattering problems. The goal of this work is to demonstrate the possibility of predicting RCS for full-scale aircraft by applying efficient computational eletromagnetic techniques and massively parallel processing. The exact surface patch model, a parametric patch model proposed by Wilkes and Cha [2], is used for scattering from conducting bodies with or without dielectric coatings. Electric field, magnetic field, and combined field integral equations are derived for unknown surface currents. A sophisticated and complicated computer program package, called ParaMoM, has been developed by Cha's group at the Syracuse Research Corporation (SRC). ParaMoM utilizes the curvilinear surface patches in conjunction with a new type of basis function developed at SRC. We extend ParaMoM to treat electromagnetic scattering from conducting bodies coated with dielectric material. The ParaMoM code is parallelized on three MIMD (multiple instruction, multiple data) distributed memory systems. Thinking Machine Corporation's CM-5, the Intel Paragon, and the IBM SP-1 are representative of the state-of-the-art massively parallel processing architectures. The parallel ParaMoM code is called ParaMoM-MPP. The CM-5 implementation is discussed in detail and the difference of other two implementations is given when it is necessary. The Intel and IBM SP-1 implementations are done by Mr. Gang Cheng at the Northeast Parallel Architecture Center. This dissertation includes this for the purpose of comparison and completeness. The work porting the ParaMoM-MPP code to the network cluster is also done by Mr. Gang Cheng.

The performance, scalability, and portability of the ParaMoM-MPP code are discussed. Some of the Electromagnetic Code Consortium (EMCC) benchmark cases are computed and the results are in good agreement with the EMCC benchmark measurement data.

The thesis is organized into six chapters. The parametric patch model of the moment method formulation for scattering from arbitrarily shaped three dimensional conducting bodies is derived in Chapter 2. The parametric patch model and a set of basis functions proposed by Wilkes and Cha [2] are described in Section 2.1, the electric field integral equation (EFIE) formulation is derived in Section 2.2, the magnetic field integral equation (MFIE) formulation is developed in Section 2.3, and the combined field integral equation formulation is discussed in Section 2.4. The electric field integral equation for electromagnetic scattering from coated conducting bodies is given in Chapter 3. The impedance boundary condition is used to formulate the electric field integral equation. The parallel implementation is given in Chapter 4. The parallel algorithms for matrix fill and factor/solve are given in detail. An out-of-core matrix fill algorithm is discussed. In Chapter 5, we present not only performance measurements and numerical results but also a brief comparison between ParaMoM-MPP and PATCH, which is a parallel MoM code [43]. The conclusion is presented in Chapter 6.

In the rest of this chapter, we review numerical methods using integral equations and their parallel implementations. Particularly, in Section 1.1, we review surface patch development in the integral equation approach using different numerical techniques. In Section 1.2, parallel implementation of some integral equation methods is reviewed.

1.1 Integral Equation Methods

Since Harrington first proposed the method of moments (MoM) in 1967 [1], MoM has become one of the most important numerical methods using the integral equation approache in computational electromagnetics. The MoM of Harrington is a method which transforms a functional operator equation describing the physical problem into a matrix equation by first approximating the unknown functions by a set of expansion functions with a set of unknown coefficients and then performing a scalar (or symmetric) product on the operator equaiton with selected testing functions. Today, more than two and a half decades after Harrington's now well-known book [3] was published, the MoM has been enriched by many researchers and many new features have been added and many new application areas have been explored. Here, we only discuss MoM in scattering and radiation applications.

An integral equation can be derived for the induced current in the scatterers or for the equivalent current on the scatterers. When the unknowns are volume currents in the scatterer the associated integral equation is called the volume integral equation, whereas when the unknowns are surface currents or equivalent surface currents on the surface of the scatterer the associated integral equation is called the surface integral equation (or boundary integral equation). We can convert the integral equation into a matrix equation by discretizing the unknown currents. Since the unknown quantities to be solved for in an integral equation formulation are restricted to be on or within the scatterers, the number of unknowns in this formulation is generally smaller than the number of those found in the differential equation formulation. This is particularly true of the surface integral equation formulation, in which the number of unknowns can be much smaller than the number unknowns in differential equation formulations and volume integral equation formulations.

The MoM can be applied in both time domain and frequency domain. Principal differences in applying the method of moments in the two domains are primarily in the formulation and solution steps, and hence much of the software developed for one approach can be used for the other. In the time domain, the unknown equivalent current or field must be discretized in both time and space. Local equivalent induced currents depend only on the local excitation and, due to propagation delay effects, to the past history of the other equivalent currents on the object. Since the equivalent current can be locally updated without knowledge of the updated values of the remaining currents, the solution can proceed in a marching on-in-time fashion. The potentials used to compute time domain fields are retarded potentials, and hence one must store the history of the equivalent current distribution over the object for an interval of time equal to the longest transit time across the object.

Which approach to apply depends on the quantities of interest. It is easier to model dispersive material characteristics in the frequency domain, and relatively little additional expense is involved in obtaining the response due to multiple monochromatic excitations once the moment matrix is obtained and factored. However, nonlinear characteristics are more easily modeled in the time domain. In this thesis, only frequency domain formulations are considered.

One of the important issues in applying numerical methods is how to model the geometry of a scatterer of specified material. The geometry of an electromagnetic boundary value problem is defined by specifying the spatial dependence and electrical parameters of all materials. Let us review some common geometric models.

The wire-grid modeling approach has been remarkably successful in many problems, particularly those requiring the prediction of far-field quantities such as radiation patterns and radar cross sections [4]. In addition to resulting in a surface model that is easy to describe to the computer, the technique has the advantage that all numerically computed integrals are essentially one-dimensional.

Piecewise linear straight line segments are commonly used to approximate wires [5], the cross sections of two dimensional cylinders [6], and bodies of revolution [7, 8]. Only the nodal coordinates and the connectivity between nodes are needed to completely specify the geometry.

The most notable example of a wire-grid modeling code is the widely used Numerical Electromagnetics Code (NEC) [9] developed at Lawrence Livermore Laboratory as an extremely versatile general-purpose user-oriented code. The code can treat complex wire configurations which model either surfaces or multi-wire antennas in the frequency domain. The Livermore group has also developed the Thin Wire Time Domain (TWTD) code, which has similar capabilities for solving transient problems directly in the time domain. Apparently because of computer limitations, this code has not been used extensively to model surfaces, however.

Despite its simplicity and generality, the wire-grid modeling approach is not well-suited for calculating near-field and surface quantities such as surface current and input impedance. Other difficulties encountered in wire-grid modeling include the occasional presence of fictitious loop currents in the solution, difficulties with internal resonances [10], and the problem of relating computed wire currents to equivalent surface currents. The accuracy of wire-grid modeling has also been questioned on theoretical grounds [11]. These difficulties have provided incentives for developing the surface patch approach as an alternative to the wire-grid technique.

Surfaces are often modeled using planar rectangular and triangular elements, producing piecewise linear models of the surface [12, 13, 14, 15]. The elements are specified by enumerating their boundary edge vertices. The order of these vertices may be used to implicitly specify, via the right hand rule, the sense of the unit vector, normal to the face. On orientable surfaces, one should ensure that the normal vector is always on the same side of the surface; this is accomplished by requiring that the common edge between every pair of adjacent faces is traversed in opposite directions as one travels around the boundaries of the two faces in the sense of their orientation. In surface modeling, the position of a vertex relative to the others can be specified by listing all the vertices connected to it by means of boundary edges. The most popular method is that of Rao, Wilton, and Glisson [14], which uses flat triangles. Some most popular general purpose computer programs for electromagnetic scattering and radiation with 3-D arbitrarily-shaped surface are used RWG techniques [15, 16, 17].

The existence of artificial creases in such surface models leads to erroneous edge scattering. Because of this, and because a more efficient or accurate surface model is desirable for other reasons, a mechanism for putting current basis functions on curved surfaces is needed. The use of flat facets to model a nonplanar surface creates unnecessary man-made discretization errors in the solution. Such errors can be important, for example, in near-field calculations when the observation point is on or close to the object surface such that effects of surface roughness are more easily seen. Progress has recently been made in developing more precise geometric models which take the surface curvature into consideration [18, 19, 20, 21, 2, 22].

Graglia introduced a finite element type of parametric element in the method of moment analysis [18, 19]. He proposed parametric (curved) elements which are generated by distorting simple forms (such as triangles, rectangles, tetradrons, rectangular prisms, etc.) so as to obtain other elements of more flexible shape which better match the object to be approximated.

To demonstrate the practical usefulness of parametric elements, he developed a program, based on a point matching technique, to study scattering from penetrable cylinders of arbitrary shape. In a practical example, he showed the superiority of parametric elements in the method of moments solution of a volume integral equation. These elements permit a better geometrical description of the scatterer, reduce the number of unknowns, shorten computation times, and give results more accurate than those provided by the commonly used planar elements.

Sancer [20], reported that researchers at Northrop use a parametric element model in their MoM code to get accurate RCS predictions with a minimum number of surface patches. A pulse basis function on each parametric element and point matching are used for both MFIE and EFIE. The disadvantage of Sancer's approach is that the pulse basis functions had line charges at the patch boundaries.

At Hughes Research Laboratory, Wandzura [21] constructed basis functions for representing currents on curved surfaces using differential geometry. These basis functions are appropriate for method of moments solution of boundary integral equations. They maintain the essential properties of the basis functions of Rao-Wilton-Glisson (RWG) [14], while allowing higher order basis functions (more variables per patch). The use of these functions is expected to result in a large reduction in the computational resources required to solve a given problem for a fixed level of accuracy.

Wandzura claimed that his basis functions can be reduced to those of RWG. But we have not see his implementation yet.

The parametric model used in this report is the one proposed and developed by Wilkes and Cha [2] at Syracuse Research Corporation. Wilkes and Cha's parametric surface patch model is the exact geometric model. There is no discretization error introduced by the model. Wilkes and Cha have proposed a simple and efficient basis function which is defined in terms of curvilinear cooordinates on the parametric surface in question and conforms to its exact curvature. The parametric element is a curved triangle in physical space and a flat triangle with straight edges in the parametric space. Wilkes and Cha's basis functions have all the properties that a good basis function should have. Namely, these basis functions are linearly independent and capable of accurately representing the equivalent current on the surface. They are also simple to work with. They are discussed in detail in Chapter 2. Some comparison of performance between the popular RWG technique with flat facets and ParaMoM is presented in Chapter 5.

The integral equation formulation for computational electromagnetics is considerably aided by use of the electromagnetic equivalence principle. Harrington [3] has shown that it may be used to derive most of the practical formulations for both conductors and penetrable objects [3, 23, 24, 25]. It is now generally appreciated that use of the equivalence principle eliminates much of the tedium and possibility for error formerly associated with deriving integral equations from Green's theorems. Its many forms also suggest alternative formulations.

It is seen that a common thread in any moment method formulation is the computation of fields due to equivalent electric and magnetic current sources radiating in unbounded homogeneous regions. These fields are most conveniently represented in terms of magnetic vector, electric scalar, electric vector, and magnetic scalar potentials.

When the scatterer has axial symmetry, the MoM approaches can be simplified. The body of revolution method [7, 8, 27, 28, 23, 24, 29, 30] is a good example.

Above, we have briefly reviewed the MoM approach and given a little more detail about surface modeling.

1.2 Parallel Implementation of EM Scattering

Parallel architectures have been studied from user experience over a broad range of problem classes [31, 32, 33]. In this section, we only review the parallel implementation of EM scattering problems using the integral equation approach. The parallel processing architecture application to electromagnetic scattering has received attention since late in the last decade. JPL/Caltech led the research on Hypercube architecture application. Their work was reported in [34, 35, 36, 37, 38, 39, 40, 41, 42, 43]. The first code they have implemented is the Numerical Electromagnetics Code (NEC-2), developed at Lawrence Livermore National Laboratory. The NEC-2 code, which used the wire-grid modeling approach, is an extremely versatile general-purpose user-oriented code. The code can treat complex wire configurations which model either surfaces or multi-wire antennas in the frequency domain. The code was implemented on a JPL/Caltech Mark III Hypercube (Hypercube architecture is discussed in chapter 4). The Mark III Hypercube's configuration consists of 32 processing nodes [39]; each node has a pair of Motorola 68020 processors–one is the main application processor and the second is the communication processor. A Motorola 68881 floating point coprocessor is added for floating point operation, and a new floating point accelerator uses the Weitek chip set; each node has four megabytes (Mbytes) main memory with 128 kilobytes (Kbytes) cache memory. The Mark III Hypercube delivers about 1 to 14 megaflops (Mflops) per node in computation, 2.0 Mbytes per second per channel in synchronous communication and 0.5 Mbyte per second and per channel in asynchronous communication. The 32-node Mark III Hypercube can run cases in core which consist of up to 2400 equations. At the end of the last decade and the beginning of this decade, JPL/Caltech upgraded the Mark III Hypercube to a 128 node machine which can run cases in core which consist of 4800 unknowns [40, 41, 42, 38]. Beginning this decade, JPL/Caltech started to implement the PATCH code described in [15] on the 128-node Mark III Hypercube [36]. The PATCH code is a method of moments code which implements a discretization of the electric field integral equation (EFIE) for conducting objects of arbitrarily shaped surfaces. An object is modeled by a set flat triangular patches and Rao's basis functions [14]. For a small problem, the parallel algorithm implemented has been termed "trivial parallelization" because each processor executes identical code for varying excitations. In the parallelization of large problems, row decomposition is used and direct LU factorization is employed. The PATCH code is also implemented on the Intel iPSC Hypercube and Touchstone Delta systems. The Hypercube system has been upgraded to 64 processors, and the Delta system is a two-dimensional mesh (16×32) of 512 processors. Each processor has 16 Mbytes of RAM attached. During the first year's operation of the Intel Touchstone Delta system, Cwik [37] reported that it has solved for scattering from a conducting sphere with $ka = 33.40$ using the EFIE formulation out-of-core on a 512-node Delta machine.

We notice another parallel implementation of the integral equation method is a body of revolution code using the EFIE by Gedney [44] on Hypercube architectures. The parallel MoM algorithm in Gedney's code is based on the work presented by Gedney and Mittra [45] which was derived from original work done by Glisson and Wilton [46]. Column scattering decomposition is applied to map data onto the hypercube. Although this mapping scheme eliminates a significant amount of redundant integration computation, it requires a reshuffling of the matrix before the LU factorization is performed. They experienced that their parallel algorithm on a Coarse-Grained Hypercube has bad scalability, because the additional communication is required by reshuffling the matrix elements when the number of the processors becomes large. Gedney and Mittra have implemented their code not only on a Coarse-Grained Hypercube Mark III but on a Fine-Grained Hypercube (MIMD) nCUBE, which may employ up to 8K processors. In addition, Gedney and Mittra have implemented their code on a Fine-Grained SIMD Hypercube architecture, Thinking Machine's CM-2, which has 64K bit-serial processors.

Chapter 2

EM Scattering from Conducting Bodies

In this chapter, we deal with the application of electric, magnetic, and combined field integral equations (EFIE, MFIE, and CFIE) to analyze electromagnetic scattering from arbitrarily shaped three-dimensional perfectly conducting bodies in the frequency domain. In Section 2.1, a parametric surface model is described and the procedure of parametric patch generation is given. A set of basis function which was first proposed by Wilkes and Cha [2] is defined on a parametric sub-domain. In Section 2.2, the electric field integral equation (EFIE) formulation is presented. In Section 2.3, the magnetic field integral equation formulation is derived. In Section 2.4, the combined field integral equation is given.

2.1 Parametric Modeling and Basis Functions for a Parametric Surface

In this Section, the surface of the scatterer is decomposed into a set of parametric triangles which have curved edges in the physical space. A set of basis functions is defined on the parametric triangular sub-domain.

2.1.1 Parametric Patch Model

Many electromagnetic scattering modeling approaches represent the scatterer's surface by a set of flat patches. The most popular one is that of Rao, Wilton, and Glisson [14], which uses flat triangles. The disadvantages of the flat patch model are in the following two important areas. One is that using flat patches to discretize a curved surface is an approximate surface model which introduces discretization error in the solution. Next, the flat patch surface model requires a relatively larger number of elements to represent a complex surface to a desired accuracy than would a parametric surface model. The motivation for using curved patches rather than flat ones is to avoid any surface modeling errors. This allows one to get accurate RCS predictions with a minimum number of surface patches. Figures 5.31 and 5.32 in Chapter 5 demonstrate that the parametric surface model is superior to the flat patch surface model. As the density of the surface patches increases, the difference in accuracy between curvilinear patches and flat patches (facets) decreases. However, for the large 3D target in which we are interested, it is almost impossible to increase the number of surface patches because of the limitation of both the physical memory and the CPU time of the modern computer.

We will only consider a special class of surfaces which only depend on two surface parameters. This class of surface is very useful in the course of RCS prediction applications. Figure 2.1.1 shows an example of this type of surface, where u and v are the surface parameters, and $\vec{r}$ is a position vector which can be represented as a function of these two surface parameters as:

$$\vec{r}(u,v) = \vec{r}_0 + u\triangle\vec{r} + R(u)n(v) \tag{2.1}$$

where $R(u)$ has the following form

$$R(u) = \sqrt{Au^2 + Bu + C} \tag{2.2}$$

and A, B, and C are constants, and $n(v)$ is a simple function of v. The tangent

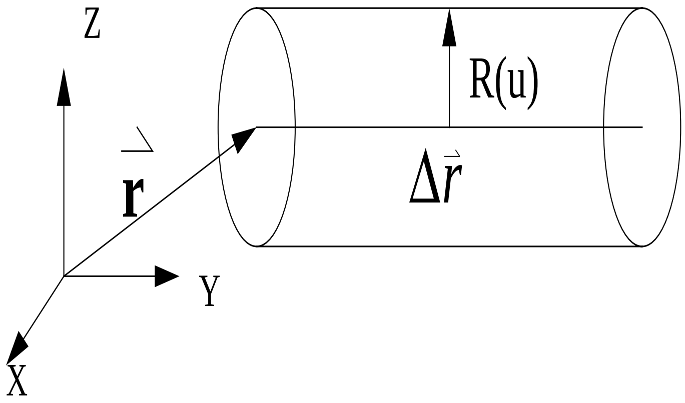


Figure 2.1: A special class of parametric surface

vectors of the position vector $\vec{r}(u,v)$ along constant u and v are defined by:

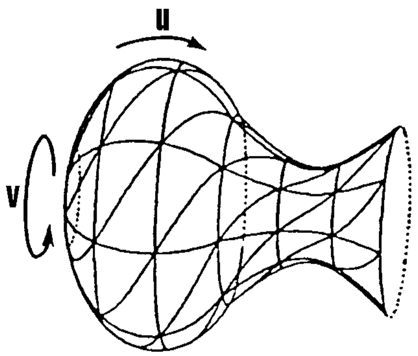
$$\begin{aligned} \frac{\partial\vec{r}}{\partial u} &= \vec{r}_u = h_u\hat{u} \\ \frac{\partial\vec{r}}{\partial v} &= \vec{r}_v = h_v\hat{v} \end{aligned} \tag{2.3}$$

where h_u and h_v are, respectively, the metrical coefficients or the scalar factors along u and v, and $\hat{u}$ and $\hat{v}$ are the unit vectors of u and v, respectively. The Jacobian of the surface is defined as a function of u and v by:

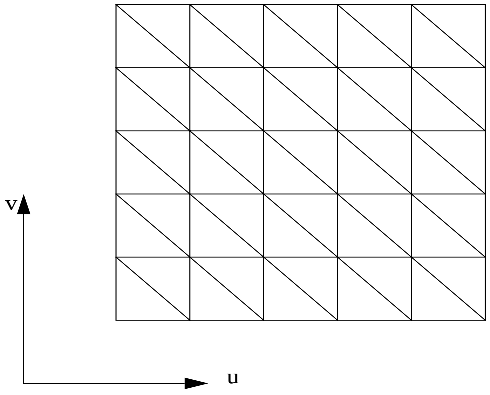
$$\mathsf{J}(u,v) = \mid \frac{\partial\vec{r}}{\partial u} \times \frac{\partial\vec{r}}{\partial v} \mid = h_u h_v \mid \hat{u} \times \hat{v} \mid \tag{2.4}$$

To generate this model, lines along constants u and v are first used to divide the surface into four sided patches. Then curved diagonal lines are used to form curved triangular patches.

The geometric model utilizes the parametric description to map the physical surface into the parametric space shown in Figure 2.2, where the physical surface is decomposed into a collection of curved triangles as shown in Figure 2.2(a). Figure 2.2(b) shows a two-dimensional parametric space (u-v) representation of the physical surface. In the physical space, the surface is closely modeled by an appropriate set of curved triangles. In the corresponding two-dimensional parametric (u-v) space, these triangles will have straight edges.



(a). Physical Space



(b). Parametric (u-v) Space

Figure 2.2: Triangulation of a parametric surface

2.1.2 Basis Functions

It is interesting to define basis functions on the parametric space that conform to the curvature of the physical surface. The basis function proposed by Wilkes and Cha [2] is illustrated in Figure 2.3. The domain of the basis function is a pair of adjacent triangles, T_n^- and T_n^+. The basis function is defined by

$$\vec{J}_n(u,v) = \begin{cases} \frac{1}{2A_n^+ \mathsf{J}(u,v)}[(u-u_1)h_u\hat{u} + (v-v_1)h_v\hat{v}], & \text{in } T_n^+; \\ \frac{-1}{2A_n^- \mathsf{J}(u,v)}[(u-u_4)h_u\hat{u} + (v-v_4)h_v\hat{v}], & \text{in } T_n^-; \\ 0, & \text{otherwise}; \end{cases} \tag{2.5}$$

where (u_i, v_i) are local vertices in the parametric u-v space. $A_n^\pm$ is the area of the triangle $T_n^\pm$.

This basis function has the following desirable properties:

- There are no line charges along the boundary (including the common edge of the conjoined triangle pair T_n^- and T_n^+)
- the component normal to the shared edge is continuous, and thus, does not generate a line charge accumulation.
- The surface divergence of the basis function, which is proportional to the surface charge density associated with the basis element is

$$\nabla \bullet \vec{J}_n = \begin{cases} \frac{1}{2A_n^+ \mathsf{J}(u,v)} & \text{in } T_n^+ \\ \frac{-1}{2A_n^- \mathsf{J}(u,v)} & \text{in } T_n^- \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

- When the patch dimensions become small compared to the radius of curvature, this basis function approaches linearly the RWG (Rao, Wilton, and Glisson [14]) basis function for a flat triangle.
- This basis function is defined in terms of a general surface parameterization, and it is not tied to any specific surface parameterization. This feature has

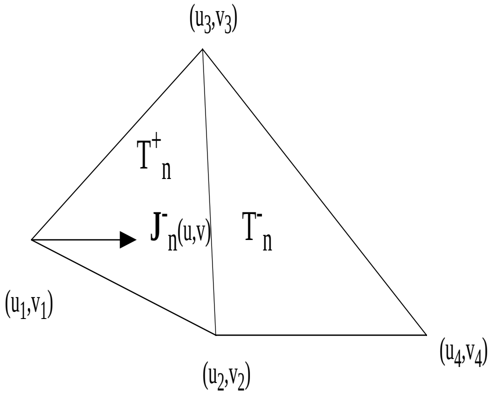


Figure 2.3: Domain for a parametric surface basis function

allowed for simple modular inclusion of several different parameterizations in the method of moment procedure.

After reviewing the above properties of the function defined in (2.5), we will select this function as a basis function to approximate the electric currents induced on the surface of scatterers. This basis function will be used throughout this thesis.

2.2 Electric Field Integral Equation Formulation

The integro-differential equation for the current distribution based on the electric field operator is called the electric field integral equation (EFIE). In this section, we are interested in the electric field operator equation. The method of moments is applied to the electric field boundary value equation to obtain a set of linear equations for the induced electric surface current on the surface of a scatterer. We shall give a derivation of the operator equation in Section 2.2.1, and we will concentrate on evaluating the so-called impedance matrix numerically in section 2.2.2.

2.2.1 Derivation of the Electric Field Operator Equation

Let S denote the surface of a perfectly conducting scatterer with unit normal vector $\hat{n}$. S may be either open or closed. The incident electric field $\vec{E}^{inc}$ is due to an impressed source in the absence of the scatterer. The boundary condition is such that the sum of the incident, $\vec{E}^{inc}$, and the scattered, $\vec{E}^{s}$, electric fields has no tangential component on the perfectly conducting body surface, i.e.,

$$\vec{E}^{s}_{tan}(\vec{J}) + \vec{E}^{inc}_{tan} = 0 \qquad \text{on } S \tag{2.7}$$

where the subscript "tan" denotes the components tangential to the surface S. $\vec{J}$ is the electric current which is induced on the surface due to the incident field. If S is open, we regard $\vec{J}$ as the vector sum of the currents on opposite sides of S.

The scattered electric field can be represented by the so-called vector potential and the scalar potential which are produced by the surface current, as below:

$$\vec{E}^{s}(\vec{J}) = -j\omega\vec{A} - \nabla\Psi \tag{2.8}$$

The magnetic vector potential, $\vec{A}$, and the electric scalar potential Ψ are given by [47]:

$$\vec{A}(\vec{r}) = \mu_0 \int_S \vec{J}(\vec{r}\,')\frac{e^{-jk|\vec{r}-\vec{r}\,'|}}{4\pi \mid \vec{r}-\vec{r}\,' \mid} ds' \tag{2.9}$$

$$\Psi(\vec{r}) = \frac{1}{\epsilon_0} \int_S \sigma(\vec{r}\,')\frac{e^{-jk|\vec{r}-\vec{r}\,'|}}{4\pi \mid \vec{r}-\vec{r}\,' \mid} ds' \tag{2.10}$$

An $exp(j\omega t)$ time dependence is assumed and is suppressed, and $k = \omega\sqrt{\mu_0\epsilon_0} = 2\pi/\lambda$, where λ is the wavelength. The permeability and permittivity of the surrounding medium are μ_0 and ϵ_0, respectively, and $\vec{r}$ and $\vec{r}\,'$ are the arbitrarily located observation point and source point, respectively. The surface charge density σ is related to the surface divergence of $\vec{J}$ through the equation of continuity,

$$\nabla_s \bullet \vec{J} = -j\omega\sigma. \tag{2.11}$$

where ∇_s is the surface divergence operator. Substituting (2.8) into eq (2.7), an integro-differential equation for $\vec{J}$ is given by

$$(j\omega\vec{A} + \nabla\Psi)_{tan} = \vec{E}^{inc}_{tan}, \quad \vec{r} \text{ on } S. \tag{2.12}$$

With $\vec{A}$ and Ψ given by eqs (2.9) and (2.10), (2.12) is the so-called electric field integral equation (EFIE). In the next subsection, the method of moments is applied to obtain a matrix equation for the unknown surface current.

2.2.2 Numerical Formulation for Surface Current

Let a basis function $\vec{J}_n$ as defined in the previous section be associated with the n^{th} non-boundary edge of the curved triangulated structure. Surface current exists on both sides of the structure for an open surface. The unknown current $\vec{J}$, being solved for in the integral equation, is the vector sum of the currents on opposite sides of S. At boundaries of S, the component of this vector sum normal to the boundary must vanish due to continuity of the current; therefore, we need not define basis functions associated with boundary edges. The current on S can be approximated as a linear combination of the basis functions with a set of unknown coefficients.

$$\vec{J} \approx \sum_{n=1}^{N} I_n \vec{J}_n \tag{2.13}$$

where N is the total number of non-boundary edges, and I_n is the unknown coefficient associated with the n^{th} basis function $\vec{J}_n$. Since there may be up to three non-boundary edges in a triangle, so there will be up to three non-zero basis functions within each triangular face. To convert (2.12) into a matrix equation, we choose the expansion functions as testing functions. The symmetric product for any two vector functions $\vec{f}$ and $\vec{g}$ as given by Harrington [3] is

$$< \vec{f}, \vec{g} > \equiv \int_S \vec{f} \bullet \vec{g} ds. \tag{2.14}$$

Testing (2.12) with $\vec{J}_m$ yields

$$j\omega < \vec{A}, \vec{J}_m > + < \nabla\Psi, \vec{J}_m > = < \vec{E}^{inc}, \vec{J}_m > \tag{2.15}$$

where $\vec{J}_m$ is defined in (2.5), $m = 1, 2, \cdots, N$. Utilizing a surface calculus identity [48], the last term of left hand side in (2.15) can be rewritten as

$$< \nabla\Psi, \vec{J}_m > = - < \Psi, \nabla_s \bullet \vec{J}_m > \tag{2.16}$$

The continuity equation (2.11) is substituted into (2.10) to make the current be the only unknown for the scattered field. Inserting the current expansion (2.13) into eqs (2.9) and (2.10) then into (2.12) yields N linear equations for N unknown coefficients. It may be written in a matrix form as:

$$[Z_{mn}][I_n] = [V_m^e] \tag{2.17}$$

where $[Z_{mn}]$ is an $N \times N$ square matrix which is called the "moment matrix" or the "generalized impedance" matrix. $[I_n]$ and $[V_m^e]$ are column vectors. $[I_n]$ is called the "generalized current" vector, and $[V_m^e]$ is named the "generalized voltage" vector. The n^{th} element of $[I_n]$ is I_n. The m^{th} element of $[V_m^e]$ is given by:

$$V_m^e = \int_{S_m} \vec{J}_m \bullet \vec{E}_{tan}^{inc} ds. \tag{2.18}$$

The evaluation of the generalized impedance matrix elements follows from equations (2.14) through (2.16). Thus, utilizing the approximation (2.13),

$$Z_{mn} = j\omega\mu_0 \int_{S_m} \vec{J}_m \bullet \vec{a}_n ds + \frac{1}{j\omega\epsilon_0} \int_{S_m} (\nabla_s \bullet \vec{J}_m)\psi_n ds \tag{2.19}$$

where S_m denotes the domain of $\vec{J}_m$, and $\vec{a}_n$ and ψ_n are given by:

$$\vec{a}_n = \frac{\vec{A}_n}{\mu_0} \tag{2.20}$$

and

$$\psi_n = -j\omega\epsilon_0\Psi_n \tag{2.21}$$

where $\vec{A}_n$ and Ψ_n are the magnetic vector potential and the scalar potential due to the current $\vec{J}_n$ on the n^{th} edge, Substituting the n^{th} expansion function into (2.9) and (2.10), (2.20) and (2.21) can be rewritten as:

$$\vec{a}_n(\vec{r}) = \int_{S_n} \vec{J}_n(\vec{r}\,') \frac{e^{-jk|\vec{r}-\vec{r}\,'|}}{4\pi \mid \vec{r} - \vec{r}\,' \mid} ds', \tag{2.22}$$

and

$$\psi_n(\vec{r}) = \int_{S_n} [\nabla_s' \bullet \vec{J}_n(\vec{r}\,')] \frac{e^{-jk|\vec{r}-\vec{r}\,'|}}{4\pi \mid \vec{r} - \vec{r}\,' \mid} ds'. \tag{2.23}$$

where S_n denotes the domain of $\vec{J}_n$ and ∇_s' is the surface divergence operator on the primed variables.

We note that each matrix element of Z_{mn} is associated with a pair of non-boundary edges m and n. However, the domains of the integrals and locations of the observation points are associated with the faces attached to these edges. For each pair of triangular patches, contributions to the interactions between up to nine different combinations of source and field basis functions must be computed. Each source-field basis function interaction corresponds to a single matrix element. Much of the information required to compute the interaction between the source and field basis functions is only related to geometry. This information is the same regardless of which basis function is currently considered. Once computed, the geometry information between a pair of patches may be used to obtain the contributions to a maximum of nine different matrix elements.

To evaluate these surface integrals, we first transfer the curved triangular domain to a flat triangular parametric domain. Secondly, we transform the (u, v) parametric space to a local system of area coordinates (see Appendix A) with the corresponding triangle. Finally, the numerical integration formula for a triangular region in (see Chapter 8, [49]) are used to evaluate these surface integrals. Implementing this idea, we can rewrite Z_{mn} in terms of each pair of faces as:

$$Z_{mn} = Z_{mn}^{++} + Z_{mn}^{+-} + Z_{mn}^{-+} + Z_{mn}^{--} \tag{2.24}$$

where Z_{mn}^{pq} is the contribution from testing over T_m^p on the electric field due to the electric current $\vec{J}_n$ on T_n^q, and p and q are either $+$ or $-$ signs.

$$Z_{mn}^{pq} = j\omega\mu_0 \int_{T_m^p} \vec{J}_m \bullet \vec{a}_n^q ds + \frac{1}{j\omega\epsilon_0} \int_{T_m^p} [\nabla_s \bullet \vec{J}_m] \psi_n^q ds \tag{2.25}$$

where $\vec{a}_n^q$ and ψ_n^q are given by:

$$\vec{a}_n^q = \frac{\vec{A}_n^q}{\mu_0} \tag{2.26}$$

and

$$\psi_n^q = -j\omega\epsilon_0 \Psi_n^q \tag{2.27}$$

where $\vec{A}_n^q$ and Ψ_n^q are, respectively, the magnetic vector potential and the electric scalar potential produced by the part of $\vec{J}_n$ on the patch T_n^q. Both $\vec{a}_n^q$ and ψ_n^q can be numerically computed after mapping the curved physical space to the (u, v) parametric space with parametric description in (2.1) and Jacobian in (2.4). Then, we transform the (u, v) space to a local coordinate system. After substituting the basis function and the expression for the surface element, $\vec{a}_n^q$ is given by

$$\vec{a}_n^q = q\int_{T_n^q} \frac{1}{2A_n^q \mathsf{J}(u',v')}[(u'-u_i)h_{u'}\hat{u}+(v'-v_i)h_{v'}\hat{v}]\frac{e^{-jk|\vec{r}-\vec{r}\,'|}}{4\pi \mid \vec{r}-\vec{r}\,' \mid}\mathsf{J}(u',v')du'dv' \tag{2.28}$$

where (u_i, v_i) is (u_1, v_1) if q is $+$ and (u_4, v_4) if q is $-$ (see Fig. 2.3), and A_n^q is the area of the triangle T_n^q. Since $\vec{r}$ and $\vec{r}\,'$ are, respectively, $\vec{r}(u, v)$ and $\vec{r}(u', v')$ where $\vec{r}(u, v)$ is given by (2.1), $q\vec{a}_n^q$ can be rewritten as:

$$q\vec{a}_n^q = q\vec{a}_n^q(u,v) = \int_{T_n^q} \vec{f}_{nq}(u,v,u',v')du'dv' \tag{2.29}$$

where $\vec{f}_{nq}(u, v, u', v')$, a vector function, is defined by

$$\begin{aligned} \vec{f}_{nq}(u,v,u',v') &= \frac{1}{2A_n^q}[(u'-u_i)\vec{r}_u(u',v')+(v'-v_i)\vec{r}_v(u',v')] \\ &\quad \cdot \frac{e^{-jk|\vec{r}(u,v)-\vec{r}(u',v')|}}{4\pi \mid \vec{r}(u,v)-\vec{r}(u',v') \mid} \end{aligned} \tag{2.30}$$

where (2.3) is used to replace $h_{u'}\hat{u}$ by $\vec{r}_u(u', v')$ and $h_{v'}\hat{v}$ by $\vec{r}_v(u', v')$.

Similarly, ψ_n^q can be written as

$$\begin{aligned} \psi_n^q &= q\int_{T_n^q} \frac{1}{2A_n^q \mathsf{J}(u',v')}\frac{e^{-jk|\vec{r}-\vec{r}\,'|}}{4\pi \mid \vec{r}-\vec{r}\,' \mid}\mathsf{J}(u',v')du'dv' \\ &= q\int_{T_n^q} g_{nq}(u,v,u',v')du'dv' \end{aligned} \tag{2.31}$$

where $g_{nq}(u, v, u', v')$, a scalar function of (u, v, u', v'), is defined by

$$g_{nq}(u,v,u',v') = \frac{1}{2A_n^q}\frac{e^{-jk|\vec{r}(u,v)-\vec{r}(u',v')|}}{4\pi \mid \vec{r}(u,v)-\vec{r}(u',v') \mid} \tag{2.32}$$

Equations (2.29) and (2.31) can be evaluated by Gaussian quadrature after transforming the (u', v') coordinates to a local system of area coordinates (ξ, ζ, γ) within T_n^q. The details of the local system definition are given in Appendix A. Then, $a_n^q(u, v)$ and ψ_n^q are given by:

$$\vec{a}_n^q(u,v) = q2A_n^q \int_0^1 \int_0^{1-\xi} \vec{f}_{nq}(u,v,u'(\xi,\zeta),v'(\xi,\zeta))\, d\zeta d\xi \tag{2.33}$$

$$\psi_n^q(u,v) = q2A_n^q \int_0^1 \int_0^{1-\xi} g_{nq}(u,v,u'(\xi,\zeta),v'(\xi,\zeta))\, d\zeta d\xi \tag{2.34}$$

where only ξ and ζ appear in the above equations because $u'(\xi,\zeta)$ and $v'(\xi,\zeta)$ are given in (A.4) where γ is a linear combination of ξ and ζ. Note that direct application of a technique for numerical technique over a triangular region [49] allows equations (2.33) and (2.34) to be evaluated as:

$$\vec{a}_n^q(u,v) = q2A_n^q \sum_{i=1}^{N_s} w_i \; \vec{f}_{nq}(u,v,u'(\xi_i,\zeta_i),v'(\xi_i,\zeta_i)) \tag{2.35}$$

$$\psi_n^q(u,v) = q2A_n^q \sum_{i=1}^{N_s} w_i \; g_{nq}(u,v,,u'(\xi_i,\zeta_i),v'(\xi_i,\zeta_i)) \tag{2.36}$$

where N_s is the total number of integration points, w_i is the weight for the i^{th} integration point (ξ_i,ζ_i), where w_i and (ξ_i,ζ_i) are given in Table 8.2, [49], and $u'(\xi,\zeta)$ and $v'(\xi,\zeta)$ are given in (A.4). The testing integrals over T_m^p in (2.25) can also be evaluated numerically in the same way. For $T_m^p \neq T_n^q$, both $\vec{a}_n^q$ and ψ_n^q are well behaved. Substituting (2.35) and (2.36) into (2.25) and evaluating the testing integrals using the same procedure as for the potential integrals, the numerical representation of Z_{mn}^{pq} is given, for $T_m^p \neq T_n^q$, by

$$\begin{aligned} Z_{mn}^{pq} &= j\omega\mu_0 p2A_m^p \sum_{j=1}^{N_t} w_j \; \vec{f}_{mp}(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j)) \bullet \vec{a}_n^q(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j)) \\ &+ \frac{p2A_m^p}{j\omega\epsilon_0} \sum_{j=1}^{N_t} w_j \; g_{mp}(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j))\psi_n^q(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j)) \end{aligned} \tag{2.37}$$

where N_t is the total number of integration points on T_m^p chosen according to the accuracy requirement, and w_j is the weight associated with the integration point at (ξ_j,ζ_j). The quantities w_j and (ξ_j,ζ_j) are given in Table 8.2, [49], $u(\xi_j,\zeta_j)$ and $v(\xi_j,\zeta_j)$ are given in (A.4), p is either $+$ or $-$, and A_m^p is the area of the triangle T_m^p. The vector function $\vec{f}_{mp}$ and the function g_{mp} are defined by

$$\begin{aligned} \vec{f}_{mp}(u,v) &= \frac{1}{2A_m^p}[(u-u_i)\vec{r}_u(u,v) + (v-v_i)\vec{r}_v(u,v)] \\ g_{mp}(u,v) &= \frac{1}{2A_m^p} \end{aligned} \tag{2.38}$$

For $T_m^p = T_n^q$, the integrands of both the vector and scalar potentials integrals are singular. A term will be added and subtracted from each integrand. The term to be added and subtracted must have the same singular behavior when $\vec{r}\,'$ approaches to $\vec{r}$ and can be analytically evaluated. Thus, the result of the magnetic vector potential $\vec{a}_n^q$ in (2.25) is presented after adding and subtracting a selected term.

$$\begin{aligned} \vec{a}_n^q &= \int_{T_n^q} [\frac{\vec{J}_n(u',v')\mathsf{J}(u',v')e^{-jkR}}{4\pi R} - \frac{\vec{J}_m(u,v)\mathsf{J}(u,v)}{4\pi R_s}] du'dv' \\ &\quad + \frac{\vec{J}_m(u,v)\mathsf{J}(u,v)}{4\pi} \int_{T_n^q} \frac{1}{R_s} du'dv' \end{aligned} \tag{2.39}$$

Adding and subtracting a term which has a $1/R$ singularity when R approaches zero in the integraand of the scalar potential integral ψ_n^q, one has

$$\psi_n^q = \frac{q}{2A_n^q}\{\int_{T_n^q} du'dv'\,[\frac{e^{-jkR}}{4\pi R} - \frac{1}{4\pi R_s}] + \int_{T_n^q} du'dv'\,\frac{1}{4\pi R_s}\} \tag{2.40}$$

where q is either $+$ or $-$, and R is the distance between the testing point $\vec{r}$ and the source point $\vec{r}\,'$. R_S is an approximation to R when $\vec{r}$ is very close to $\vec{r}\,'$, which can be expressed in terms of a Taylor series approximation. Here, R and R_s are given by

$$R = \mid \vec{r} - \vec{r}\,' \mid \tag{2.41}$$

$$\begin{aligned} R_s &= \lim_{\vec{r}\to\vec{r}\,'} \mid \vec{r} - \vec{r}\,' \mid \approx \mid \vec{r}_u(u'-u) + \vec{r}_v(v'-v) \mid \\ &= \sqrt{r_u^2(u'-u)^2 + r_v^2(v'-v)^2 + 2(\vec{r}_u \bullet \vec{r}_v)(u'-u)(v'-v)} \end{aligned} \tag{2.42}$$

Now, R_S has the same behavior as R near the singularity. Hence, the integrand of the first integral on the right-hand side of each of (2.39) and (2.40) is well-behaved on entire T_n^q region, so these integrals can be evaluated numerically with the technique in [49]. The second integral on the right-hand side of each of (2.39) and (2.40) can be evaluated analytically, and is discussed in Appendix C of [16] and [50]. Therefore, Z_{mn}^{pq} for $T_m^p = T_n^q$ is given by

$$Z_{mn}^{pq} = j\omega\mu_0 p 2A_m^p \sum_{j=1}^{N_t}\{w_j\,\vec{f}_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j))$$

$$\begin{aligned}
&\bullet[\vec{a}^{q}_{n1}(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j))+\vec{a}^{q}_{n2}(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j))]\} \\
+\ &\frac{p2A^p_m}{j\omega\epsilon_0}\sum_{j=1}^{N_t}\{w_j\ g_{mp}(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j)) \\
&\cdot[\psi^{q}_{n1}(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j))+\psi^{q}_{n2}(u(\xi_j,\zeta_j),v(\xi_j,\zeta_j))]\} \qquad (2.43)
\end{aligned}$$

where p is either $+$ or $-$ depending on whether the patch of the testing function J_m that contributes to Z^{pq}_{mn} resides on T^+_m or T^-_m. The vector function $\vec{f}_{mp}$ and the scalar function g_{mp} are defined in (2.38). $\vec{a}^q_{n1}$, $\vec{a}^q_{n2}$, ψ^q_{n1}, and ψ^q_{n2} are given by

$$\vec{a}^{q}_{n1}(u,v)=q2A^q_n\sum_{i=1}^{N_s}w_i\ \vec{f}^{1}_{nq}(u,v,u'(\xi_i,\zeta_i),v'(\xi_i,\zeta_i)) \qquad (2.44)$$

$$\vec{a}^{q}_{n2}(u,v)=q\vec{f}^{2}_{nq}(u,v) \qquad (2.45)$$

and

$$\psi^{q}_{n1}(u,v)=q2A^q_n\sum_{i=1}^{N_s}w_i\ g^{1}_{nq}(u,v,u'(\xi_i,\zeta_i),v'(\xi_i,\zeta_i)) \qquad (2.46)$$

$$\psi^{q}_{n2}(u,v)=qg^{2}_{nq}(u,v) \qquad (2.47)$$

where N_s is the total number of integration points and w_i is the weight for the i^{th} integration point (ξ_i,ζ_i). Here w_i and (ξ_i,ζ_i) are given in (Table 8.2 [49]). Furthermore, $u'(\xi_i,\zeta_i)$ and $v'(\xi_i,\zeta_i)$ can be found in (A.4).

The vector functions $\vec{f}^1_{nq}$ and $\vec{f}^2_{nq}$ in (2.44) and (2.45) are defined by

$$\begin{aligned}
\vec{f}^{1}_{nq}(u,v,u',v') &= \frac{1}{2A^q_n}[(u'-u_l)\vec{r}_u(u',v')+(v'-v_l)\vec{r}_v(u',v')]\frac{e^{-jkR}}{4\pi R} \\
&- \frac{1}{2A^q_n}[(u-u_l)\vec{r}_u(u,v)+(v-v_l)\vec{r}_v(u,v)]\frac{1}{4\pi R_s} \qquad (2.48)
\end{aligned}$$

and

$$\vec{f}^{2}_{nq}(u,v)=\frac{1}{2A^q_n}[(u-u_i)\vec{r}_u(u,v)+(v-v_i)\vec{r}_v(u,v)]\int_{T^q_n}\frac{du'\,dv'}{4\pi R_s(u,v,u',v')} \qquad (2.49)$$

where (u_l,v_l) is (u_1,v_1) if q is $+$ and (u_4,v_4) if q is -. And the scalar functions in (2.46) and (2.47) are defined by

$$g^{1}_{nq}(u,v,u',v')=\frac{1}{2A^q_n}[\frac{e^{-jkR}}{4\pi R}-\frac{1}{4\pi R_s}] \qquad (2.50)$$

and

$$g_{nq}^{2}(u,v) = \frac{1}{2A_n^q} \int_{T_n^q} \frac{du'\,dv'}{4\pi R_s} \tag{2.51}$$

The analytical evaluations of the integrals in 2.49 and 2.51 are given in [50].

2.3 Magnetic Field Integral Equation Formulation

In this section, the magnetic field integral equation (MFIE) is derived for a conducting scatterer. It is well-known that the MFIE applies only to closed bodies, so that throughout this section we assume that the object has no boundary edges. The vector basis functions given in Section 2.1, are used for both the expansion and testing functions in the numerical solution of the MFIE.

Let S denote the surface of a perfectly conducting scatterer with unit normal vector $\hat{n}$. The incident magnetic field $\vec{H}^{inc}$ is due to an impressed source in the absence of the scatterer. The scatterer is in a homogeneous space characterized by a pair of parameters (μ_0, ϵ_0), where μ_0 is the inductivity or permeability and ϵ_0 is the capacitivity or permittivity. The result of enforcing the boundary condition on the magnetic field is given by

$$\hat{n} \times (\vec{H}^{inc} + \vec{H}^{s}) = 0 \qquad \text{on } S^{-} \tag{2.52}$$

where $\hat{n}$ is an outward unit normal vector on S, S^{-} is the surface is just inside of S, and $\vec{H}^{inc}$ and $\vec{H}^{s}$ are the incident and scattered magnetic fields, respectively. The tangential component of the scattered magnetic field can be expressed as a limit for observation points $\vec{r}$ not on an edge, (see [48] and [51])

$$\begin{aligned} \hat{n} \times \vec{H}^{s} &= \lim_{\vec{r} \to S} \hat{n} \times \nabla \times \vec{A} \\ &= -\frac{\vec{J}}{2} + \hat{n} \times \int_{S} (\vec{J} \times \nabla' G)\, ds', \end{aligned} \tag{2.53}$$

where $\vec{J}$ is the induced electric surface current on S, and the integral on the right hand side in (2.53), with the field point exactly on S, is interpreted as the Cauchy principal value. G is free space Green's function, and ∇' is the gradient operator on the primed coordinates. The Green's function and its gradient are given below as

$$G = \frac{e^{-jkR}}{4\pi R} \tag{2.54}$$

and

$$\nabla' G = (\vec{r} - \vec{r}\,')\frac{(1 + jkR)e^{-jkR}}{4\pi R^3} \tag{2.55}$$

where k is the wave number as defined in the previous section, R is the distance between the source point and the observation point which is $R = | \vec{r} - \vec{r}\,' |$, and $\vec{r}$ approaches S from the interior. Substituting (2.53) into (2.52), we obtain the magnetic field integral equation:

$$\frac{\vec{J}}{2} - \hat{n} \times \int_S (\vec{J} \times \nabla' G)\, ds' = \hat{n} \times \vec{H}^{inc} \tag{2.56}$$

In order to apply the method of moments, the surface of the scatterer is decomposed into a set of curved triangular patches using a parametric description of the surface. The procedure of the parametric surface model generation has been described in Section 2.1. The next step after surface modeling is to define a set of basis functions which are used to approximate the surface current. Here, the basis functions defined in (2.5) are to be used as expansion functions on the parametric surface. Then, the electric surface current can be approximated as a linear combination of the expansion functions with a set of unknown coefficients as in (2.13). Substituting (2.13) into (2.56) gives an integral equation with N unknown coefficients. The method of moments allows one to select a set of testing functions which are used to test (2.56) with a symmetric product which is defined in (2.14). When the testing functions are the same as the expansion functions, the procedure is called the Galerkin procedure. This procedure gives

$$\sum_{n=1}^{N} I_n [< \vec{J}_m, \frac{\vec{J}_n}{2} > - < \vec{J}_m, \hat{n} \times \int_S (\vec{J}_n \times \nabla' G)\, ds' >] = < \vec{J}_m, \hat{n} \times \vec{H}^{inc} > \tag{2.57}$$

where $\vec{J}_m$ is the m^{th} testing function and $m = 1, 2, \cdots, N$. I_n is the unknown coefficient associated with the n^{th} basis function $\vec{J}_n$.

The N linear equations for N unknowns in (2.57) can be rewritten in a matrix equation as

$$[L_{mn}][I_n] = [V_m^m] \tag{2.58}$$

where $[I_n]$ and $[V_m^m]$ are column vectors and $[L_{mn}]$ is an $N \times N$ square matrix. The n^{th} element of $[I_n]$ is the unknown coefficient associated with the n^{th} expansion function. The element of the m^{th} row and the n^{th} column of the matrix $[L_{mn}]$ and the element of the m^{th} row of the vector $[V_m^m]$ are given by

$$L_{mn} =< \vec{J}_m, \frac{\vec{J}_n}{2} > - < \vec{J}_m, \hat{n} \times \int_S (\vec{J}_n \times \nabla' G)\, ds' > \tag{2.59}$$

and

$$V_m^m =< \vec{J}_m, \hat{n} \times \vec{H}^{inc} >= \int_S ds\, \vec{J}_m(\vec{r}) \bullet [\hat{n}(\vec{r}) \times \vec{H}^{inc}(\vec{r})] \tag{2.60}$$

From the definition of the symmetric product in (2.14), (2.59) can be rewritten as

$$\begin{aligned} L_{mn} &= \frac{1}{2} \int_S ds\, \vec{J}_m(\vec{r}) \bullet \vec{J}_n(\vec{r}) \\ &\quad - \int_S ds\, \vec{J}_m(\vec{r}) \bullet \hat{n}(\vec{r}) \times \int_S ds' \, [\vec{J}_n(\vec{r}\,') \times \nabla' G(\vec{r}, \vec{r}\,')] \end{aligned} \tag{2.61}$$

Note that when the testing function and the expansion function reside on the same patch the singularity contribution is the same for both the flat patch and the curved patch. In other words, there is a $\vec{J}_n/2$ term when $\vec{r} = \vec{r}\,'$ for both the flat patch and the curved patch. However, the contribution for the principal value integral is different. In the flat patch case, the current vector $\vec{J}_n$ is always on the same plane as $\vec{r} - \vec{r}\,'$, so that $\hat{n} \times \vec{J}_n \times \nabla' G = 0$ (see [16]). However, it is not the case for a curved patch. Due to the surface curvature the current vector $\vec{J}_n$ on the patch is not always in the same plane as $\vec{r} - \vec{r}\,'$, so that $\hat{n} \times \vec{J}_n \times \nabla' G \neq 0$, as shown in Figure 2.4. Thus, the principal value integral has a singular integrand when the testing patch is also the source patch. That may be the only disadvantage of this model for MFIE. Fortunately, this integral is easily evaluated..

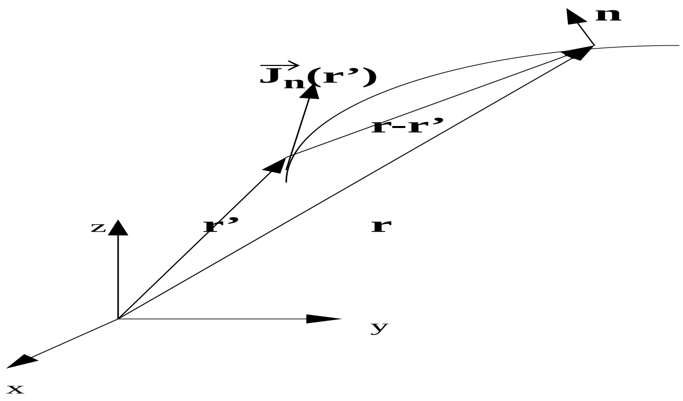


Figure 2.4: Illustration of the relationship of $\vec{J}_n$ and $\vec{r}-\vec{r}'$

Following the procedure in the previous section, we intend to compute the elements of L_{mn} sequentially by source-field patch pairs for all integrals. It will avoid the costly and inefficient recomputation of an identical integral up to nine times which would result if the elements of L_{mn} were computed sequentially by basis functions. As with the EFIE, it is convenient to write all the required integrals in terms of integrals over a source-field patch pair, as

$$L_{mn} = L_{mn}^{++} + L_{mn}^{+-} + L_{mn}^{-+} + L_{mn}^{--} \tag{2.62}$$

where L_{mn}^{pq} is the computation from testing over T_m^p the magnetic field due to the part of the electric current $\vec{J}_n$ on T_n^q, p and q are either $+$ or $-$ signs, and L_{mn}^{pq} is given by

$$\begin{aligned} L_{mn}^{pq} &= -\int_{T_m^p} ds \vec{J}_m \bullet \hat{n} \times \int_{T_n^q} ds' \vec{J}_n \times \nabla' G \\ &+ \begin{cases} \frac{1}{2}\int_{T_m^p} \vec{J}_m \bullet \vec{J}_n ds, & T_m^p = T_n^q \\ 0, & T_m^p \neq T_n^q \end{cases} \end{aligned} \tag{2.63}$$

For convenience, let $\vec{B}_n^q$ be the inner integral of the second term on the right-hand side of (2.63). It can be expressed as

$$\begin{aligned} \vec{B}_n^q &= \int_{T_n^q} [\vec{J}_n(\vec{r}') \times \nabla' G(\vec{r},\vec{r}')] \mathrm{J}(u',v') du' dv' \\ &= q \int_{T_n^q} \vec{F}_{nq}(u,v,u',v') du' dv' \end{aligned} \tag{2.64}$$

where the vector function $\vec{F}_{nq}(u, v, u', v')$ is defined after substituting (2.5) and (2.55) into the integral, as

$$\begin{aligned}\vec{F}_{nq}(u, v, u', v') &= \frac{[1 + jkR]e^{-jkR}[(u' - u_i)\vec{r}_u(u', v') + (v' - v_i)\vec{r}_v(u', v')]}{2A_n^q 4\pi R^3} \\ &\quad \times \frac{[\vec{r}(u, v) - \vec{r}(u', v')]}{2A_n^q 4\pi R^3} \qquad (2.65)\end{aligned}$$

Here $R = \mid \vec{r}(u, v) - \vec{r}(u', v') \mid$, and (u_i, v_i) is (u_1, v_1) when q is $+$ and (u_4, v_4) when q is $-$. We transform the parametric coordinates to a local system of area coordinates (ξ, ζ, γ) within T_n^q, so (2.64) can be rewritten as

$$\vec{B}_n^q = q 2A_n^q \int_0^1 \int_0^{1-\xi} \vec{F}_{nq}(u, v, u'(\xi, \zeta), v'(\xi, \zeta)) d\zeta d\xi \qquad (2.66)$$

where $u'(\xi, \zeta)$ and $v'(\xi, \zeta)$ are given by (A.4). Directly applying the numerical integration technique for a triangular region in [49] to (2.66) gives

$$\vec{B}_n^q(u, v) = q 2A_n^q \sum_{i=1}^{N_s} w_i \ \vec{F}_{nq}(u, v, u'(\xi_i, \zeta_i), v'(\xi_i, \zeta_i)) \qquad (2.67)$$

where N_s is the number of points where the integrand is sampled on the triangle T_n^q, and w_i is the weight corresponding to the integration point (ξ_i, ζ_i). N_s, w_i, and (ξ_i, ζ_i) are given in (Table 8.2, [49]).

Similarly, the testing integral in (2.63) can also be treated in the same way as above. The final numerical equation for L_{mn}^{pq} when $T_m^p \neq T_n^q$ is then given by

$$\begin{aligned}L_{mn}^{pq} &= -p 2A_m^p \sum_{j=1}^{N_t} w_j \ \vec{f}_{mp}(u(\xi_j, \zeta_j), v(\xi_j, \zeta_j)) \\ &\quad \bullet [\hat{n}(u(\xi_j, \zeta_j), v(\xi_j, \zeta_j)) \times \vec{B}_n^q(u(\xi_j, \zeta_j), v(\xi_j, \zeta_j))] \qquad (2.68)\end{aligned}$$

where $\vec{f}_{mp}(u, v)$ is given in (2.38) and $\vec{B}_n^q(u, v)$ is given in (2.67). N_t, w_j, ξ_j, and ζ_j have the same meanings as in (2.37), and $u(\xi_j, \zeta_j)$ and $v(\xi_j, \zeta_j)$ are given in (A.4).

When $T_m^p = T_n^q$, the integrand of the integral over T_n^q in (2.63) is singular at the testing point. There are two ways to treat this singularity. One is, as with EFIE, to subtract and add an analytically integrable function with the same singularity as the integrand. Another is to divide the patch into three sub-patches for which the testing point is a common vertex and then apply numerical integration directly on these sub-patches. Numerically integrating over these sub-patches, we obtain

$$L^{pq}_{mn} = \frac{1}{2}\int_{T^p_m} \vec{J}_m \bullet \vec{J}_n ds - \int_{T^p_m} ds \vec{J}_m \bullet \hat{n} \times \left(\int_{T^q_{n1}} ds' + \int_{T^q_{n2}} ds' + \int_{T^q_{n3}} ds'\right)\vec{J}_n \times \nabla' G \quad (2.69)$$

where T^q_{n1}, T^q_{n2}, and T^q_{n3} are shown in Figure 2.5. Since the testing point is a vertex of these three sub-patches and vertices are never picked as integration points, the result of numerical integration over the sub-patches will be finite and, hopefully, accurate. The numerical integration technique used in the previous section and this section is applied to (2.69). To avoid repeating work, the same notation as in the previous section will be used. Thus, for $T^p_m = T^q_n$, the L^{pq}_{mn} is given by

$$\begin{aligned} L^{pq}_{mn} &= pqA^p_m \sum_{j=1}^{N_t} w_j \; [\vec{f}_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \bullet \vec{f}_{nq}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \\ &\quad /\mathsf{J}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j))] \\ &- p2A^p_m \sum_{j=1}^{N_t} w_j \; [\vec{f}_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \bullet \hat{n}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \\ &\quad \times\{\vec{B}^q_{n1}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) + \vec{B}^q_{n2}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \\ &\quad +\vec{B}^q_{n3}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j))\}] \end{aligned} \quad (2.70)$$

where $\vec{f}_{nq}$ is defined on T^q_n in the same way that $\vec{f}_{mp}$ is defined on T^p_m. Because $T^p_m = T^q_n$, $\vec{f}_{nq}$ is $\vec{f}_{mp}$ of (2.38) with the vertex (u_i, v_i) replaced by the vertex of T^q_n opposite the common edge of the two triangles where $\vec{J}_n$ exists. For $i = 1, 2, \text{and } 3$, $\vec{B}^q_{ni}$ is the contribution to $\vec{B}^q_n$ of (2.64) due to integration over T^q_{ni}.

2.4 Combined Field Integral Equation

One of the most common integral equation deficiencies is failure to have a unique solution at certain discrete frequencies. At these frequencies, there exist nontrivial solutions of the source-free (homogeneous) form of the integral equation. It has been shown theoretically that neither the H-field equation nor the E-field equation has

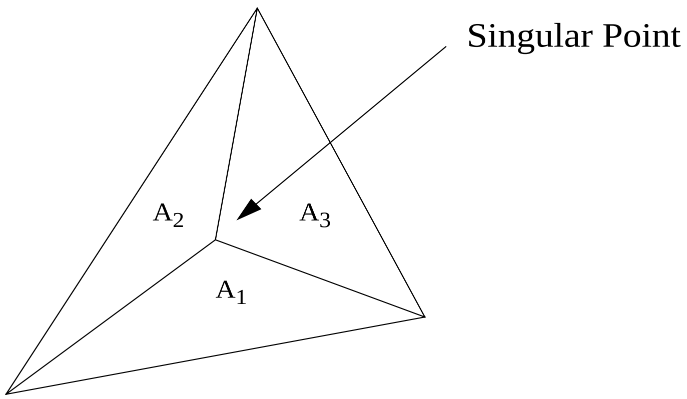


Figure 2.5: Three sub-triangles generated from a testing point to avoid a singularity in the integration

unique solutions for the current on a conducting body at frequencies corresponding to resonant frequencies of the region enclosed by the conducting surface, but the combined-field equation does have a unique solution [53],[52].

In this section, the scatterer is an arbitrarily shaped perfectly conducting body with a closed surface S. $\vec{J}$ denotes the electric surface current induced on S by the incident field $(\vec{E}^{inc}, \vec{H}^{inc})$. This current satisfies (2.7) and (2.52). The question is whether (2.7) alone is sufficient to determine $\vec{J}$, whether (2.52) alone is sufficient, or whether both are necessary. The answer is given by Mautz [53]. In [53], Mautz has proven that the solution $\vec{J}$ to (2.52) is not unique for values of k (wave number) at which the equations

$$\hat{n} \times \vec{H}^s(\vec{J}) = 0 \qquad \text{just inside } S \tag{2.71}$$

$$\nabla \times \nabla \times \vec{H}^s(\vec{J}) = k^2 \vec{H}^s(\vec{J}) \qquad \text{inside } S \tag{2.72}$$

which are valid when there is no incident field, admit a nontrivial solution. This solution $\vec{J}$ is called a magnetic cavity mode. Applying the duality theory, one concludes that the solution $\vec{J}$ to (2.7) is not unique for the same value of k.

The combined field formulation, which is a linear combination of (2.52) and (2.7), is given by

$$-\hat{n}\times\vec{H}^{s}(\vec{J})-\frac{\alpha}{\eta}\vec{E}^{s}_{tan}(\vec{J})=\hat{n}\times\vec{H}^{inc}+\frac{\alpha}{\eta}\vec{E}^{inc}_{tan}\quad\text{just inside } S \tag{2.73}$$

The solution of (2.73) is unique and satisfies both (2.7) and (2.52) whenever α is a positive real number (see [53]).

Since (2.73) is the linear combination of (2.7) and (2.52) with a relative weight α, the method of moments formulation obtained from (2.73) is the same linear combination of (2.17) and (2.58). Hence,

$$\{[L_{mn}]+\frac{\alpha}{\eta}[Z_{mn}]\}[I_n]=[V^{m}_{m}]+\frac{\alpha}{\eta}[V^{e}_{m}] \tag{2.74}$$

where all matrices and column vectors have the same meaning as in Section 2.2 and 2.3, and α is a constant. From experience, α should be between 0.2 and 1.0.

Chapter 3

EM Scattering from a Coated Conducting Body

In this chapter we discuss the electromagnetic scattering from an arbitrarily shaped perfectly conducting body either partially or fully covered by a thin layer of lossy material.

It is well known that the radar cross section of a conducting body can be reduced if it is coated with an electrically or magnetically lossy layer. There are many applications where lossy materials are applied as coatings to metallic bodies in recent years [54, 55, 56, 57]. Fighter jets and stealth aircraft are good examples.

3.1 Impedance Boundary Conditions for Scattering from a Conducting Body with a Thin Lossy Dielectric Coating

In this section, the scattering of a plane electromagnetic wave by a 3-D perfectly conducting body with a dielectric material coating is studied by the parametric method of moments described in the previous chapter with an impedance boundary condition. The problem can be formulated in terms of various integral equations derived from

,

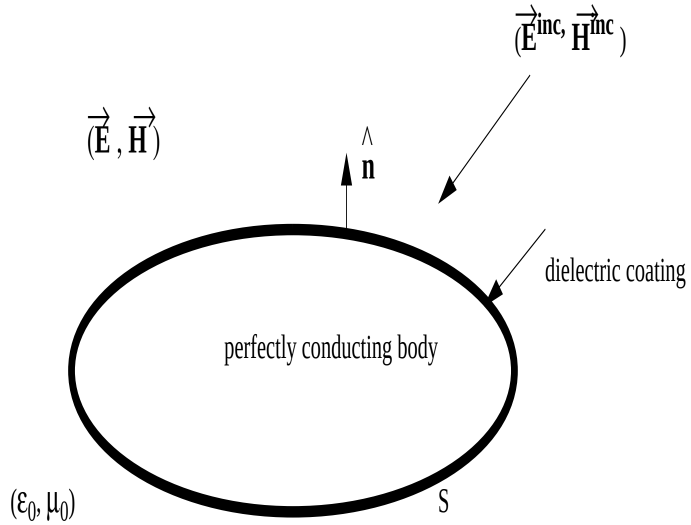


Figure 3.1: Original problem of EM scattering from a conducting body with a dielectric coating

the Leontovich [58] impedance boundary condition (IBC). There are many papers on IBC [59, 60, 61, 62, 63]. This approximation makes integral equation formulations of the problem nearly as simple as those for perfectly conducting bodies without coating. When using IBC, one can take advantage of the derivation in the previous chapter for using IBC.

Let S denote the closed surface of a three-dimensional perfectly conducting body with an infinitely thin layer of lossy dielectric material coating shown in Figure 3.1 where $\hat{n}$ is the outward unit vector normal on S and $(\vec{E}^{inc}, \vec{H}^{inc})$ are incident fields which are produced by some impressed sources $(\vec{J}^{imp}, \vec{M}^{imp})$ in the absence of the scatterer. The material coating on the perfectly conducting body is characterized by a pair of complex parameters (μ, ϵ), where ϵ is the permittivity and μ is the permeability. The space outside the scatterer is filled with the homogeneous material with permeability μ_0 and permittivity ϵ_0.

In solving the problem, it is often useful to apply the equivalence principle (Chapter 3, [47]) using equivalent electric and magnetic surface currents to represent the

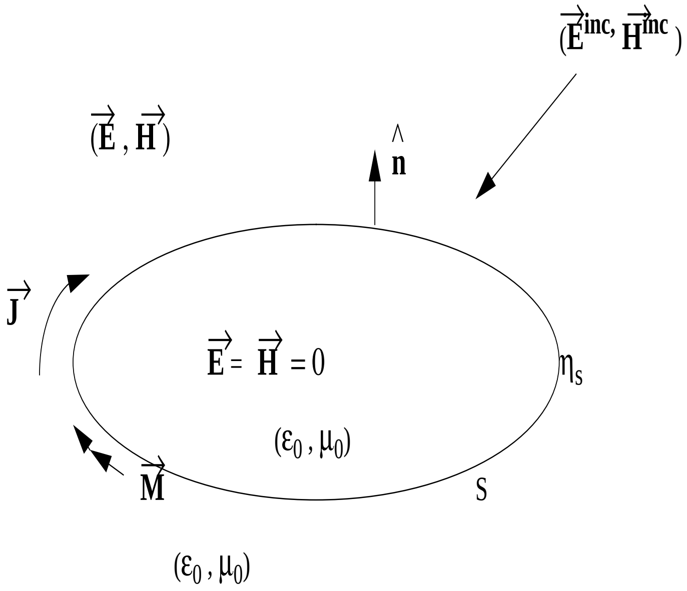


Figure 3.2: An external equivalent to the original problem

scatterer. If the task is to find the exterior field only, an exterior equivalent problem can be shown in Figure 3.2, where the fields external to the scatterer can be considered equivalent (to those of the original problem) due to the electric ($\vec{J}$) and magnetic ($\vec{M}$) surface current densities on S which are given by

$$\vec{J} = \hat{n} \times \vec{H} \qquad \text{on } S^+ \tag{3.1}$$

$$\vec{M} = \vec{E} \times \hat{n} \qquad \text{on } S^+ \tag{3.2}$$

where S^+ is the external surface of the dielectric coating.

Although the exterior fields must be unique, there are many sets of equivalent currents and interior fields which will give rise to the correct exterior fields in general scatterers. It is natural to let the interior field be the null field, since the perfectly conducting body is inside S. The Leontovich impedance boundary condition on S implies that only the electric and magnetic fields external to the scatterer are relevant and their relationship is a function of the material constitution (here, surface impedance) of the scatterer. As shown in Figure 3.2, the electric and magnetic fields are zero inside S, and the electric and magnetic fields outside S are related by [63]

$$\vec{E}_{tan} = \eta_0 \eta_s (\hat{n} \times \vec{H}) \tag{3.3}$$

The dual form of the IBC is

$$\vec{H}_{tan} = \frac{-1}{\eta_0 \eta_s} (\hat{n} \times \vec{E}) \tag{3.4}$$

where η_0 is the intrinsic impedance of free space which is given by $\eta_0 = \sqrt{\mu_0/\epsilon_0} \approx 377\Omega$; η_s is the relative surface impedance.

The total electric field is the vector sum of the incident electric field and the scattered electric field. The scattered electric field produced by the surface currents $(\vec{J}, \vec{M})$ can be expressed in terms of the magnetic and electric vector potentials and the electric scalar potential, and the total electric field is given by

$$\vec{E}(\vec{r}) = \vec{E}^{inc} - j\omega\vec{A}(\vec{r}) - \nabla\Psi(\vec{r}) - \nabla \times \frac{1}{\epsilon_0}\vec{F}(\vec{r}) \tag{3.5}$$

where $\vec{A}(\vec{r})$ and $\Psi(\vec{r})$ are the magnetic vector potential and the electric scalar potential, respectively, given by (2.9) and (2.10) in the previous chapter. $\vec{F}(\vec{r})$ is the electric vector potential given by

$$\vec{F}(\vec{r}) = \epsilon_0 \int_S ds' \vec{M}(\vec{r}\,')G(\vec{r}, \vec{r}\,') \tag{3.6}$$

where $G(\vec{r}, \vec{r}\,')$ is the free space Green's function given in the previous chapter. Substituting (3.5) and (3.1) into (3.3) yields

$$\eta_0\eta_s\vec{J} + \{j\omega_0\vec{A} + \nabla\Psi + \frac{1}{\epsilon_0}\nabla \times \vec{F}\}_{tan} = \vec{E}^{inc}_{tan} \quad \text{on } S^+ \tag{3.7}$$

Either of the boundary conditions in (3.3) and (3.4) the following simple relationship between the electric surface current and the magnetic surface current

$$\vec{M}(\vec{r}) = -\eta_0\eta_s(\hat{n} \times \vec{J}(\vec{r})) \tag{3.8}$$

With $\vec{M}(\vec{n})$ given by (3.8), (3.7) is the so-called electric field integral equation when the impedance boundary condition exists. In the next section, we will apply the parametric method of moments to solve for the electric current.

3.2 Numerical Solution of EFIE

There are two equations for two unknowns. The surface currents will be obtained after solving the equations together. Using (3.8) the curl of the electric vector potential (3.6) can be expressed as

$$\frac{1}{\epsilon_0}\nabla \times \vec{F}(\vec{r}) = \frac{-\eta_0\eta_s}{2}\vec{J} + \int_S ds'\vec{M}(\vec{r}\,') \times \nabla' G(\vec{r},\vec{r}\,') \tag{3.9}$$

where $\nabla' G(\vec{r},\vec{r}\,')$ is the divergence of the Green's function with respect to the primed coordinates. Since $\vec{r}$ is on S, the integral on the right-hand side of the above equation must be interpreted in the Cauchy principal value sense.

The surface current will be computed using the parametric method of moments described in the previous chapter. To do so, first the surface S is decomposed into a set of curved triangular patches. Secondly, the unknown current on S is approximated as a linear combination of the basis functions $\vec{J}_n$ in (2.5) with a set of unknown coefficients I_n in (2.13). Using the symmetric product defined in (2.14) to test the result of substituting (3.9) into (3.7) with $\vec{J}_m$ gives

$$\begin{aligned} < \vec{E}^{inc}_{tan}, \vec{J}_m > \;&=\; \frac{1}{2} < \eta_0\eta_s\vec{J}, \vec{J}_m > + j\omega < \vec{A}, \vec{J}_m > \\ &\quad + < \nabla\Psi, \vec{J}_m > + < \int_S ds'\,\vec{M} \times \nabla' G,\ \vec{J}_m > \end{aligned} \tag{3.10}$$

where $\vec{J}_m$ is defined in (2.5) for $m = 1, 2, \cdots, N$. The electric charge is obtained from the electric current according to the continuity equation (2.11). Substituting the current expansion (2.13) into (3.10) yields the following matrix equation for the unknown electric current coefficients

$$[Z_{mn}][I_n] = [V_m] \tag{3.11}$$

where $[I_n]$ and $[V_m]$ are column vectors, and $[Z_{mn}]$ is an $N \times N$ square matrix. The n^{th} element of the vector $[I_n]$ is I_n, the unknown coefficient associated with the n^{th} expansion function. The n^{th} element of the vector $[V_m]$ is given by

$$V_m = \int_S ds\,\vec{J}_m(\vec{r}) \bullet \vec{E}^{inc}_{tan}(\vec{r}) \tag{3.12}$$

The element of the m^{th} row and the n^{th} column of the matrix $[Z_{mn}]$ is given by:

$$\begin{aligned} Z_{mn} &= \frac{1}{2} < \eta_0 \eta_s \vec{J}_n,\ \vec{J}_m > + j\omega\mu_0 < \vec{a}_n,\ \vec{J}_m > \\ &+ \frac{1}{j\omega\epsilon_0} < \psi_n,\ \nabla_s \bullet \vec{J}_m > + < \int_S ds'\, \vec{M}_n(\vec{r}\,') \times \nabla' G,\ \vec{J}_m > \end{aligned} \tag{3.13}$$

where $\vec{a}_n$ and ψ_n are given in (2.22) and (2.23), respectively. $\vec{M}_n$ is the magnetic current related to the n^{th} electric current expansion by (3.8):

$$\vec{M}_n = -\eta_0 \eta_s (\hat{n} \times \vec{J}_n) \tag{3.14}$$

Applying the same technique as stated in Chapter 2, we intend to compute sequentially by faces all the vector and scalar potential integrals associated with each observation-face and source-face combination, to avoid the costly and inefficient recomputation of identical integrals which would result if the elements of Z_{mn} were computed sequentially by edges. The numerical integration for a triangular region in [49] is applied to evaluate the integrals in (3.13) after transforming coordinates to a local system of area coordinates. To implement this idea, it is necessary to rewrite the element of Z_{mn} in (3.13) in terms of each pair of faces as

$$Z_{mn} = Z_{mn}^{++} + Z_{mn}^{+-} + Z_{mn}^{-+} + Z_{mn}^{--} \tag{3.15}$$

where Z_{mn}^{pq} is the contribution from testing over T_m^p the electric field due to the parts of $\vec{J}_n$ and $\vec{M}_n$ on T_n^q, and p and q are either + or − signs. Z_{mn}^{pq} can be further divided into two parts, as

$$Z_{mn}^{pq} = Z_{mn}^{pqe} + Z_{mn}^{pqm} \tag{3.16}$$

where Z_{mn}^{pqe}, the part of Z_{mn}^{pq} originally contributed by the electric surface current, arises from the second and third terms on the right-hand side of equation (3.10). It is given by

$$Z_{mn}^{pqe} = j\omega\mu_0 \int_{T_m^p} \vec{J}_m \bullet \vec{a}_n^q ds + \frac{1}{j\omega\epsilon_0} \int_{T_m^p} ds\, (\nabla_s \bullet \vec{J}_m) \psi_n^q \tag{3.17}$$

where $\vec{a}_n^q$ and ψ_n^q are given in (2.28) and (2.31). Similarly, Z_{mn}^{pqm}, the part of Z_{mn}^{pq} originally contributed by the magnetic surface current, arises from the first and fourth terms on the right-hand side of equation 3.10. It is given by

$$\begin{aligned} Z_{mn}^{pqm} &= \int_{T_m^p} ds\, \vec{J}_m(\vec{r}) \bullet \int_{T_n^q} ds'\, \vec{M}_n(\vec{r}\,') \times \nabla' G(\vec{r}, \vec{r}\,') \\ &+ \left\{ \begin{array}{ll} 0, & T_m^p \neq T_n^q \\ \frac{\eta_0}{2} \int_{T_m^p} ds \vec{J}_m(\vec{r}) \bullet \eta_s(\vec{r}) \vec{J}_n(\vec{r}), & T_m^p = T_n^q \end{array} \right. \end{aligned} \tag{3.18}$$

Comparing the expression of Z_{mn}^{pqe} in (3.17) with the expression in (2.25), they are exactly the same. Therefore, the numerical evaluation of Z_{mn}^{pqe} in (3.17) is going to be exactly the same as that of (2.25) described in Chapter 2. Therefore, we will only use the result produced in Chapter 2 and leave the derivation out here. The comparison between the expression of Z_{mn}^{pqm} in (3.18) and that of L_{mn}^{pq} in (2.63), shows that there is only a little bit of difference in the inner integral of the second term on the right-hand side of them. Here, we only focus on the numerical evaluation of the second term on the right-hand side of (3.18).

As with MFIE for a perfect conductor in the previous chapter, let $\vec{C}_n^q$ be the inner integral of the second term on the right hand side of (3.18). It can be written in terms of the (u, v) coordinates, as:

$$\begin{aligned} \vec{C}_n^q(u, v) &= \int_{T_n^q} \vec{M}_n(\vec{r}\,') \times \nabla' G(\vec{r}, \vec{r}\,') \mathsf{J}(u', v') du'\, dv' \\ &= \int_{T_n^q} \vec{D}_{nq}(u, v, u', v') du'\, dv' \end{aligned} \tag{3.19}$$

where $\mathsf{J}(u', v')$ is the Jacobian defined in (2.4), and the vector function $\vec{D}_{nq}(u, v, u', v')$ is defined as

$$\vec{D}_{nq}(u, v, u', v') = \vec{M}_n(u', v') \times \nabla' G(u, v, u', v') \mathsf{J}(u', v') \tag{3.20}$$

where

$$\vec{M}_n(u', v') = -q \frac{\eta_0 \eta_s(u', v')}{2A_n^q \mathsf{J}(u', v')} \{\vec{n}(u', v') \times [(u' - u_i)\vec{r}_u(u', v') + (v' - v_i)\vec{r}_v(u', v')]\} \tag{3.21}$$

and

$$\nabla' G(u, v, u'v') = \frac{[\vec{r}(u, v) - \vec{r}(u', v')](1 + jk \mid \vec{r}(u, v) - \vec{r}(u', v') \mid) e^{-jk|\vec{r}(u,v) - \vec{r}(u',v')|}}{4\pi \mid \vec{r}(u, v) - \vec{r}(u', v') \mid^3} \tag{3.22}$$

where A_n^q denotes the area of the triangle T_n^q, q is a $+$ or $-$ sign, (u_i, v_i) is (u_1, v_1) if q is $+$ and (u_4, v_4) if q is $-$ (see Fig. 2.3). If we transform the parametric coordinates to a local system of area coordinates (ξ, ζ, γ) within in T_n^q, (3.19) can be rewritten as:

$$\vec{C}_n^q(u,v) = 2A_n^q \int_0^1 \int_0^{1-\xi} \vec{D}_{nq}(u,v,u'(\xi,\zeta),v'(\xi,\zeta))\, d\zeta d\xi \tag{3.23}$$

where $u'(\xi,\zeta)$ and $v'(\xi,\zeta)$ are given in (A.4), and the details of the coordinates transformation are in Appendix A.

Directly applying the numerical integration technique for a triangular region in [49] to (3.23) gives

$$\vec{C}_n^q(u,v) = 2A_n^q \sum_{i=1}^{N_s} w_i\, \vec{D}_{nq}(u,v,u'(\xi_i,\zeta_i),v'(\xi_i,\zeta_i)) \tag{3.24}$$

where N_s is the total number of integration points on the triangle T_n^q, and w_i is the weight corresponding to the integration point (ξ_i, ζ_i). N_s, w_i, ξ_i, and ζ_i are given in (Table 8.2, [49]).

When $T_m^p \neq T_n^q$, all the integrals in Z_{mn}^{pq} are well-behaved, so Z_{mn}^{pq} can be numerically evaluated as

$$\begin{aligned} Z_{mn}^{pq} &= p2A_m^p \sum_{j=1}^{N_t} w_j\, \vec{f}_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \bullet \\ &\quad [j\omega\mu_0 \vec{a}_n^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) + \vec{C}_n^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j))] \\ &+ \frac{p2A_m^p}{j\omega\epsilon_0} \sum_{j=1}^{N_t} w_j\, g_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \psi_n^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \end{aligned} \tag{3.25}$$

where the vector function $\vec{f}_{mp}$ and the scalar function g_{mp} are defined in (2.38), the vectors $\vec{a}_n^q$ and $\vec{C}_n^q$ are given by (2.35) and (3.24), respectively, ψ_n^q is given in (2.36), and $u(\xi_j,\zeta_j)$ and $v(\xi_j,\zeta_j))$ are given in (A.4).

For the case of $T_m^p = T_n^q$, G and $\nabla' G$ have a singularity at $\vec{r}' = \vec{r}$. To remove the singularity of G in Z_{mn}^{pqe}, the treatment described in Section 2.2.2 of Chapter 2 will be applied. The singularity of $\nabla' G$ in Z_{mn}^{pqm} will be treated by the method used in Section 2.3 of Chapter 2.

When $T_m^p = T_n^q$, numerical formulation of Z_{mn}^{pq} is given by

$$\begin{aligned}
Z_{mn}^{pq} &= pqA_m^p \sum_{j=1}^{N_t} w_j \, [\vec{f}_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \bullet \vec{f}_{nq}^{ibc}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j))] \\
&+ j\omega\mu_0 p2A_m^p \sum_{j=1}^{N_t} w_j \, \vec{f}_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \\
&\quad \bullet[\vec{a}_{n1}^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) + \vec{a}_{n2}^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j))] \\
&+ \frac{p2A_m^p}{j\omega\epsilon_0} \sum_{j=1}^{N_t} w_j \, g_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \\
&\quad \cdot[\psi_{n1}^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) + \psi_{n2}^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j))] \\
&+ p2A_m^p \sum_{j=1}^{N_t} w_j \, [\vec{f}_{mp}(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \\
&\quad \bullet\{\vec{C}_{n1}^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) + \vec{C}_{n2}^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j)) \\
&+ \vec{C}_{n3}^q(u(\xi_j,\zeta_j), v(\xi_j,\zeta_j))\}] \qquad (3.26)
\end{aligned}$$

where $\vec{f}_{mp}$ and g_{mp} are defined in (2.38), and

$$\vec{f}_{nq}^{ibc}(u,v) = \frac{\eta_0\eta_s(u,v)}{2A_n^q \mathsf{J}(u,v)}[(u-u_i)\vec{r}_u(u,v) + (v-v_i)\vec{r}_v(u,v)] \qquad (3.27)$$

Furthermore, $\vec{a}_{n1}$, $\vec{a}_{n2}^q$, ψ_{n1}^q, and ψ_{n2}^q are given in (2.44), (2.45), (2.46), and (2.47), respectively. $\vec{C}_{n1}^q$, $\vec{C}_{n2}^q$, and $\vec{C}_{n3}^q$ are defined in (3.19) with the integral domain T_{n1}^q, T_{n2}^q, and T_{n3}^q, respectively.

Chapter 4

Parallel Implementation

4.1 Introduction

Parallel processing has emerged as an enabling technology in modern computers, driven by the ever increasing demand for higher performance, lower costs, and sustained productivity in real life computer applications. Concurrency is being used in today's high performance computers with the common practice of multiprogramming, multiprocessing, or multicomputing.

Over the past five decades, electronic computers have gone through five generations of development. Fifth-generation computers are targeted to achieve teraflop performance by the end of this century. Massively parallel processing is the mainstream of the software and applications of the fifth generation. The fifth-generation MPP systems are represented by several projects at IBM (SP-2), Cray Research (T3D), Thinking Machine Corporation (CM-5), and Intel Supercomputer Systems (Paragon).

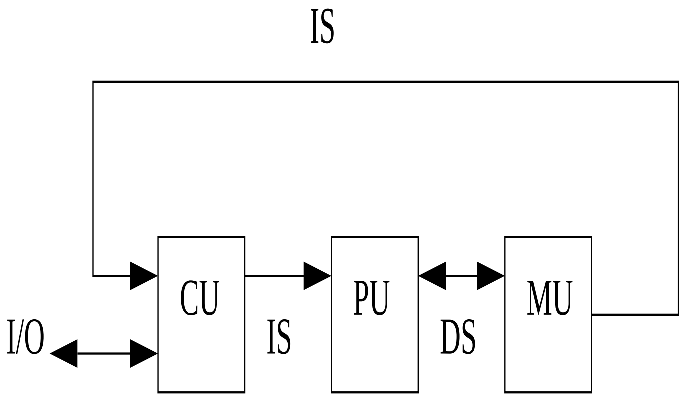
Before exploring the methods for mapping a given formulation of an electromagnetic scattering problem onto parallel computers, we take a look at different computer architecture classifications given by Flynn [64]. As shown in Figure 4.1, conventional sequential computers are classified as SISD (single instruction stream over a single data stream) computers, SIMD (single instruction over multiple data streams) computers shown in Figure 4.1, and MIMD (multiple instruction stream over multiple data streams) machines shown in Figure 4.1. In Figure 4.1, CU denotes Control Unit; PU denotes processing unit; MU is memory unit; IS is instruction stream; DS is data stream; PE represents processing element; LM denotes local memory.

The predominant type of computers until recently has been SISD or von Neumann; the CPU progresses in sequential manner from one instruction to the next and performs operations on data items one at a time. Many widely used computer languages, such as FORTRAN, were designed for SISD computers. The advantage of the SISD approach is its simplicity and familiarity to many programmers.

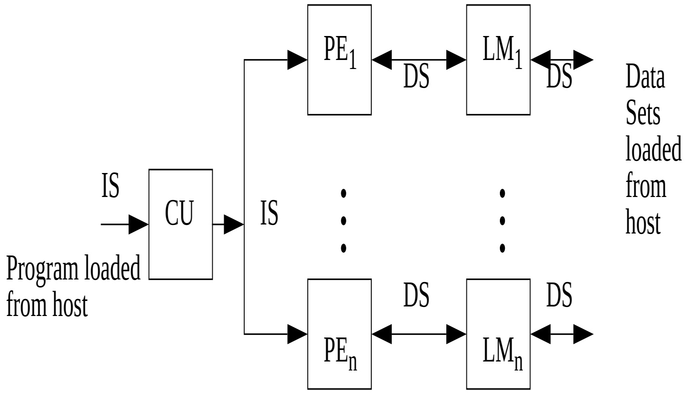
The first important dichotomy in parallel systems is how the processors are controlled. In SIMD systems, all processors are under the control of a master processor, called the controller, and the individual processors all do the same instruction (or nothing) at a given time. Thus, there is a single instruction stream operating on multiple data streams, one for each processor. The Illiac IV, the first large parallel system (which was completed in the early 1970s), was a SIMD machine. Thinking Machine Corporation's connection machine CM-2 is a SIMD machine with 64k simple 1-bit processors. Vector computers may also be conceptually included in the class of SIMD machines by considering the elements of a vector as being processed individually under the control of a vector hardware instruction.

SIMD machines execute efficiently on the types of problems for which they were designed, but there are limitations to their applications. Disappointed numerical analysts discovered soon after the first SIMD machines were programmed that the projected speed improvements were not obtained. The cause was found to be program sections that are not vector operations. In the limit that vector operations take zero time, the maximum throughput is determined by the execution of the non-vector (scalar) instructions in SISD mode (see [65]).

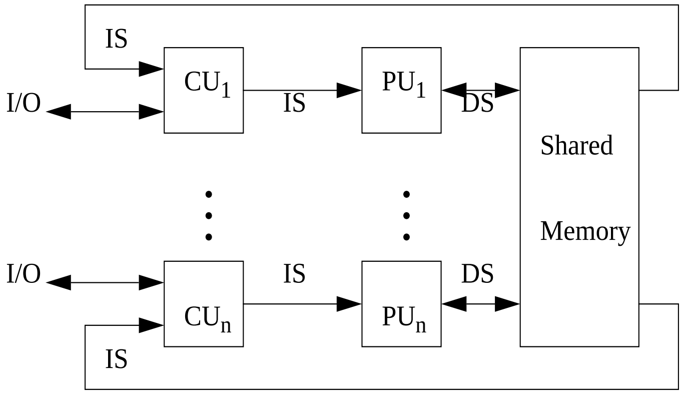
Most parallel computers built since the Illiac IV have been MIMD systems. Here, the individual processors run under the control of their own program, which allows great flexibility in the tasks the processors can do at any given time. Theoretically,



(a). SISD Architecture



(b). SIMD architecture (with distributed memory)



(c). MIMD architecture (with shared memory)

Figure 4.1: Flynn's classification of computer architectures

this flexibility allows the application programmer to run a greater fraction of the problem in parallel than with vector processing alone. In practice, programming MIMD machines is difficult because of the inherent complexity of multiple processors doing different things simultaneously, as well as the general lack of automatic parallel decomposition software tools. It also introduces the problem of synchronization. In an SIMD system, synchronization of the individual processors is carried out by the controller, but in an MIMD system other mechanisms must be used to ensure that the processors are doing their tasks in the correct order with the correct data.

For many problems, the programs in the individual processors for an MIMD system may be identical (or nearly so). Thus, all programs are carrying out the same operations on different sets of data, just as SIMD machines would do. This gives rise to the Single-Program Multiple-Data (SPMD) model of computation.

Another important dichotomy in parallel computers is shared versus distributed memory. An example of shared memory system with four processors is illustrated in Figure 4.2. Here, all the processors have access to a common memory. Each processor can also have its own local memory for program code and intermediate results. Interprocess communication consists simply of writing to and reading from the common data area, a fast operation because it occurs at random memory access rates. Shared memory architectures also lend themselves to conventional time-sharing applications with multiple users running multiple jobs. Since each user's program and data reside in a common memory area, jobs can be assigned to CPUs dynamically for the purpose of load balancing.

A serious disadvantage is that different processors may wish to use the common memory simultaneously, in which case there will be a delay until the memory is free. This delay, called contention time, can increase as the number of processors increase. Typically, shared memory has been used for systems with a small number of processors such as Cray machines with up to 16 processors.

An alternative to shared memory systems is distributed memory systems, in which each processor can address only its own local memory. Communication between processors takes place by message passing, in which data or other information is

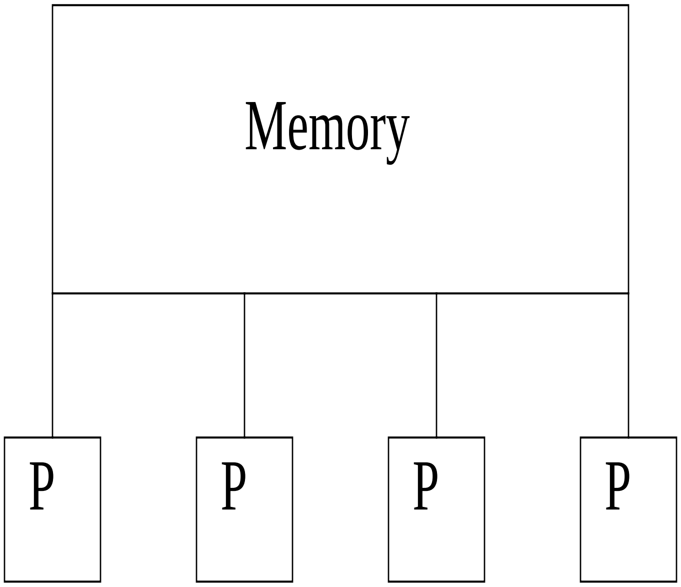


Figure 4.2: An example of a shared memory system

transferred between processors. The most practical massively parallel computers are distributed memory MIMD systems. The connection machine CM-5, the Intel Paragon, and the IBM SP-1 are examples.

4.1.1 Distributed Memory System

An important and interesting aspect of parallel computer systems is how the individual processors communicate with one another. This is particularly important for distributed memory systems but it is also important for shared memory systems since the connection to the shared memory can be implemented by several different communication schemes. We next discuss briefly a number of the more common schemes.

A distributed-memory machine consists of a set of processors linked by interconnection networks. Each processor has its own memory that is directly accessible only by this processor. Data exchange and global operations among processors are accomplished through message passing. Each processor is connected to a fixed number of processors in some regular geometry (Figure 4.3) such as ring, 2-D mesh (e.g., Intel

Touchstone Delta and Intel Paragon), fat tree (e.g., TMC CM-5), and hypercube (e.g., Intel iPSC/860 and nCUBE 2).

Writing an efficient program on distributed-memory machines is more difficult than programming on sequential machines or shared-memory machines. Some issues that arise in programming distributed-memory machines that must be carefully addressed are:

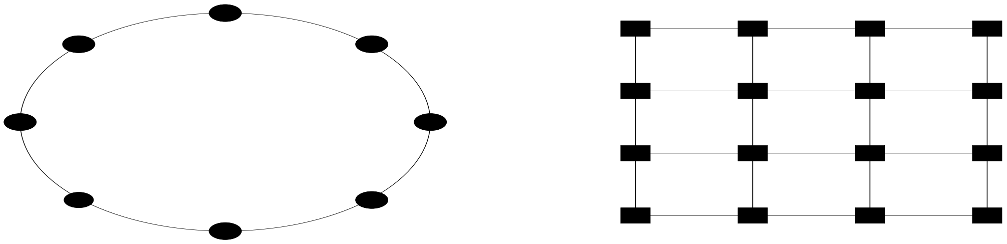
- *Parallelism*—This involves decomposing a large computation into a set of tasks that are assigned to processors and executed in parallel.
- *Data Distribution*—Since there is no globally shared memory on a distributed-memory machine, data structures in an application must be partitioned and distributed among processors.
- *Data Exchange and Global Operations*—Parallel tasks that coordinate a joint computation often need to exchange data or synchronize operations with each other. To do this, explicit message passing must be inserted into the user program.

Load balancing and reduction of communication cost are always two important factors for achieving good performance on distributed-memory machines. They are discussed in the implementation section.

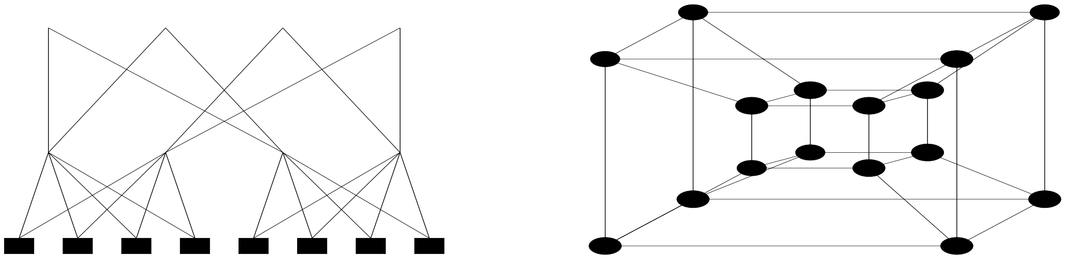
In Appendix C, we describe important features of three distributed-memory architectures that were used in this thesis. We will focus on the architecture of the Connection Machine CM-5 in detail and also briefly on the Intel and the IBM SP-1.

4.1.2 Parallelism and Performance Issues

Conventional algorithms that have been optimized for a single processor computer, or a vector-pipelined computer, are often taken for granted since they are readily available as subroutine packages. However, these algorithms may perform poorly in the parallel environment of a particular parallel computer. To achieve superior



(a) Ring (b) 2-D Mesh



(c) Fat Tree (d) 4-D Hypercube

Figure 4.3: Distributed-memory architectures.

performance, new algorithms must be developed that conform better to the parallel architecture.

To illustrate parallelism and load balancing, we consider the problem of adding two n-vectors $\vec{a}$ and $\vec{b}$. The additions

$$a_i + b_i, \qquad\qquad i = 1, 2, \cdots, n, \tag{4.1}$$

are all independent and can be done in parallel. Thus, it has perfect mathematical parallelism. On the other hand, it may not have perfect parallelism on a parallel computer because it may not have perfect load balancing. By *load balancing* we mean the assignment of tasks to the processors of the system so as to keep each processor doing useful work as much as possible. For example, there are $p = 32$ processors and $n = 98$ in (4.1). Then, the processors can work in perfect parallelism on 96 additions but only two processors will be busy during the remaining 2 additions. Thus, there is not perfect load balancing to match the perfect mathematical parallelism.

In general, load balancing may be done either statically or dynamically. In *static load balancing*, tasks (and, perhaps, data for distributed memory system) are assigned to processors at the beginning of a computation. In *dynamic load balancing*, tasks (and data) are assigned to processors as the computation proceeds. A useful concept for dynamic load balancing is that of a pool of tasks, from which a processor obtains its next task when is is ready to do so.

Related to load balancing is the idea of *granularity. Large-scale granularity* means large tasks that can be performed independently in parallel. *Small-scale granularity* means small tasks that can be performed in parallel.

In general, in a distributed memory system, exchange of data between processors will be necessary at various times during an overall computation, and to the extent that processors are not doing useful computation during the communication, this constitutes an overhead.

Synchronization is necessary when certain parts of a computation must be completed before the overall computation can proceed. There are two aspects of synchronization that contribute to overhead. The first is the time to do the synchronization; usually this requires that all processors perform certain checks. The second aspect is that some, or even almost all, processors may become idle, waiting for clearance to proceed with the computation.

The degree to which an algorithm can exploit a multiprocessor is often measured by either speed-up or efficiency. Ideally, we could solve a problem p times as fast on p processors as on a single processor. This ideal is rarely achieved; what is achieved is called the *speed-up* defined by:

$$S_p = \frac{\text{execution time for a single processor}}{\text{execution time using p processors}} \tag{4.2}$$

The efficiency can be defined as

$$E_p = \frac{S_p}{p} \tag{4.3}$$

Since $S_p \leq p$, we have $E_p \leq 1$ and an efficiency of $E_p = 1$ corresponds to a perfect speedup of $S_p = p$.

The speed-up defined in (4.2) is a measure of how a given algorithm compares with itself on 1 and p processors. However, the parallel algorithm may not be the best algorithm on a single processors. Hence, a better measure of what is gained by parallel computation is given by the alternative definition

$$S'_p = \frac{\text{execution time on a single processor of fastest serial algorithm}}{\text{execution time of the parallel algorithm on p processors}} \tag{4.4}$$

Both of the measurements S_p and S'_p are useful and the context of discussion will determine which is to be used.

4.2 Structure of the Code

A sequential computer program was written based on the formulation developed in Chapter 3. This code was merged into the ParaMoM code previously developed by Cha's group at SRC based on the formulation derived in Chapter 2. In this section, we discuss the ParaMoM features and its program structure.

There are two major components in the ParaMoM package: the target model processing software and the MoM code itself. The target model processing software generates the target model using CAD and a model processor for input into the MoM code. The package is capable of accepting data files that consist of a limited class of Initial Graphics Exchange Standard (IGES) data types, as well as files from SCAMP which is a computer-aided design (CAD) package. The model processor allows the user to perform the following tasks:

- specify wires as either radiating antennas or scatterers
- place distributed load impedances on both surfaces and wires
- place lumped loads at antenna feed nodes
- find the intersection of wires and surfaces
- view a simple, coarse grid version of the model
- triangulate all curved surfaces (and subdivide all wires) according to a specified maximum edge length

- specify planes of symmetry

The output of the model processor (DC file) is in a format that can be read by the MoM program. It can also be read back in to the model processor to be regridded, or have its wire or material impedance characteristics modified.

Some of the capabilities of the MoM code in this package are summarized as follows:

- For curved surfaces with a parametric description, a new basis function has been used.
- Three operator equations have been implemented; electric, magnetic, and combined field integral equations.
- In addition to the default of perfect electric conductors (PECs), objects can be described using a surface impedance value in units of ohms per square. This is intended for either a material sheet (for example an R-card) of finite conductivity or a dielectric material coating on a PEC surface and is handled within the E-field formulation.
- A target with mirror symmetries can be treated by only modeling a portion of the target and specifying up to three planes of symmetry.
- Wires can be included in the model. They may be separate or connected to surfaces. Where they are connected to surfaces, appropriate junction basis functions will be used (see Reference [66]).
- The user may compute either radiation or scattering due to wire antennas.
- The desired far-field pattern can be computed for a variety of monostatic and bistatic configurations.
- The far-field patterns can be computed for up to 16 different combinations of transmit/receive polarizations (four receiver polarizations for antenna gain) in a single run.

The ParaMoM's structure is shown in Figure 4.4. A brief description of each box in Figure 4.4 is given below:

Setup: The role of the ParaMoM setup phase is to read in necessary information from two files. One is the parameters' file and the other is the target geometry description (DC) file. The information given in the parameters' file is listed as

1. The integration formulation type (EFIE, or MFIE, or CFIE).
2. The excitation frequency in megahertz.
3. The transmitter and receiver polarization combinations (up to 16 of them).
4. Three angle sweep scenarios to be chosen (see Appendix B): monostatic or fixed bistatic angle, bistatic with fixed transmitter, or bistatic with fixed equivalent line of sight.

The target geometry description (DC) file is generated by a preprocessor, which triangulates all curved surfaces according to a specified maximum edge length. The DC file contains the following information:

1. The total number of triangular faces, edges, and nodes in the model.
2. The connectivity of these triangles.
3. The parametric coordinates of vertices of these triangles
4. The material specification

Precomputation: In this stage, several arrays, which are required by the fill algorithm, are computed. The position vector arrays are computed at each integration point for both expansion and testing integrals on each triangular patch. The expansion and testing functions and their divergences are calculated at each integration point on each patch.

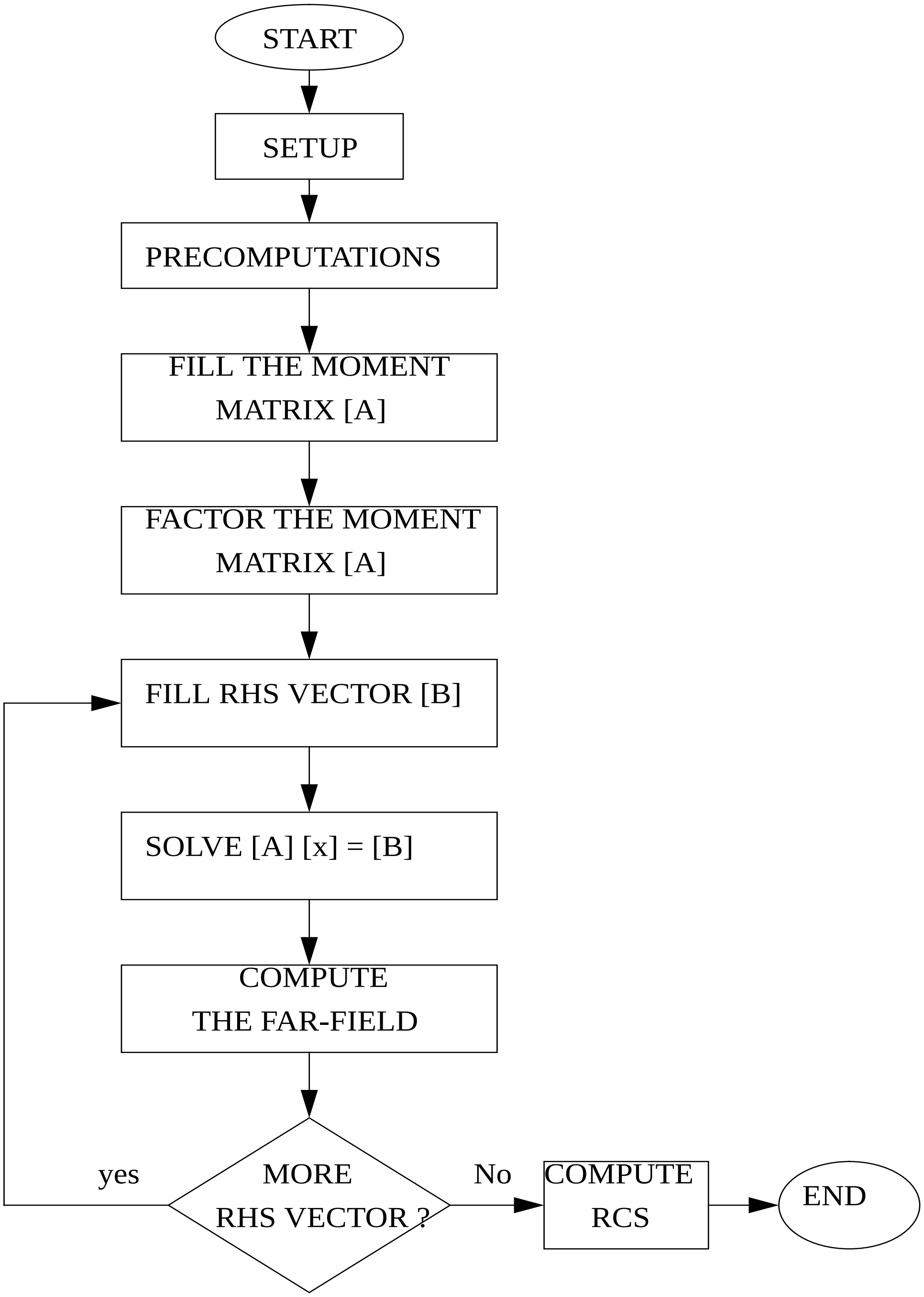


Figure 4.4: The sequential code structure

Fill Moment Matrix: Most sequential MoM codes fill the matrix by looping through surface patches. The reason for this is that each source or testing basis function domain generally extends over more than one patch; each basis function overlaps one or more neighboring basis functions. A source-patch is only half the domain of each of three basis functions (likewise for field patches); a patch-patch interaction produces contributions to nine different matrix locations. A matrix element is not completely computed until the two patches that make up the basis function have interacted with the two patches that make up the testing function domain. Thus, each matrix location is written to four times. Each time the new contribution is added to the current value.

The moment matrix can be either one of EFIE, MFIE, and CFIE depending on the specification given in the input file.

RHS Vector Fill: Computing the excitation (right-hand side vector) for either EFIE, or MFIE, or both. This is due to an incident plane wave.

Factorization and Solve: Linpack library LU function calls for the single precision complex data used to factorize the moment matrix into lower and upper triangular matrices, then back substitution is applied to solve the linear system.

Far-Field and RCS Computation: After the solution of the system, the induced current is used for computing the far field for each combination of the transmitter and receiver polarizations. For the case of multiple excitation vectors, summation of all the contributions from all induced currents is required to compute the far-field for evaluating the radar cross section.

4.2.1 The Parallel Programming Model and Approaches

The Parallel ParaMoM Code is implemented on three parallel platforms (CM-5 , Intel, IBM SP-1). Each of the platforms is a coarse-grained MIMD (multiple instruction multiple data) with relatively powerful nodes. Each machine supports an explicit message-passing programming model. In each case, the message-passing library may be accessed from a Fortran program.

The explicit message-passing approach has a number of advantages for the MoM application. Performance of a message-passing program is less dependent on the interconnection topology of the parallel machine than it is in data-parallel and shared-memory approaches. There is more direct control over the message volume and timing. Therefore, message-passing programs are more likely to port efficiently from one parallel architecture to the next. In addition, the functions of the message-passing library are quite similar on all architectures; differences are mainly in syntax rather than in philosophy. In contrast, various data-parallel and SIMD machines rely on compiler technologies to varying extents to perform the parallelization work, and the parallel facilities may exist at different levels of abstraction. For the data-parallel processing, an array object refers to all the data elements of the array simultaneously from the software perspective; the separate operations on the array's elements are all performed simultaneously from the hardware perspective.

There are a number of methods available to perform the solution of dense linear system of equations. In specialized problems, iterative approaches can be quite efficient in terms of memory and computation time. However, iterative techniques suffer when used to treat many right-hand side vectors. In this study, it is of interest to solve systems with a large number of right-hand side vectors (i.e., scatterers illuminated by multiple incident waves) and iterative methods will not be considered. The Gaussian elimination method will be employed under these conditions, since the computational intensive factorization need only be performed once.

The computational expenses of the setup phase, precomputation phase, rhs vector fill phase, and scattered-field computation phase are of order N, where N is the number of unknowns (proportional to surface area of target). The matrix fill phase is of order N^2, the matrix factorization is of order N^3, and the matrix solution is of order N^2. Because our primary interest in applying parallel computers lies in reducing the total time required to solve large problems (i.e., large N), we concentrate our energy on reducing the N^2 and N^3 processes. In this effort we are developing parallel algorithms for the matrix fill, factorization, and solution processes, but also for the setup, precomputation, rhs vector (or vectors) fill, and scattered-field computation phases as well, since the whole program must be run on the target system.

In addition to computational cost concerns, it is necessary to consider memory requirements. On the class of computers we are considering, the main memory (RAM) is distributed among the nodes. The moment matrix is generally much too large to fit in the memory of one node, so it must be distributed equitably among the nodes. The matrix size is of order N^2, but all other arrays used in the ParaMoM code are of order N or lower.

To parallelize the ParaMoM code, we first need to discuss what kind of algorithm design strategy to use. In designing an parallel algorithm for MIMD computers, several approaches are possible. Here, we give a few commonly used approaches. For one of them, each processor may execute exactly the same program, but with different input data. No communication is ever required. This is the trivial parallelization strategy, which is the easiest for programming. The drawback is that this approach requires the entire computation fits into a single processor's memory, which is often found to be impossible. The pipeline approach is to break the computation into a series of small tasks which have to finished sequentially for a large set of data. Each task is assigned to one or a few processors. After completing its computation, a processor passes its result on to processors which handle the next task. However, this approach is applicable to a restricted class of problems.

The problem partition approach has received the most attention in scientific computing applications. In this approach, each processor executes substantially the same program, but on a portion of the problem data. Processors are loosely coupled throughout the computation, exchanging information whenever necessary. It is very suitable for solving large problems, where all available memory is required. The implementation difficulties are how to partition the problem to have a good load balance.

The parallel algorithm implemented on the Intel Paragon and IBM SP-1 is an example of the partitioning approach. On the CM-5, the algorithm used is one that combines both the message passing paradigm and the data parallel paradigm. The matrix fill and field computation use the partitioning strategy implemented in the message passing paradigm. The dense linear system solver is the CM Fortran interface with the CMSSL library [67] which uses Gaussian Elimination and back substitution. The connection between these two paradigms is accomplished by a high speed massive storage device.

The implementation on Intel utilizes the NX message-passing library [68], the standard PVM (Parallel Virtual Machine) message-passing library [69] and ScaLAPACK [70] for excellent portability. This PVM implemented on Intel machines has been ported to the IBM SP-1 with little effort. ScaLAPACK, a publicly available software, was developed at Oak Ridge National Laboratory.

ScaLAPACK is a distributed-memory version of the standard LAPACK linear algebra library. It is built on the BLAS library, which is at the heart of LAPACK[1] and its predecessor LINPACK, in combination with a linear algebra communication library (BLACS [71]). The current portable version of the BLACS utilizes calls to the standard PVM (Parallel Virtual Machine) message-passing library and, therefore, can be easily ported to any machine on which PVM runs (for example Intel or Cray T3D). In addition, BLACS has also been implemented using the Intel NX message-passing library for optimum performance. ScaLAPACK performance [72] is scalable over a wide range of problem sizes and machine sizes.

In the Intel and IBM implementations, the parallel code structure can be the same as the sequential one in Figure 4.4, whose components are implemented in parallel instead of sequentially.

ScaLAPACK is currently not an efficient matrix solution option on the CM-5 because a BLAS library that is optimized for the CM-5 vector units does not exist. Therefore, the platform specific CMSSL matrix equation solver is being used.

The CMSSL library [67] uses a data-parallel programming model and cannot be

---

[1] LAPACK is a public domain, transportable linear algebra library in Fortran 77 designed to replace EISPACK and LINPACK libraries.

globally interfaced with the message-passing, matrix-filling algorithm. Therefore, in our implementation, the matrix fill and matrix factorization and solution (factor/solve) are two distinct program units. A high-speed device, such as the scalable data array (SDA) or DataVault (see [73]), is used to link these two stages. The message-passing MoM matrix-filling program fills the matrix and writes it to a file in the format required for the factor/solve stage. The matrix is subsequently read in by the data-parallel matrix solver stage.

There is very little performance penalty for using separate program units for the filling and factor/solve because the DataVault or the SDA has very high data rates. As an illustration, we have run some exercises to demonstrate our claim. There are time data from both CMMD[2] timer [74] and the CM Fortran timer [75, 75] listed in Tables 4.1 and 4.2. In Table 4.1, the elapsed time for writing the moment matrix to a SDA file is measured by the CMMD timer. The writing operation is executed by the CMMD global write under CMMD synchronous sequential mode [74, 77]. The CM Fortran Utility library [75] provides a "SO" mode which is compatible with almost all CM systems. One can see that the extra effort in using a high performance storage device to utilize both the message-passing paradigm and the data-parallel paradigm is justified.

For this moderate-sized problem the I/O time is small compared to the factor time. Since the I/O time scales as N^2 and the LU time scales as N^3, it is evident that for very large problems the I/O time will not be a limiting factor.

The implementation of the matrix-filling algorithm on the CM-5 is written in Fortran 77 with calls to the CMMD message-passing library. The resulting code is adequate to investigate the efficiency of the parallel algorithm as measured by the speed-up (ratio of elapsed time for one node versus elapsed time for p nodes). However, the per-node performance is expected to be poor because Fortran 77 code cannot utilize the CM-5 vector units efficiently [78, 79]. However, the data-parallel code for matrix factor and matrix solve utilizes the vector units. The LU and Solve operations

---

[2] CMMD is a trademark of Thinking Machines Corporation. It is a library of message-passing routines for the CM-5 system.

Table 4.1: The elapsed time of writing the moment matrix to a SDA file using CMMD global write under CMMD synchronous sequential mode (recorded using the CMMD timer).

$N^\dagger$	32-node		512-node	
	write	fill	write	fill
3000	1.19(s)	1207(s)	0.39(s)	94.1(s)
5000	2.93(s)	3295.2(s)	0.78(s)	237.9(s)
10000			1.47(s)	901(s)

Table 4.2: The elapsed time of reading the moment matrix from a SDA file using CM Fortran (SO mode) function (recoded by CM Fortran timer).

N	32		512	
	read	factor	read	factor
3000	2.01(s)	79.6(s)	0.67(s)	19.9(s)
5000	5.24(s)	170.7(s)	2.08(s)	41.0(s)
10000			2.25(s)	156.4(s)

require the most floating point operations. The performance of the CM-5 implementation is expected to have a good overall speed-up.

Memory Requirements

The matrix size is N^2 where N is the number of basis functions (or edges between triangular facets or "faces"; if the scatter has an open surface then N is slightly less than the number of edges since boundary edges do not have basis functions, but this effect is small). Each face has three edges and each edge is shared by two faces, so the relationship between the number of faces N_f and N is $N \approx \frac{3}{2}N_f$.

For the parallel approach we use to develop a parallel algorithm, each node runs the same program. The matrix is distributed among the p nodes so that the memory required by each node is $8N^2/p$ bytes assuming single precision complex matrix storage. The sequential paramom requires memory M

$$M = M_s + M_z \approx 2500N_f + 18N_f^2 \tag{4.5}$$

where M_s denotes the portion of memory required by arrays other than the matrix, and M_z is the memory required by the matrix. In the parallel approach, the local memory of each node must be large enough to exceed M_s and a portion of M_z.

The most important task before implementing the ParaMoM as a parallel code is to reconstruct the program in such a way to reduce M_s. This is very difficult to accomplish. The new version of the ParaMoM code has a lower M_s, which is given by

$$M_s \approx 1000N_f \tag{4.6}$$

This significant reduction of M_s makes it possible for a MIMD computer with a typical configuration of 16 Mbytes to 32 Mbytes local memory to be used to compute a large application problem. For a very large problem which has a large M_z and moderate M_s, an out-of-core algorithm may be needed to save on memory. In the next section, we discuss an out-of-core fill algorithm for the CM-5 implementation.

4.3 Implementation

In this section, the parallel implementation of the ParaMoM is presented. We discuss the CM-5 implementation in detail and give brief information about the Intel and IBM implementations as well. This section is organized following the order of the diagram in Figure 4.4. In Section 4.3.1, the details of the parallel implementation of the setup code are given and the CMMD *global synchronous broadcast* I/O mode is introduced to perform parallel I/O for setup. In Section 4.3.2, a parallel precomputation algorithm is presented and a pseudo code of the algorithm is given. In Section 4.3.3, a data-parallel algorithm for filling the moment matrix is given. The matrix filling is implemented with Fortran 77 and CMMD message passing library on the CM-5 system, Fortran 77 and NX message-passing library or PVM on the Intel, and Fortran 77 and PVM on the IBM SP-1. In Section 4.3.4, we discuss a very flexible algorithm to compute the excitation vectors either sequentially or in parallel, depending on the problem. In Section 4.3.5, a data-parallel implementation of the Gaussian elimination linear system solver is presented. In Section 4.3.6, the parallel implementation of the RCS code is discussed and a pseudo code of the parallel algorithm is given. Finally, in Section 4.3.7, the parallel out-of-core fill algorithm is presented.

4.3.1 Parallel Setup

During setup, there are two input files to be read. One is a parameters file which is described in the previous section. Another is the target DC file. They are described in some detail in the previous section. For a large problem, the size of the DC file varies from several Mbytes to several tens of Mbytes. One may use the sequential setup for each node, where each node opens the DC file and reads the data independently. The argument for that is that the setup complexity is in the order of N where N is the maximum number of unknowns of the system. Since UNIX sets a limit on the number of files that can be open at any one time, for a coarse grain computer with a UNIX type of operating system this may easily exceed the limited number of files that can be opened at the same time. Also only one node can read data at a time so the rest of the nodes remain idle. This could be very time consuming. A solution is to use parallel I/O to achieve high performance. On the CM-5, CMMD provides a parallel I/O mode called *global synchronous broadcast*, in which a global file is accessed by all nodes simultaneously, with each node reading (or writing) the same data. This mode provides a significant increase in flexibility and parallelism for I/O operations.

For Intel and IBM machines, there is a global broadcasting mode in which one node opens the file and reads the data sequentially then broadcasts the data to the rest of nodes.

4.3.2 Precomputation

In the precomputation stage, there are six arrays to be computed. They are position vectors, basis functions, and the divergence of the basis functions at each integration point on the source patches and field patches, respectively. These arrays are three or four dimensional arrays. One of the dimensions of each of these arrays has the size of the maximum number of patches. For the problem we are interested in, the maximum number of the patches could be in the range from one thousand to several thousand. Each array is required by the filling algorithm in each node. We can simply let each node compute these arrays independently. However, it is too computationally expensive and inefficient, so instead these arrays will be computed in parallel. Let $r_s(k,i,j)$ and $r_f(k,i,j)$ denote the position arrays of the source patch and field patch, respectively. Also $b_s(k,n,i,j)$ and $b_f(k,n,i,j)$ respectively denote the basis function arrays on the source patches and the field patches, and $db_s(n,i,j)$ and $db_f(n,i,j)$ denote the divergences of the basis function on the source patches and the field patches, respectively where $k = 3$, $i = 4$ or 7, $n = 3$, and $j = N_f$, the number of the patches in the model.

To compute these arrays, we first divide the computational work almost evenly for each node. Each node works on its portion, and when it has completed its work the node broadcasts the result to the rest of nodes. The partition of the arrays is done along the largest dimension. Let $d = \frac{N_f}{p}$, and $q = mod(N_f, p)$, then these nodes whose index values are less than q compute $(d+1)$ components of the arrays, and the rest of the nodes compute d components. That is, the node 0 computes $j = 0 : d$, the node 1 computes $j = d+1 : 2d+2$, node $q-1$ computes $j = (q-1)*(d+1) : q*(d+1)$, node q computes $j = q*(d+1)+1 : q*(d+1)+d$, and node p computes $j = N_f - d+1 : N_f$.

The details of the algorithm are shown in Figure 4.5.

The CMMD library provides a global broadcast function. In a broadcast, a message is sent from a single source to all nodes. All nodes must take part in the broadcast. One node must signal its intention to send the broadcast message; all other nodes must signal their readiness to receive the message. From the hardware aspect, once a broadcast signal is up, the system begins checking the responses. When all the receiving nodes have received the broadcast message, the system considers the broadcast completed and returns the broadcast call. Keep in mind that all nodes receive data simultaneously and all receive the same amount of data in every broadcast. Therefore, every node must have sufficient buffer space to hold the broadcast message.

4.3.3 Moment Matrix Fill

In the matrix filling process, there are two separate decomposition or partitioning concerns. An important point is that these decompositions need not be the same. Our general approach may be summarized as follows:

- The data decomposition is driven by the requirements of the matrix solver software.

- The basic matrix filling algorithm should be relatively independent of the specific data decomposition and the work decomposition schemes used. This increases flexibility and portability in terms of working with a variety of solvers and architectures.

- The work decomposition should be done to optimize the tradeoff between computational and message-passing costs.

Precomputation

```
1. Compute the lower bound and upper bound for each node;
2. Loop over the patches from lower bound to upper bound;
      Loop over all the integration points on one patch;
         Compute all three components of the position vector arrays, $r_s$ and $r_f$;
           Loop over the edges on the patch;
              Compute the divergence of the basis functions, $db_s$ and $db_f$;
              Compute all three components of the basis function arrays, $b_s$ and $b_f$.
3. Find out the size of the message and the start address of the message to be
   broadcasted for each node;
4. Find out the location of the received message in each node from the rest of the
   nodes.
5. Each node broadcasts its data to the system;
```

Figure 4.5: The pseudo code for the precomputation algorithm

The MoM matrix filling is very well suited to MIMD parallel processing. Each matrix element is completely independent of every other element. Therefore, the simplest parallel algorithm is to simply partition the work the same way the data is partitioned (which is driven by the needs of the matrix solver as mentioned above) and have every node compute a piece of the matrix independently. However, this is not necessarily the most efficient algorithm, as described below.

The most obvious way to fill the matrix in a sequential program is for the outermost loops to be indexed by the columns and rows of the matrix. Each column of the matrix corresponds to an expansion function ("source basis function") and each row corresponds to a testing function ("field basis function"). In practice, however, most sequential MoM codes fill the matrix by looping through surface patches. The reason for this is that each source or testing basis function domain generally extends over more than one patch; each basis function overlaps one or more neighboring basis functions.

In a typical triangular patch formulation such as that described in Chapters 2 and 3, each basis function is defined on the edge between two adjacent patches. Each patch is part of the domain of three different basis functions. A source-field pair of patches is involved in the computation of nine different source-field basis function interactions. Each such source-field basis function interaction corresponds to a single matrix element. Much of the information required to compute the interaction between the source and field basis functions is related only to the geometry (e.g., the distance between two points in the basis function domain). This information is the same regardless of which basis function interaction is currently considered. Because there are nine basis function interactions for a given pair of patches, the geometry information may be computed once and used nine times if the loops are indexed by patches rather than by basis functions.

A source patch is only half the domain of each of three basis functions (likewise for field patches); a patch-patch interaction produces contributions to nine different matrix elements. A matrix element is not completely computed until the two patches that make up the basis function have interacted with the two patches that make up the testing function domain. Thus, each matrix location is written to four times. Each time the new contribution is added to the current value.

The exact savings due to patch-by-patch looping depends on the specific formulation and computer system being used. However, a typical timing shows that 25% to 40% of the total fill time is spent computing the patch-patch geometry interactions. If each such interaction were to be calculated nine times, as is necessary when looping is done over columns and rows, the total computational cost for matrix filling would be 300% to 420% of the original.

By using a sequential computer, there is no question that the patch looping algorithm is more efficient than looping over basis functions. On the parallel computer, if the work is partitioned by patches the patch interactions can again be computed once and used nine times. However, each resulting matrix contribution must be distributed to the node on which it resides, which in general is NOT the local node. Therefore, the costs associated with communication between nodes could possibly overcome the advantage of reusing the patch interactions.

A valid point is that if the work is partitioned according to matrix position, as previously mentioned, no internode communication is required during the matrix filling loops. Therefore, although the computation time per node may be increased 300% to 420% of the original time as previously described, the TOTAL performance should increase directly proportionally to the number of nodes p, giving speed-up in the range of p/3 to p/4.2 relative to the single-node patch-looping algorithm. These are very respectable speedup numbers, and should be very easy to achieve. We consider this partitioning to be an acceptable fallback position if performance with the more complicated patch partitioning algorithm described in the next subsection is not as good as expected.

Patch-Partioned Parallel Algorithm

As described in the previous paragraph, patch partition has certain advantages over edge partition. In this section, a patch partitioned parallel algorithm will be presented. The question is how to design an efficient patch partition algorithm. A simple thought is to select one node to compute the patch pair interaction for a given source-field patch pair, and then send the result as message to the nodes that need the result. There are up to eight nodes which receive this message. That means that eight possible messages are required to replace eight possible recomputations. Communication is very expensive in a distributed memory system. Although some techniques like asynchronous message passing, collective message passing, and active message passing will reduce communication overhead, intensive communication is what we try to avoid. We can conclude that it is not an attractive idea. If the partition is done in such a way that the computation involves a field patch that can be performed by one node, then the possible number of computations of interaction is reduced to three. From a data decomposition point view, this is nothing but column decomposition. We have the choice of scattering and slab decomposition; a slab is a matrix block consisting of a number of adjacent columns of the matrix. The decision is based on which decomposition is more efficient. Relating to this question, we have to look at the global numbering scheme used to number the edges of the triangular patches in the geometry model. Each triangle has at least two of its edges with an adjacent global number. If the triangle is a source patch then the global number of its edges is related to the column number in the moment matrix. On the other hand, the global edge number of a field patch is the row number of the moment matrix. The locality of the edge number within a patch gives an indication to use the block decomposition instead of the scattering one. The column block (or slab) decomposition can be described as follows.

Let $q = mod(N, p)$ and $d = \frac{N}{p}$, then the partition is as presented in Figure 4.6. Except for the boundary column in each node, the possible number of computations for the interaction is two. In other words, there is only one possible redundant computation for patch interaction. If N/p is a big number or there is a large granularity problem, the probability is high for there being no redundant computation. Since there are no communication requirements for this filling implementation, the term

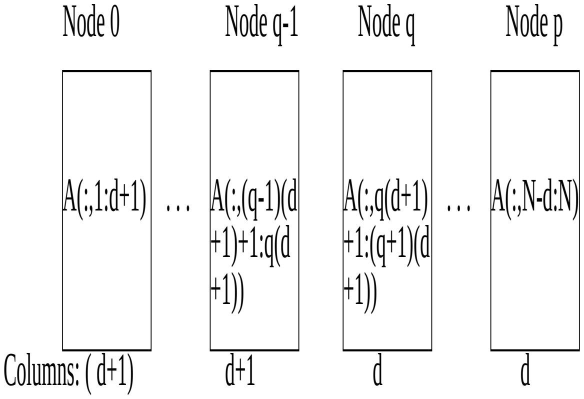


Figure 4.6: Data decomposition for matrix fill implementation

'embarrassingly parallel' is used to discribe such an algorithm in the parallel computing community.

The details of the filling algorithm are described in the rest of this subsection. In our algorithm, every node loops over all the source patches sequentially (actually, this could be done in any order). On one source patch, there are up to three possible edges. Each node finds out the column number which is associated with the edge, the index of the node that should compute this column, and the local index of the column in the slab within the node. A job counter is assigned to each node. One node's job counter will be increased by one if it is found to be the one to compute the current column. After each node goes through all three possible edges of that source patch, only the nodes which are selected to do the computation fill the moment matrix elements. The rest of the nodes go ahead to pick another source patch, repeating the selecting process. One can see the selecting process by the nodes is the only overhead compared with the sequential algorithm. So, one can expect the parallel algorithm to have a good speed-up. Since there are usually only two nodes selected for one source patch and both of them will compute the patch-pair interaction, this is only one extra computation. This fill algorithm allows for nearly total load balance because any processor will have at most one extra row compared to all other processors. Figure 4.7 shows a pseudo code of matrix fill algorithm which described.

Parallel Moment Matrix Fill Algorithm

```
1. Loop over source patches from one to $N_f$;
2. Loop over three possible edges on the source patch;
    (a) Find out the global index of the edge;
    (b) Compute the index of the node in whose slab includes the column whose number is the same as the global edge index;
    (c) Compute the local index for the location of the column in the node;
    (d) The selected node increases its job counter by one and stores the information of the edge (the global edge index, the local edge index, and the local position in the node);
3. Those nodes whose job counter is 0 go back to 1;
4. Loop over field patches from one to $N_f$;
    (a) Compute all the integration points for the source-field patch pair interaction (Green's function);
    (b) Loop over source basis functions for that node, from 1 to its job counter;
        • Loop over all possible field basis functions (or edges on the field patch)
        • Use the formula derived in Chapter 2 to compute the elements of the moment matrix;
        • Sum all the contributions from the four pairs of patches associated with each source-field basis function;
End of the source patch loop.
```

Figure 4.7: The pseudo code of the parallel fill algorithm

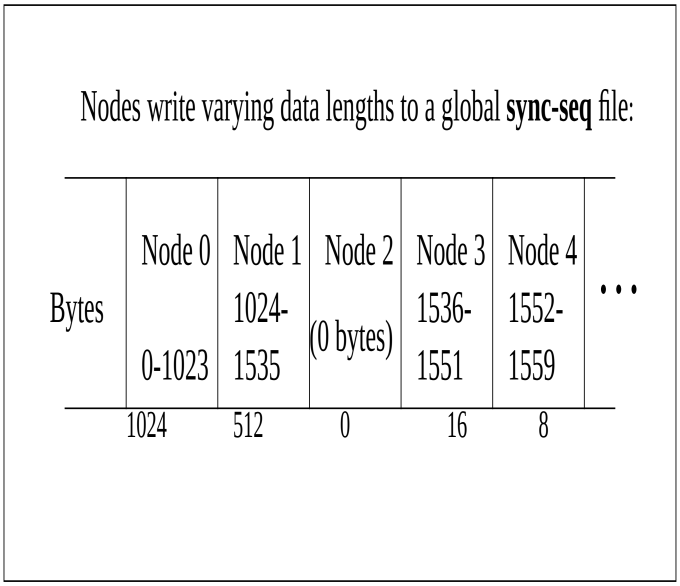


Figure 4.8: Varying amounts of data can be written by nodes in *CMMD synchronous sequential* mode

To write the moment matrix to a file, we use the CMMD global I/O mode called *global synchronous sequential* mode, in which a global file is accessed by all nodes simultaneously, with each node reading (or writing) its own portion of the file. The data written to the file comes from each node in sequence, from node 0 up to the highest numbered node. Each node contributes a buffer of data (of arbitrary size) to be written, and the data is written into the file sequentially by node number (node 0 first, node 1 next, etc.) This mode allows the amount of data written by each node to be different; the buffer on each node can have a different size. Nodes that have no data to write may set their buffer length to be zero shown in Figure 4.8.

The CMMD I/O library only provides UNIX-level I/O calls. So there are potential problems in using Fortran 77 read/write functions, which use buffered I/O. Buffered I/O calls are permitted in all nodes. However, they are not guaranteed to work as expected in *CMMD-sync-seq* mode. Since CMMD relies on the C run-time library to provide the buffered I/O code, CMMD has no control over the timing of when a buffer spills and actually performs a read and write.

Buffered I/O, by its nature, removes control from the user over when the underlying read or write occurs. It is difficult or impossible for the programmer to guarantee synchronous calls across the partition to the underlying I/O routines.

This restriction results in a difficulty for Fortran I/O under this mode, as the only provided Fortran I/O mechanisms are built on top of a layer of buffering. To overcome this restriction, CMMD provides a global write function which utilizes UNIX write mechanisms to perform global write under *CMMD-sync-seq*.

The algorithm implemented on the Intel machine is almost the same as the one described above. There are two points to be mentioned. One is that the slab partition is very good for matrix fill but may not be good for factor/solve using ScaLAPACK since the slab partition may not have good load balancing for ScaLAPACK. The other is that the moment matrix does not need to be written out in the Intel and IBM implementations.

4.3.4 Fill the Right-Hand Side Vectors

The computation for filling the right-hand side vectors does not take a lot of time. However, it could be significant when the number of the vectors increases. The way it is implemented is to assign one node to fill one right-hand side vector when the total number of the vectors is less than the number of the nodes in the system. The block decomposition is employed as in the previous subsection when the number of the vectors is more than the number of the nodes of the system. After being filled, the vectors are written to a SAD file or a Data Vault file using a CMMD global write function with the *CMMD sync-seq* mode. The order of the file is the first vector and the second vector and so forth.

Let N_{tr} and N_{po} be the number of the transmitters and the number of the polarizations, respectively. The total number of right-hand side vectors is M, where $M = N_{tr} \cdot N_{po}$. The excitation vectors filling algorithm is presented in Figure 4.9.

As with the matrix fill, the algorithm for the Intel and IBM implementation is the same except for no write operation.

Fill the Right-Hand Side Vector Algorithm

```
1. Loop over the transmitters;
    (a) Loop over the polarizations; Find out the node that should do the work;
    (b) Find out the local index of the vector within the node;
    (c) Call the excitation vector fill code which is the same as the sequential one;
2. Compute the amount of data in each node for the vectors;
3. Output the vectors to a storage device in parallel.
```

Figure 4.9: The pseudo code for the fill RHS vectors algorithm

4.3.5 LU Factor and Solve

In this section, a data-parallel implementation to perform LU decomposition of the moment matrix and solution of the matrix equation will be presented.

As we mentioned in Section 4.1 that the factor/solve code is a completely independent program linking the rest of the ParaMoM-MPP with a an SDA file or a Data Vault file. The CMSSL, a mathematical library on the CM-5 system, uses a data-parallel programming model. A program which can call a CMSSL function must be written either in CM Fortran or C-Star. Both computer languages are extensions of the sequential traditional computer languages Fortran 77 and C, respectively.

The linear solver used for dense systems consists of the Gaussian elimination (LU decomposition) and back substitution solver. For numerical stability we have chosen the LU factorization with partial pivoting. The CMSSL's LU routines use the combination of blocking and load balancing. Blocking means routines operating on and transferring blocks of data rather than single data elements. Blocking reduces vector-vector operations and increases matrix-vector operations, which can give very high performance when local to a processing element. ScaLAPACK partition is controlled by the user.

The elimination operation presents a load balancing problem using a slab block decomposition. The load balancing provided in CMSSL uses cyclic ordering to achieve load balancing. Let the columns be processed in cyclic order (column 1 of the first column of processing elements, the column 1 of the second column of processing elements, and so on). Then as columns are eliminated, the active subgrid on each processing element shrinks, but no processing element becomes completely inactive until the last column is eliminated from some column of processing elements. That is, it increases the number of processing elements that are active at any time.

The combination of these two strategies is called Block Cyclic Ordering. In Block Cyclic Ordering, a routine computes on a block of columns or rows as a unit. A routine performs a block update of a number of columns instead of a single column at a time. Thus, the load balancing scheme processes blocks, rather than single columns, in cyclic order. Assuming that the block size is chosen to be b, the LU factorization routine eliminates columns and rows in the following order:

- Columns 1 through b of the first column of processing elements, along with rows 1 through b of the first row of processing elements.
- Columns 1 through b of the second column of processing elements, along with rows 1 through b of the second row of processing elements, and so on.

After eliminating b columns from each column of processing elements, the routine returns to columns b+1 through 2b of processing element column 1, and so on.

Letting a linear system be $AX = B$, the forward elimination gives

$$A = LU \tag{4.7}$$

Let $UX = C$, and then $C = L^{-1}B$. Following the back substitution

$$X = U^{-1}C = U^{-1}(L^{-1}B) \tag{4.8}$$

where B and X are $N \times M$ matrices, and L and U are the $N \times N$ lower triangular and upper triangular matrices respectively. When $M \neq 1$, (4.8) provides the opportunity to take advantage of data-parallel paradigm in this implementation.

The CM Fortran Utility Library's parallel I/O is used to open a SDA or Data Vault file and read in the moment matrix and the right-hand side vectors. The 'SO' I/O mode function is used (see [75]). Although, the 'SO' I/O mode is slower than the 'FMS' I/O mode, it provides the most portability across CM configurations and execution models. The LU routine is called after reading in the moment matrix and the right-hand side vectors. The lower and upper triangular matrices are written to the moment matrix to save memory space. The back substitution is applied to obtain the solution vector or vectors. Once again, the 'SO' I/O mode is used to write the solution vectors to a storage device. The LU/SOLVE program reports the Mflops achieved for the LU decomposition and back substitution separately.

As mentioned in the previous section, ScaLAPACK is used for factorizing the moment matrix and back substitution solution on the Intel machines and IBM SP-1. The performance is very much dependent on the data partition. Achieving good node-level performance on the IBM SP-1 is relatively easy, because each node is a general purpose RISC processor and there are no node-level vectorization issues.

4.3.6 RCS Computation

The RCS computation program implemented on the CM-5 is written in Fortran 77 with the CMMD message passing library. The first task is to read in the current coefficients from a storage device. The CMMD global-read function, which is the same as the CMMD global-write described in the matrix fill subsection, is employed to carry out this task. Then, the far field contributed by all currents is computed. Finally, the radar cross section is computed. The far field computation is the major task discribed in this subsection. There are two cases; one is for a single current vector and another is for multiple current vectors.

Let N_{tr} and N_{re} denote the numbers of the transmitters and receivers, respectively,

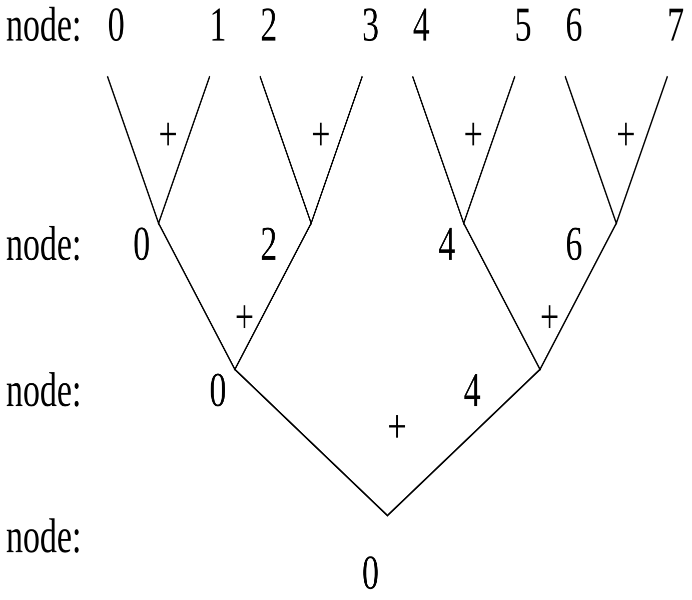


Figure 4.10: Divide and conquer algorithm for global summation

and N_{pt} and N_{pr} be respectively the numbers of the polarizations of the transmitter and the receivers.

For a single current vector, if N/p is a small number then only one node is active and the rest of the nodes stay inactive. If N/p is considerably larger it is worthwhile to divide the current vector into p pieces and assign each node to do some work. The result of each node is summed up in node 0. For M current vectors, if $M \leq p$, then make the first M nodes do the computation and the rest of nodes remain idle. If $M \geq p$, then each node is associated with a slab of the current matrix. The result of each node is again summed up in node 0. The divide and conquer algorithm is applied to sum the result of each node in node 0. The algorithm is illustrated in Figure 4.10, where $p = 8$.

A pseudo code of the algorithm implemented is given in Figure 4.11.

Parallel Far-field Computation Algorithm

```
1. Loop over the transmitters;
    (a) Loop over the transmitter's polarizations; Find out the node that should do the work;
    (b) Find out the local index of the vector within the node;
        Loop over the receivers;
        Loop over the receivers' polarizations;
            • If node index is not equal to the selected one go back to 1;
            • The selected node computes the far field produced by the current due to the transmitter with the polarization along a given direction.
2. Sum the far field for all current vectors;
3. Compute the radar cross section from the computed far field.
```

Figure 4.11: The pseudo code for the RCS computation algorithm

4.3.7 Out-of-Core Algorithm

Solution of very large problems requires out-of-core matrix filling and solution. The out-of-core matrix solution would be done using a vendor-specific out-of-core solver, such as ProSolver DES [80] on the Intel Paragon or the out-of-core CMSSL solver on the CM-5. Here, we only consider the development of out-of-core filling on the CM-5 system. The reason for developing an out-of-core filling algorithm is that the matrix is too large to be stored in the main memory of the system. We have to fill one portion of the matrix at a time and write it to a file, and then fill another portion of the matrix and write it out, and so on. Compare this with the in-core matrix fill algorithm, where the matrix is filled once then written out once. The main idea of designing an out-of-core algorithm is to modify the in-core filling algorithm structure and fill a portion of the matrix instead of the whole matrix.

Similarly, the issues involved with parallel out-of-core algorithm design are the problems of decomposition, load-balancing and communication. As with the in-core fill algorithm, the no communication approach is adopted here. The important question for out-of-core filling is how to decompose the problem into a set of small problems which can be fitted in in-core memory.

Before answering the question, we assume that the matrix, with N rows and columns, contains N_b bytes; the system has M bytes of available memory attached on each node; the number of processors is p; and the total number of out-of-core filling is N_{out}, where N_{out} has to satify the inequality

$$N_{out} > \frac{N_b}{pM} \tag{4.9}$$

When the column block decomposition is used, each in-core fill is to fill a slab of matrix. The size of the i^{th} out-of-core slab N_i is restricted by

$$N = \sum_{i=1}^{N_{out}} i \cdot N_i \tag{4.10}$$

and

$$N_i = n_i \cdot p \qquad i \neq N_{out} \tag{4.11}$$

where n_i is the number of columns for each node to fill at the i^{th} out-of-core fill. At the last out-of-core fill (or $i = N_{out}$), the number of unfilled columns is $N_{N_{out}} = N - \sum_{i=1}^{N_{out}-1} i \cdot N_i$. The job to fill the $N_{N_{out}}$ columns is distributed into p nodes as evenly as possible. Let $q = mod(N_{N_{out}}, p)$ and $d = \frac{N_{N_{out}}}{p}$. A node fills the $(d+1)$ columns if its index is less q, otherwise it fills d columns. The worst load balancing is when some nodes have one column more to fill than the other nodes.

Now, we can discuss our out-of-core fill algorithm since the data decomposition is decided. Based on the application and the particular system architecture, one supplies N_{out} and N_c which is the maximum number of columns that can be held in each node's memory. Let each node go through a loop of number of out-of-core procedures (from 1 to N_{out}). Each node calculates N_i for the i^{th} out-of-core fill and sets the global upper and lower bound. For example, the global lower bound is one and upper bound is N_1 for $i = 1$; and the global lower bound is $N_1 + 1$ and upper bound is $N_2 + N_1$ and so on. Each node fills a portion of the matrix in the same way as the in-core fill algorithm. However, each node does not pay any attention to the columns which fall outside the fill bound. After every node has completed the desired filling, a global synchronous write command is issued to write the portion of the matrix into a file. Then, each node examined goes back to the loop.

A pseudo code of the out-of-core matrix fill is shown in Figure 4.12

We implemented this algorithm and tested it on the CM5 for the $N = 9882$ case. The results are summarized in Table 4.3. It is seen that there is relatively little difference between out-of-core and in-core fill times.

Parallel Moment Matrix Out-of-Core Fill Algorithm

```
1. Loop over the number of out-of-core procedures, from one to $N_{out}$;
2. Calculate the global upper-bound and lower-bound;
    (a) Loop over source patches from one to $N_f$;
    (b) Loop over three possible edges on the source patch;
        i. Find the global index (i) of the edge;
        ii. Compute the node index in whose slab includes the column whose number is the same as the global edge index ;
        iii. If (i) is not in [lower-bound, upper-bound], go to 2(b)i;
        iv. Compute the local index for the location of the column in the node;
        v. The selected node increases its job counter by one and stores the information of the edge (the global edge index, the local edge index, and the local position in the node);
    (c) Those nodes whose job counter is 0 go back to 1;
    (d) Loop over field patches from one to $N_f$, the same as in the in-core fill algorithm;
3. Write the portion of matrix out using CMMD_global_write;
End out-of-core fill.
```

Figure 4.12: The pseudo code of the parallel out-of-core fill algorithm

Table 4.3: The time comparison between the out-of-core fill algorithm with $N_{out} = 10$ and the in-core fill algorithm on the CM-5.

Matrix Fill algorithm	Nodes	Fill Time (Sec.)	Write Time (Sec.)
in-core	256	1841	3.74
in-core	64	7364(estimated)	
out-of-core	64	7883	30.77

Chapter 5

Performance and Numerical Results

In this chapter, we present performance measurement data for the parallel algorithms, which are given in the previous Chapter, implemented on three parallel platforms. Measuring the performance of the parallel implementation, we present the following two test procedures: 1) with a fixed machine size, we change the problem size to see the run time of all components of the implementation; 2) with a fixed problem size, we change the machine size to see the run time and speed-up for the dominant parts of the implementation. The scalability analysis is given for each parallel algorithm in the parallel implementations. Portability is demonstrated by porting the PVM implementation from Intel machines to IBM SP-1 with little work.

The numerical accuracy of the algorithms developed in Chapters 2 and 3 is investigated by running test cases provided by the Electromagnetic Code Consortium (EMCC). The results obtained by the ParaMoM-MPP code are presented for comparison with measurements provided by the EMCC done on their well-known benchmark targets. A discussion of the suitability of multiprocessing architectures for electromagnetic scattering problems is given. A conclusion drawn from this work is presented in this Chapter.

This chapter is divided into 3 sections: Section 5.1 discusses the performance measurement and scalability analysis, Section 5.2 provides the numerical results of the EMCC testing cases, and Section 5.3 presents a discussion and comparison of the performance between the PATCH code and the ParaMoM-MPP code.

5.1 Performance and Scalability Analysis

In this section we analyze the performance and scalability of parallel algorithms and their implementation. We demonstrate scalability empirically using speed-up curves and show expressions for memory, computation, I/O, and communication costs versus problem size and machine size. The performance is measured in terms of the run time, speed-up, and Mflops. The first performance measurement is the run time for fixed machine size versus problem size. The time is reported for most components of the implementation on the CM-5 which gives the overall picture of the role of each component in terms of run time requirements within the code. We present the run times for three machine sizes for two Intel machines and two machine sizes for the IBM SP-1 computer. The section consists of three sub-sections. In Section 5.1.1, the CPU time is reported on a certain number of nodes of partition of CM-5, Intel Touchstone Delta, Intel Paragon, and IBM SP-1 of the particular architecture. In Section 5.1.2, the performance measurement in terms of speed-up is presented by speed-up curves. The scalability analysis for each portion of parallel algorithm in the parallel code is given in Section 5.1.3.

5.1.1 Performance Measurement for a Fixed Machine Size

For the purpose of measuring performance, the parallel RCS prediction code has been run for a set of conducting spheres. The number of equations goes from a few hundred to about ten thousand. There is only one right-hand (RHS) vector. The run time of each part of the parallel code gives a better view of the CPU time distribution from a user's perspective. The machine size (or the number of nodes in the system) is fixed but the problem size (the number of equations to be solved) is variable. For

Table 5.1: The running time in seconds on a 32-node partition CM-5

N	fill	w/r	far field	precomputation
4988	3295.2	9.1	4.3	11.3
3060	1207.0	4.7	2.8	6.0
988	137.5	3.0	0.5	2.1
468	31.5	4.2	0.6	1.1

Table 5.2: The running time in seconds on a 512-node partition CM-5

N	fill	w/r	far field	precomputation
9882	901.9	4.5	2.0	9.5
4988	237.9	3.7	1.1	4.8
3060	94.1	3.3	0.9	3.1
988	12.6	0.8	0.5	1.0
468	3.1	0.7	0.2	0.5

the CM-5 implementation, two machine sizes are chosen to present the run time. They are the 32-node CM-5 at NPAC and the 512-node CM-5 at MSCI/AHPCRC.[1] In Tables 5.1 and 5.2, the matrix fill time, the total time for extra I/O operation of CM-5 implementation, far-field time, and precomputation time is presented.

The fill, far field, and precomputation time are recorded in Tables 5.1 and 5.2 using a CMMD timer. The w/r time is the total time required for writing out the matrix and RHS vector and reading in the solution vectors using CMMD timer plus the time required for reading in the matrix and RHS vectors and writing out the solution vectors using CM Fortran timer. The w/r is elapsed time and the rest of the time is the CPU busy time. The last column in Table 5.2 is the Mflops generated by

---

[1] MSCI stands for Minnesota Supercomputer Center, Inc. and AHPCRC is Army High Performance Computing Resource Center.

the matrix factorization. The formulation computing the Mflops is

$$\text{Mflops} = \frac{\text{number of operations}}{\text{CPU time}} \tag{5.1}$$

For single precision complex type, the CMSSL LU function takes $\frac{8N^3}{3}$ operations to factorize an N by N matrix. The time used to compute the Mflops is the CPU busy time recorded by the CM Fortran timer.

From the run times listed in Tables 5.1 and 5.2, we see that the extra time required for the write/read operation to connect between the CMMD message passing program and a data-parallel CM Fortran program is indeed very short. The matrix fill and factor consume the most CPU time. Both the nature of the problem making the matrix fill code very inefficient to be vectorized and Fortran 77 being inaccessible to the vector units within CM-5 node make the matrix fill slow. On the other hand, since a CM Fortran code is able to utilize all vector units associated with the CM-5 node, the matrix factor/solve code is able to use the system much more efficiently.

We only list the matrix fill time and the matrix factorization time for the Intel machines and IBM SP-1 implementation because they are the dominant parts of the code. To present the run time on the Intel machines, we choose the 32-node Intel Paragon at NAS and the 64-node partition and 512-node partition of the Touchstone Delta at JPL/Caltech in Table 5.3.

For the IBM SP-1 implementation, we list the time data on the 58-node partition and the 32-node partition of SP-1 at Argonne National Laboratory (ANL). As with the Intel machines, only the fill time and the factor time are listed in Table 5.4.

For the purpose of performance comparison, the PVM implementation of the RCS prediction code is ported to a 8-node DEC Alpha farm cluster with 64 Mbytes memory per node. The Alpha cluster is connected by a FDDI-based Gigswitch. In Table 5.5, we list the CPU time and Mflops running on these machines.

5.1.2 Performance Measurement for a Fixed Problem Size

Another way to measure performance of a parallel algorithm is in terms of speed-up which is defined as the increase in performance (units of work per unit time) with

Table 5.3: The running time in seconds on an Intel 32-node Paragon, 512-node and 64-node Touchstone Delta.

N	Paragon 32-node		Delta 512-node		Delta 64-node	
	fill	factor	fill	factor	fill	factor
9882			277.9	411.6		
4988	692.9	371.5	72.5	80.7	525.6	280.9
3060	262.4	98.1	28.9	28.3	203.5	98.5
988	29.3	6.9	4.0	3.4	23.9	4.0
468	7.3	1.9	1.0	1.2	6.5	0.9

Table 5.4: The running time in seconds on the IBM 58-node and 32-node SP-1.

N	SP-1 58-node		SP-1 32-node	
	fill	factor	fill	factor
9882				
4988	217.9	1404.8	393.6	2487.8
3060	85.5	322.5	149.8	550.9
988	9.7	9.4	17.6	13.8
468	2.6	1.3	4.6	1.7

Table 5.5: N: matrix size, fill: matrix filling time in seconds, LU: matrix factor time in seconds

N	Platform	Nodes	fill	LU	LU(Mflops)
4992	Alpha+ Gigswitch	8	1420.0	1119.9	147.8
4992	IBM SP1+ Ethernet	8	1500.9	1804.7	116.3
4992	Intel iPSC /860	64	525.6	280.9	1027.1
4992	CM-5	32	3295.2	170.7	1938.8

increase in the number of nodes for a fixed problem size. The definition of speed-up is given in Chapter 4. The hallmark of good scalability is nearly linear speed-up on sufficiently large problems. For a large problem, we would like to halve the total solution time when we double the machine size. It should be noted that many parallel algorithms do not have good scalability due to the increasing dominance of internode communication as the number of nodes increases. Speedup is computed as

$$\text{Speed-up} = \frac{T_1}{T_p} \tag{5.2}$$

where T_1 is the time needed on one node and T_p is the time on p nodes.

It must be noted that it is not possible for speed-up to be linear for endlessly increasing machine size and fixed problem size, since at some point the number of nodes would exceed the total number of operations required to solve the problem! However, for large problems this is rarely a concern, and so we strive to achieve linear speed-ups.

For large problem sizes it may not be possible to obtain the one-node timing required to compute speed-up due to memory or computation time limitations. In these cases, T_1 is estimated based on T_p for the smallest possible p. This estimate is typically very accurate since the speed-up tends to vary linearly for small p on large problems.

Speedups for matrix fill on the CM-5 for several problem sizes are shown in Figures 5.1 and 5.2. The small problems are nearly linear for small numbers of processors (p is small), as shown in Figure 5.1, but for big p the use of additional nodes is less effective than on the large problems, as shown in Figure 5.2. Important point: the parallel algorithm reduces to the sequential algorithm in the special case $p = 1$, so the speed-up may be viewed as being relative to the original sequential algorithm. Thus we can fill the matrix 500 times as fast as we could with the original serial code implemented on a SUN SPARC workstation.

The large problems are nearly linear for a large p. Figure 5.2 shows that the large problem sizes yield better performance in terms of speed-up when a 512-node CM-5 partition is used. This partition is one of the largest CM-5 partitions available for general use. For small problems, the speed-up curves are almost flat after the certain number of nodes.

The matrix factor (LU decomposition) speed-up for the CM-5 is shown in Figure 5.3. Again the speed-ups are much better for the largest problem size than for the smallest. The speed-up curve may have an negative slope when a small problem is executed on a large machine. When the CMSSL LU with partial pivoting is used, the performance cost of pivoting is very much dependent on size and layout. The extra cost of pivoting is greatest for relatively small matrices. For very large matrices (using nearly all processing element memory), the performance of the factorization with pivoting is comparable to the performance without pivoting, whereas the solver remains about 50% slower for the pivoting version. However, for a small matrix, internode communication is actually dominating the total run time, which increases the cost of pivoting.

The speed-up curves obtained on the Intel machines are shown in Figures 5.4, 5.5, 5.6, and 5.7. Figures 5.4 and 5.5 show the speed-up curves on the Intel Paragon at NAS. Figures 5.6 and 5.7 show the speed-up curves obtained on the Intel Touchstone Delta system at JPL/Caltech.

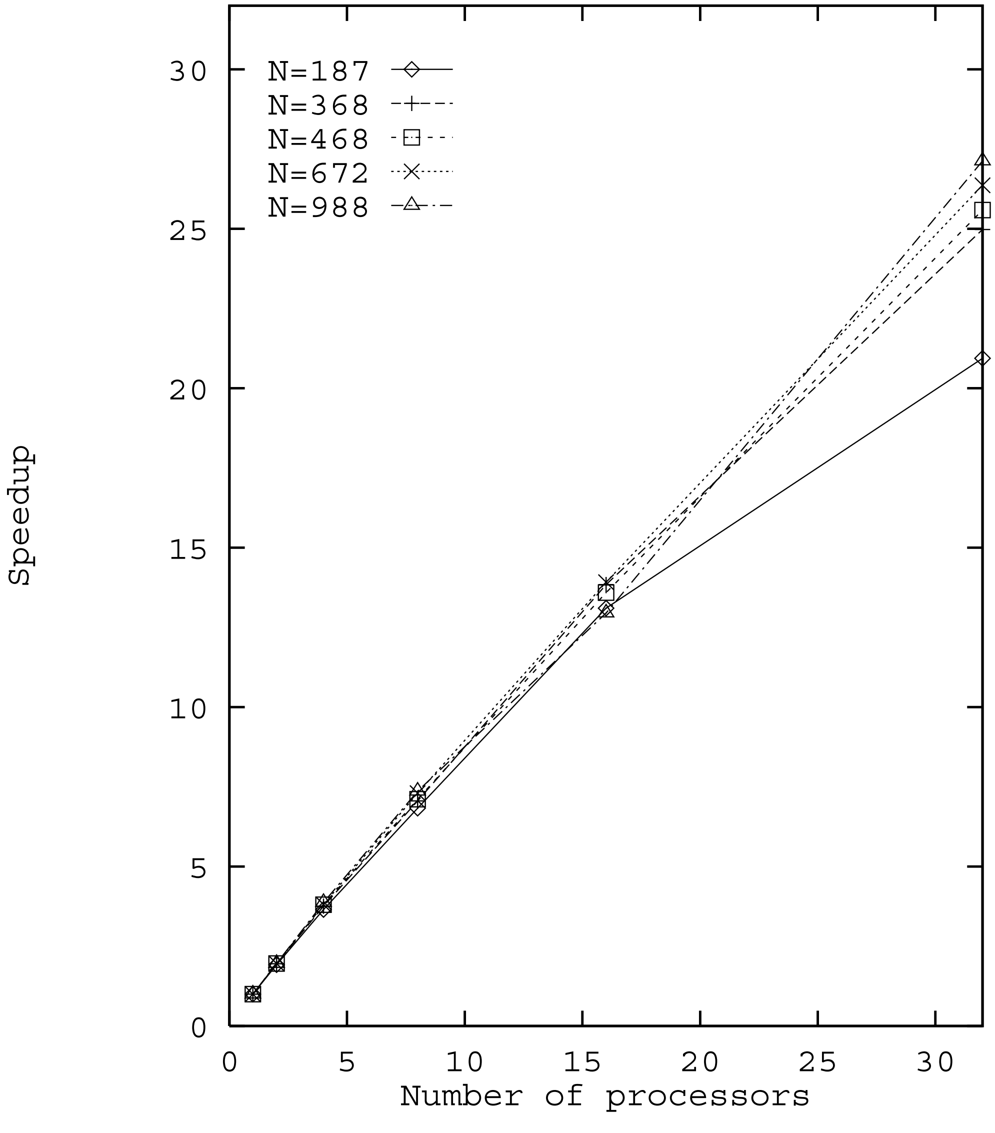


Figure 5.1: Performance of matrix fill portion of code implemented on small CM-5 partitions for fixed small problems

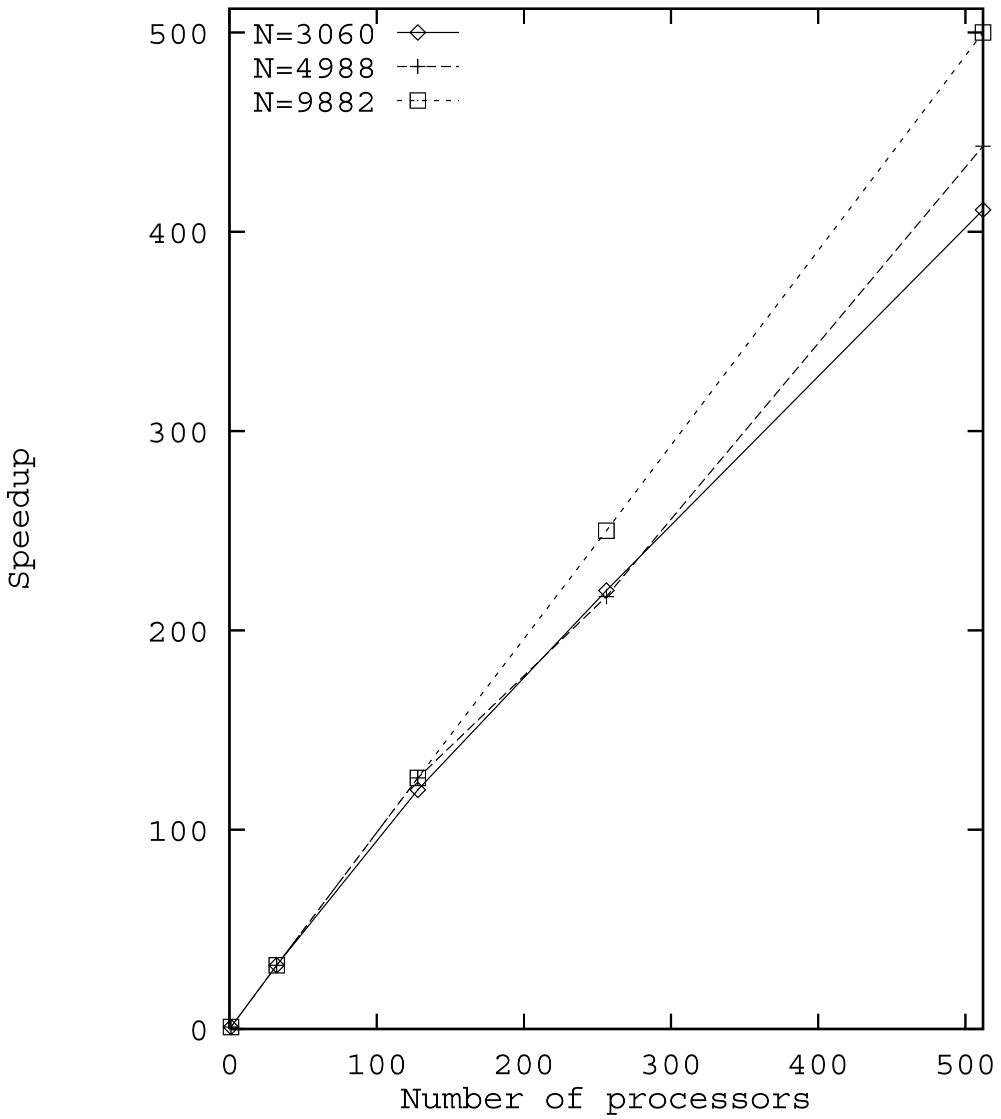


Figure 5.2: Performance of matrix fill portion of code implemented on large CM-5 partitions for fixed problem sizes

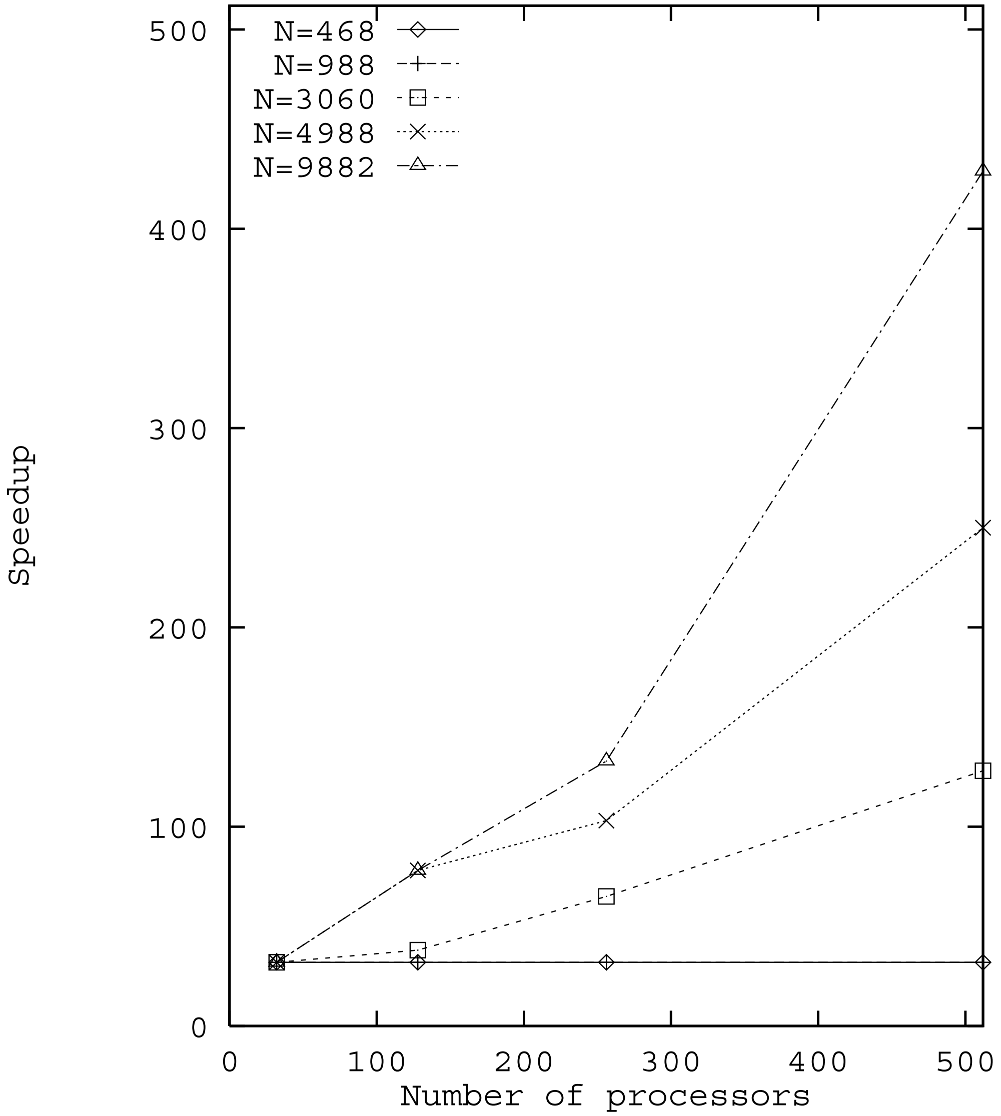


Figure 5.3: Performance of matrix factor portion of code implemented on large CM-5 partitions for fixed problem sizes

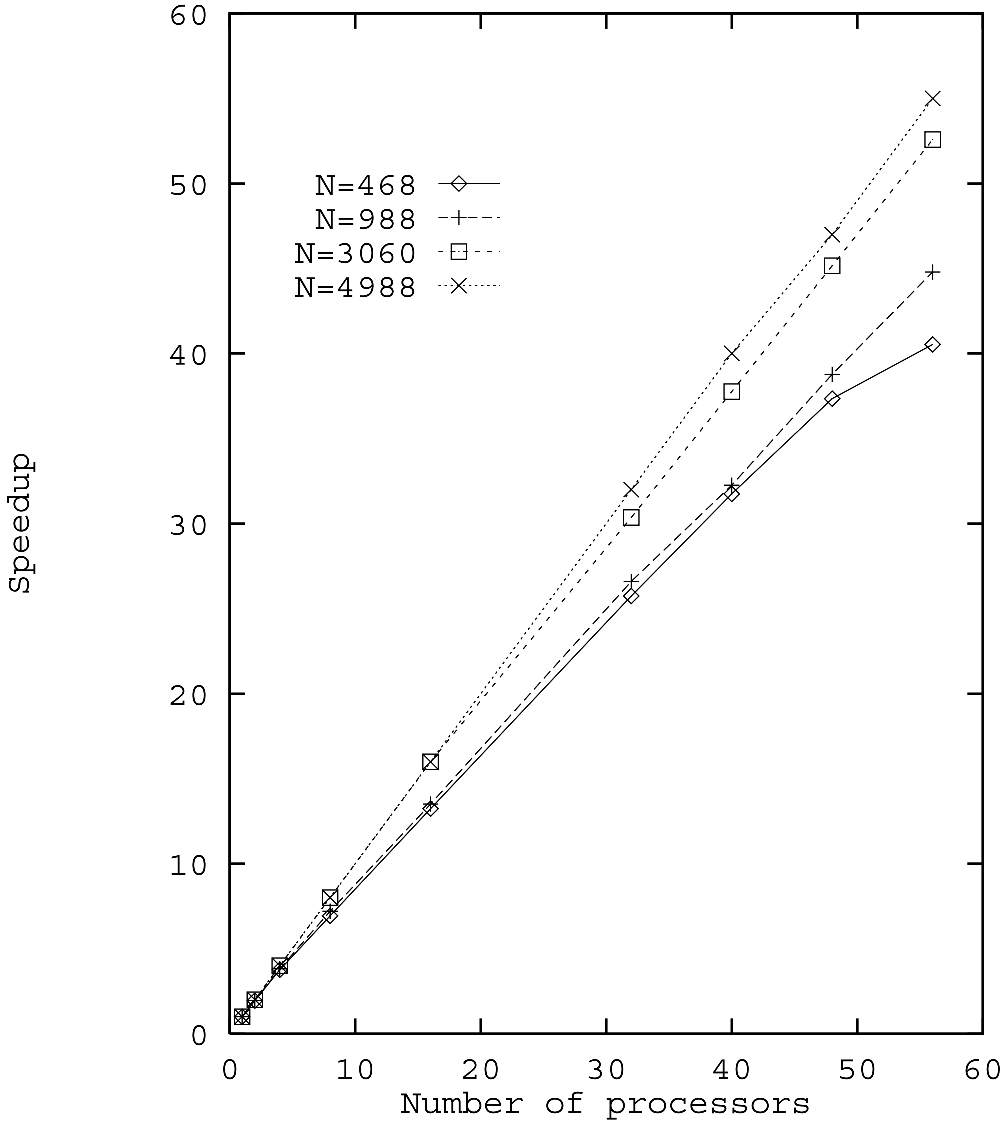


Figure 5.4: Performance of matrix fill portion of code implemented on the Intel Paragon system

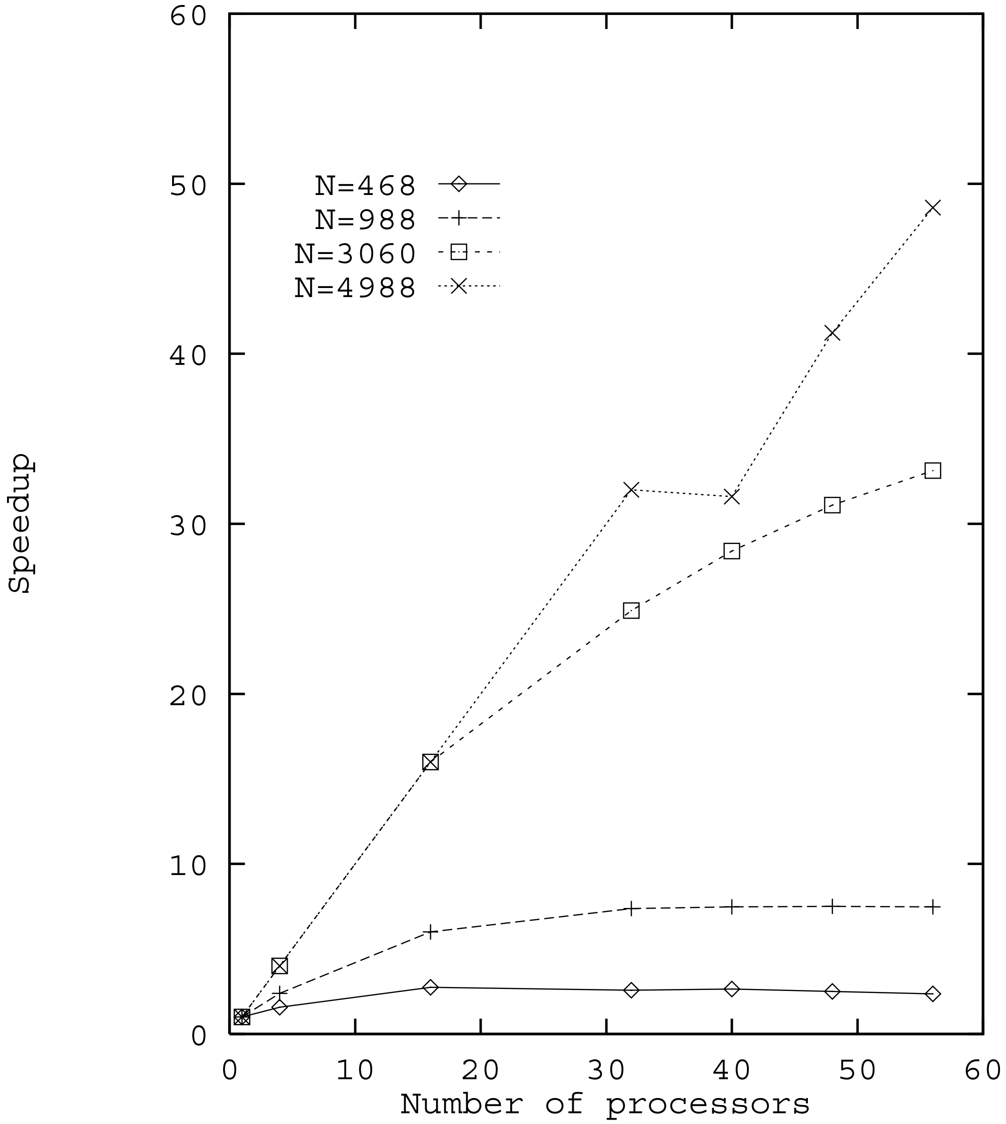


Figure 5.5: Performance of matrix factor portion of code implemented on the Intel Paragon System

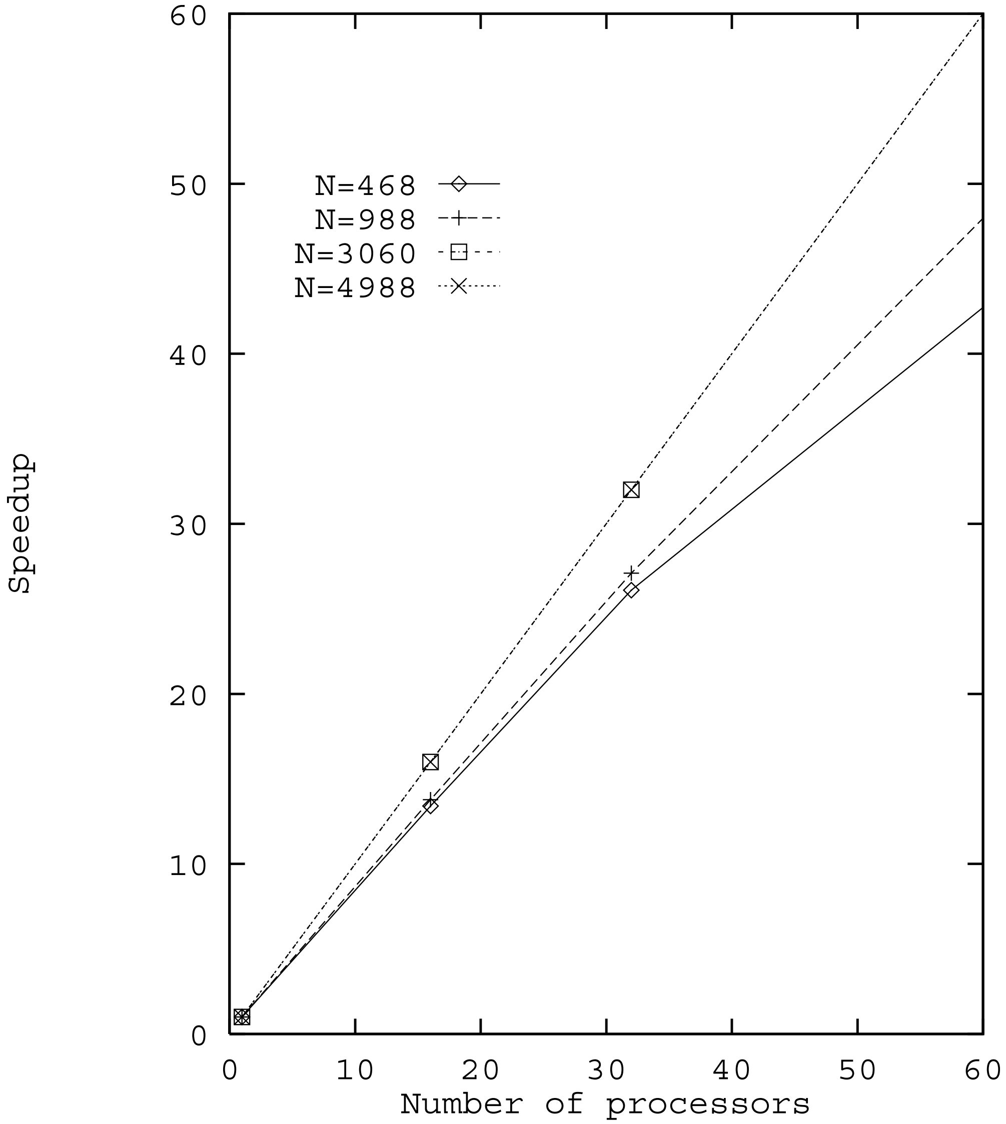


Figure 5.6: Performance of matrix fill portion of code implemented on the Intel Touchstone Delta system

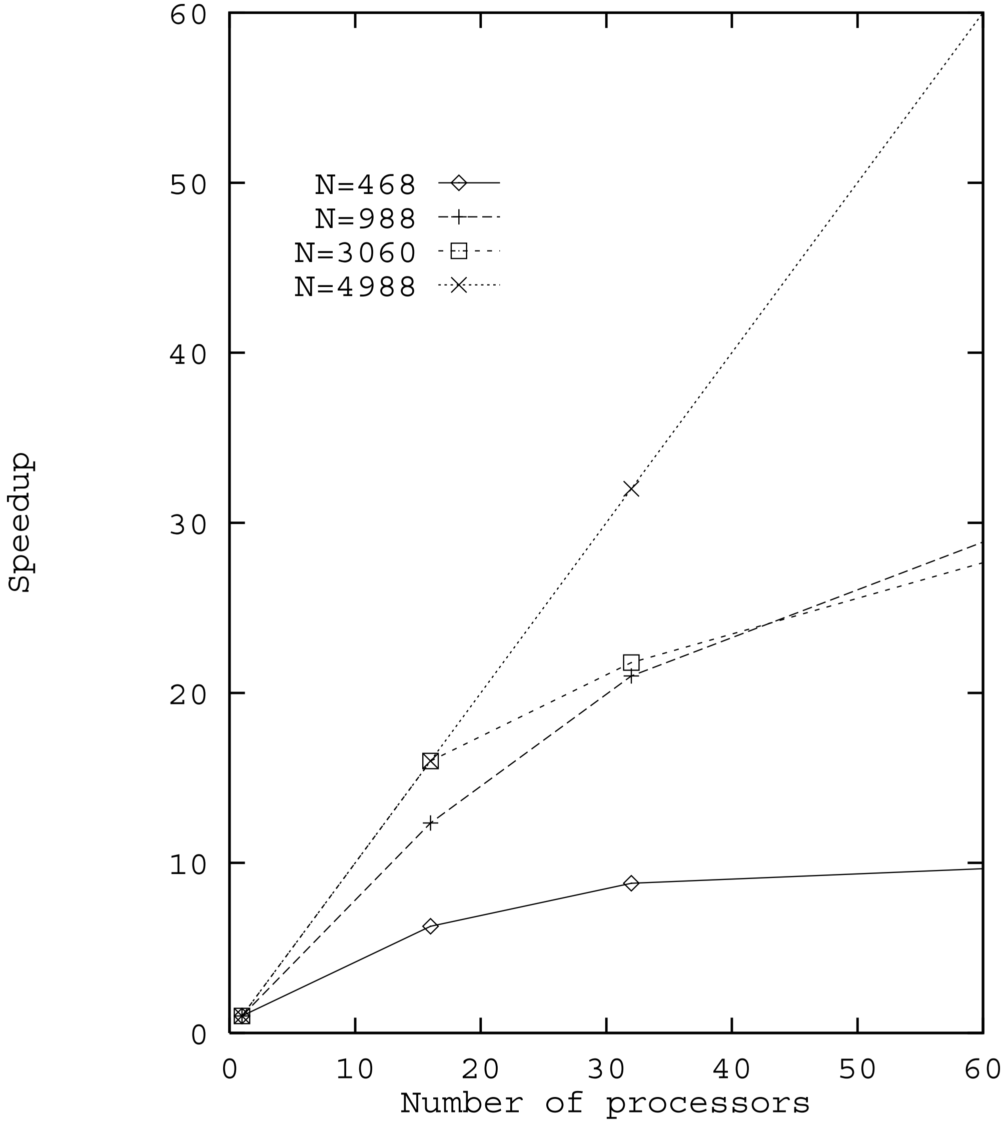


Figure 5.7: Performance of matrix factor portion of code implemented on the Intel Touchstone Delta system

The BLACS on these test runs is from an unoptimized pre-release of this package on SP-1's high performance switch with EUI-H protocol by ORNL's Clint Whaley. An optimized official release of the BLACS on EUI-H will be available in the near future. The speed-up curves obtained on the IBM SP-1 are shown in Figures 5.8 and 5.9.

5.1.3 Scalability Analysis

Six separate parallel algorithms are implemented. They are the setup phase, the precomputation phase, the matrix fill phase, the RHS vector (or vectors) fill phase, matrix factor/solve phase, and far-field phase. By reviewing Chapter 4 on parallel implementation, we know that the setup and precomputation employ the same parallel mechanism in the global broadcast operation. Therefore, only one of three algorithms will be to be analyzed. In the far-field algorithm, the far-field computation is carried out in parallel when there are multiple current vectors. The number of the current vectors is the same as that of the excitation vectors, and this number is usually limited to about thousand. These algorithms described in Chapter 4 are scalable. There is the same amount of work for each node when the number of the excitation vectors increases by the factor m and the number of the nodes in the system increases by the factor m. The RHS vector fill algorithm is very similar to the far field algorithm. It is scalable like the far field algorithm. We only consider to conduct a scalability analysis for the precomputation phase, the matrix fill phase, and the matrix factor/solve phase. The scalabilities of these components are analyzed separately below.

Precomputation Phase Scalability

In the precomputation phase, several arrays are computed in a distributed fashion. These arrays contain the locations of the numerical integration points, as well as the values of the basis functions and their divergences at those points. These arrays are of order N_f, where N_f is the number of surface patches, which is proportional to the number of basis functions or matrix size (the relation between N_f and the number of basis funcitons is given in Chapter 4). The work of computing these arrays is

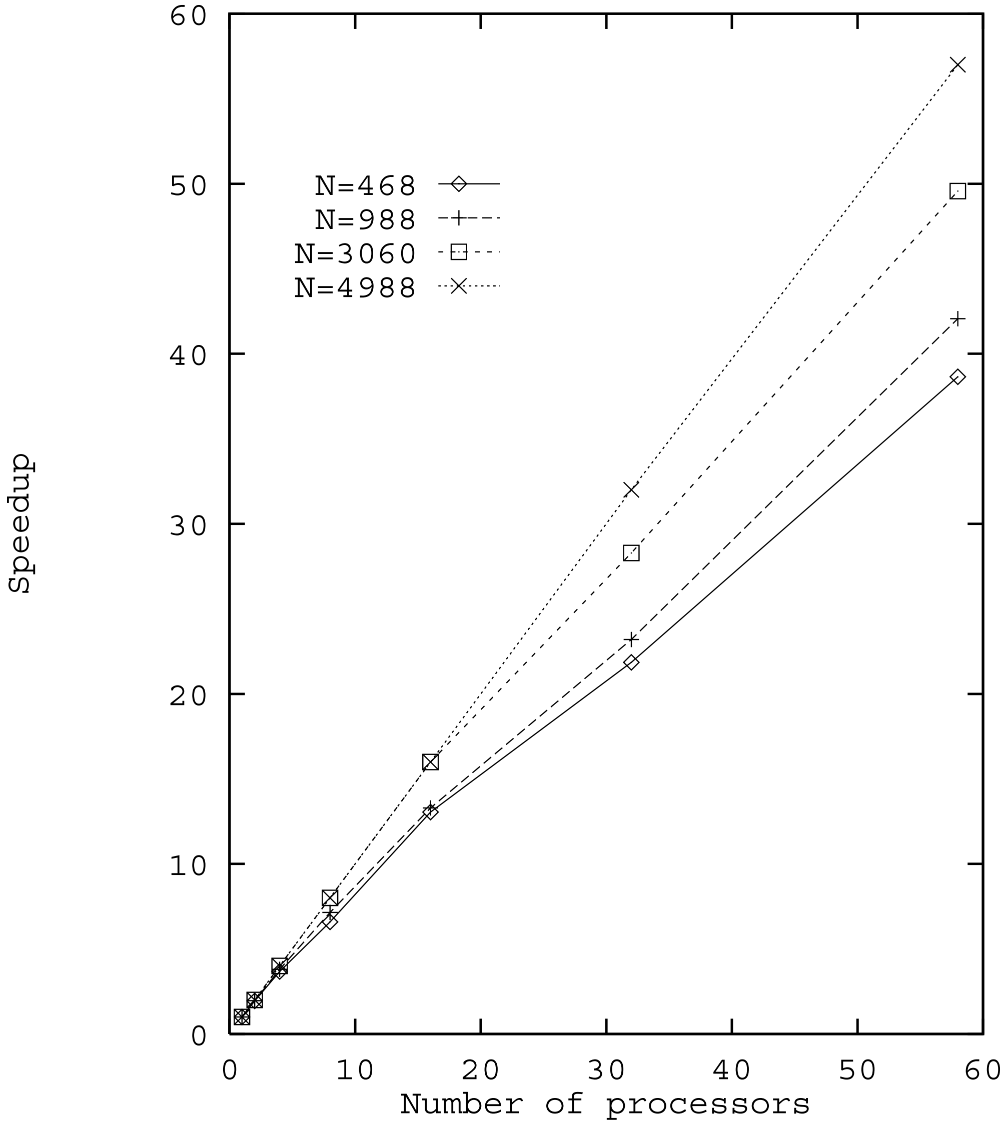


Figure 5.8: Performance of matrix fill portion of code implemented on the IBM SP-1 system

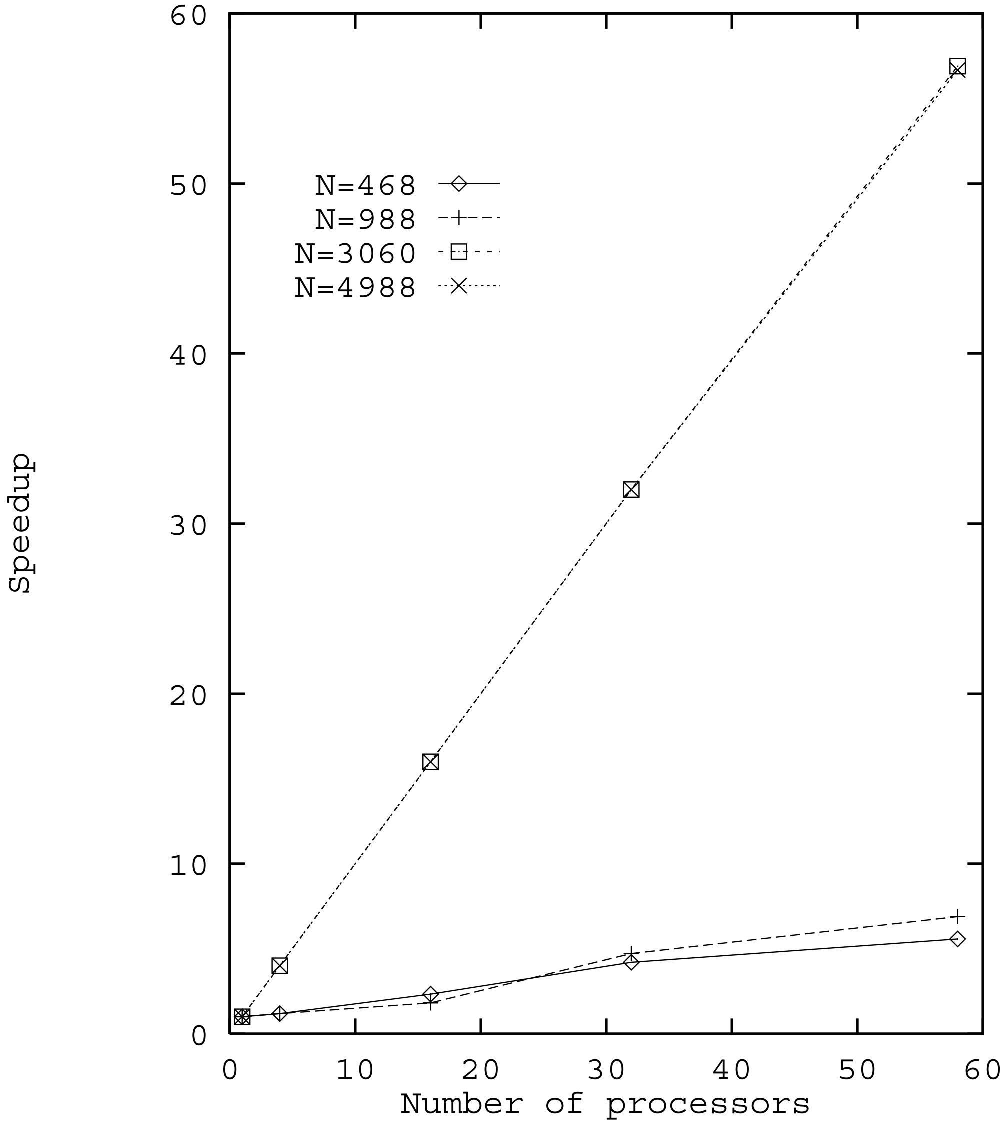


Figure 5.9: Performance of matrix factor portion of code implemented on the IBM SP-1 system

distributed almost evenly among the nodes.

One may ask what is the broadcasting complexity here. We will give a short analysis to ensure the broadcast operation used here will not be a bottleneck of our approach.

Assume that there are N_b elements in the basis function array, N_{db} elements for the divergence of the basis function array, and N_r elements for the position vector array. The total time T of this procedure is

$$T = T_c + T_{bc} \tag{5.3}$$

where T_c is the maximum computing time needed for a node, and T_{bc} is the total broadcasting time. T_c can be expressed as

$$T_c \approx t_{cb}(\frac{N_b}{p}) + t_{cdb}(\frac{N_{db}}{p}) + t_{cr}(\frac{N_r}{p}) \tag{5.4}$$

where t_{cb}, t_{cdb}, and t_{cr} are the times required to compute an element of the basis function array, the divergence of the basis function array, and the position vector array, respectively. The total broadcasting time can be expressed as

$$T_{bc} = p[t_l \log_2(p) + t_{tr}(\frac{N_b + N_{db} + N_r}{p})] \tag{5.5}$$

where t_l is the latency for broadcasting a message and t_{tr} is the time required to transfer a four-byte word from one node to another.

In the CM-5, global operations generally are extremely fast. A broadcast of a single word to all nodes take about 8 microseconds. The broadcast operations take longer for longer messages, reaching an aggregate bandwidth of 25 MB/sec on 32 nodes (see [74]).

The only potential liability in the precomputation is seen to be the $p \cdot \log_2(p)$ term. However, since the practical upper limit on p for the class of machines of interest is a few thousand, this term will never be drastically larger than the worst case we have run to date ($p = 512$). For reasonable problem sizes ($N \approx 10,000$) the setup and precomputation time with $p = 512$ was seen to be very small compared to the matrix fill and factor times. The ratio of fill to setup and precomputation remains approximately equal if p is increased by a factor of p' and N is increased by a factor of $\sqrt{p'}$. Since N is expected to increase as p increases, it is clear that the setup and precomputation will not become dominant under any practical circumstance.

Matrix Fill Scalability

No internode communication is performed during the matrix fill phase. However, the fill time is somewhat dependent on p due to the possibility of redundant computations in the parallel algorithm. As described in the previous chapter, each node computes the necessary patch-to-patch geometric interactions. There is some redundancy in this scheme in that more than one node may compute the same interactions when the nine basis functions defined for the source and field patches are not assigned to the same node.

In the slab partitioned algorithm, the amount of redundancy increases as the number of nodes increase up to a statistical limit; the probability that a patch-to-patch interaction will be computed more than twice is very small. The extreme case where the three basis functions on a source patch are assigned to three different nodes has extremely small probability. Therefore the upper bound on the worst case is obtained when each patch-to-patch interaction is computed twice. These computations typically represent less than 25% of the total fill time (this percentage is system dependent) for the sequential algorithm, and so the total work in the parallel fill is typically less than 1.25 times the total work in the sequential algorithm. In other words the lower bound on the fill speedup relative to the sequential algorithm is $\frac{p}{1.25}$.

Matrix Factor and Solve Scalability

Our approach in the matrix factor and solve phase is to utilize existing parallel LU matrix solvers. It is reasonable to assume that state-of-the-art solvers provided by a vendor or a respected sources such as the Oak Ridge National Laboratory have been designed to achieve scalable performance, and that the scalability characteristics are well documented [72]. Although it is impossible to characterize all parallel LU solvers in this report, we briefly describe the scalability of the ScaLAPACK solver as an example of the state-of-the-art.

ScaLAPACK uses a block-scattered decomposition of matrix blocks of size $r \times r$ distributed among a "virtual" $P \times Q$ array of processors (the total number of processors is $p = P \cdot Q$). If r is smaller than $\frac{N}{p}$ there may be multiple blocks per node.

The efficiency E is defined in the previous Chapter as $\frac{T_1}{T_p \cdot p}$, where T_1 and T_p are the solution times for one node and p nodes respectively. E may be thought of as the percentage of optimal speed-up obtained. For ScaLAPACK, E if given by the following formula [72].

$$E \approx \{1 + \frac{p}{N^2}[c_1 \log(P) + c_2]\frac{\alpha}{\gamma} + \frac{P}{N}[c_3 \log(P)\frac{\beta}{\gamma} + c_4] + \frac{Q}{N}[c_5\frac{\beta}{\gamma} + c_6]\}^{-1} \tag{5.6}$$

where

$$c_i \text{ depend on only r}$$

$$i = 1, \cdots, 6$$

$$\alpha = \text{communication latency}$$

$$\beta \sim \frac{1}{\text{bandwidth}}$$

$$\gamma = \text{time to compute one floating point operation}$$

Consider the special case $P = 1$, $Q = p$. This is the configuration used in the ScaLAPACK code. For sufficiently large N,

$$E \approx \{1 + \frac{p}{N^2}c_2\frac{\alpha}{\gamma} + \frac{p}{N}[c_5\frac{\beta}{\gamma} + c_6]\}^{-1} \tag{5.7}$$

In this case, we see that E is close to one provided that $N \gg p$. For smaller N, however, $\frac{p}{N^2}c_1 \log(P)\frac{\alpha}{\gamma}$ and $\frac{P}{N}c_3 \log(P)\frac{\beta}{\gamma}$ can be significant. On many multipcomputers, α is several orders of magnitude greater than γ. On a network with limited bandwidth, β is large and the term containing β may dominate.

5.2 Numerical Results

In this section, we present some numerical results computed by the ParaMoM-MPP code described in this thesis. The test targets in this study are provided by the Electromagnetic Code Consortium. All numerical results are checked against the measurements provided by the EMCC (see references [81, 82]). The incident angles are defined in Appendix B.

The parallel program requirements vary greatly and depend upon the EMCC benchmark target geometries, the frequency, and the model symmetries imposed. The numerical results are to provide a general idea of the accuracy obtained by using a parametric method of moments approach, especially the parallel code ParaMoM-MPP, in a MIMD with a distributed memory environment. In fact, the patch sampling rates used for the benchmark comparison runs are somewhat conservative and would be reasonable on a large problem. In most cases, the maximum edge length specified during target gridding was one eighth of a wavelength.

Depending upon the specific method used for RCS prediction, target descriptions may vary greatly. Of particular concern for several approaches, ParaMoM-MPP included, is the actual parametric model. Syracuse Research Corporation provides these models. All of the EMCC benchmark test cases indicated below were run on the NAS Paragon known as GRACE. The RCS values for both horizontal (HH) and vertical (VV) polarizations are plotted in dBSM the dB expression for the number of square meters that the RCS is as a function of the azimuthal angle which is defined in Appendix B.

The first example is the RCS from a flat plate in the shape of a wedge cylinder with gap. The incident wave has a frequency of 5.9 GHz with horizontal polarization. The receiver's polarization is also horizontal. The parametric patch model of the target is shown in Figure 5.10, where the maximum edge length is $\frac{\lambda}{8}$, and the number of triangular patches is 1008 with 553 nodes and 1560 edges. The radius of circular part of the wedge cylinder is λ and the length of its straight side is two λ.

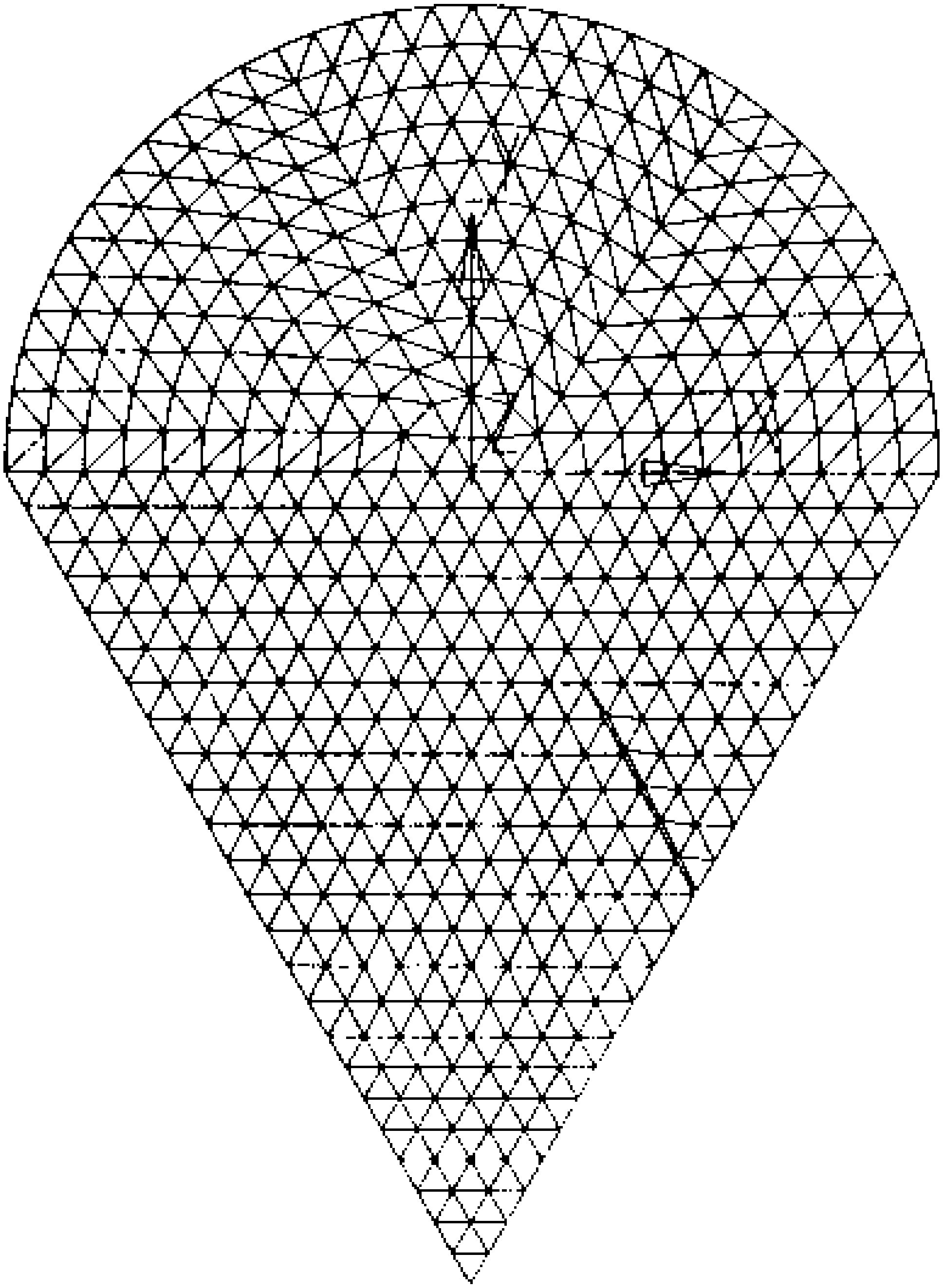


Figure 5.10: The wedge cylinder with gap geometric model

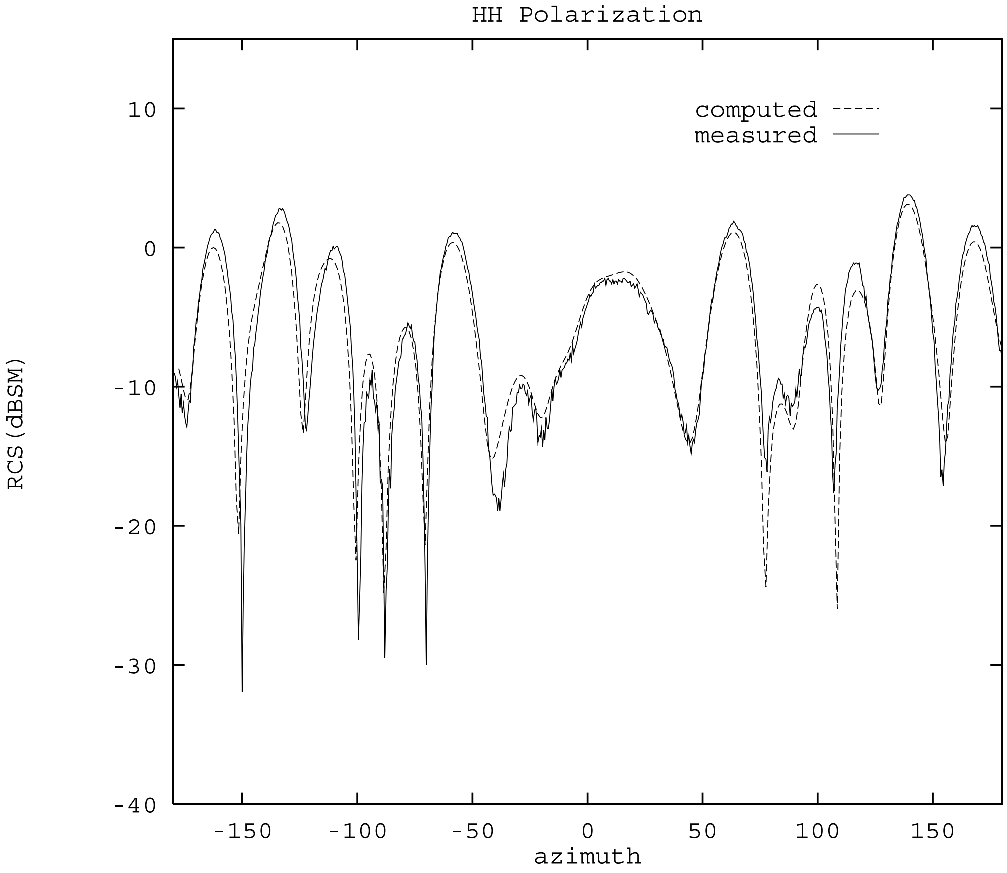


Figure 5.11: The RCS with HH polarization of a wedge cylinder with gap

The monostatic radar cross section for horizontal transmitter and receiver polarizations (HH) is shown in Figure 5.11. The result is obtained by running the ParaMoM-MPP on a 80-node partition Intel NAS paragon. Six Mbytes of memory are required to run this target. There are 360 excitation vectors to compute the monostatic radar cross section. The computation takes 309 seconds by a wall clock. The RCS, in dB λ^2, plotted against the azimuthal angle is shown in Figure 5.11 in a 10^o-elevation conical cut.

In Figure 5.12, NASA's half metallic almond parametric model is shown. The mathematical description for the NASA almond is as follows:

for $-0.41667 < t < 0$ and $-\pi < \phi < \pi$

$$x = d \cdot t \text{ Inches}$$
$$y = 0.193333d\sqrt{1 - \{\frac{t}{0.416667}\}^2}\cos\phi$$
$$z = 0.064444d\sqrt{1 - \{\frac{t}{0.416667}\}^2}\sin\phi$$

for $0 < t < 0.58333$ and $-\pi < \phi < \pi$

$$x = d \cdot t \text{ Inches}$$
$$y = 4.83345d\{\sqrt{1 - [\frac{t}{2.08335}]^2} - 0.96\}\cos\phi$$
$$z = 1.61115d\{\sqrt{1 - [\frac{t}{2.08335}]^2} - 0.96\}\sin\phi$$

where $d = 9.936$ inches. The total length of the almond is 9.936 inches. To take the symmetry into consideration, only half the NASA almond is to be modeled as shown in Figure 5.12. The plane of symmetry is the XY plane. The NASA's half almond is segmented by a curved grid with 1232 curved triangles, 1915 edges, and 684 nodes. The maximum edge length is $\lambda/8$.

The monostatic radar cross section for both the HH and the vertical-vertical (VV) polarizations at 7 GHz are plotted in dBSM as functions of azimuthal angle in Figures 5.13 and 5.14. The elevation angle is zero. There are 720 RHS vectors for this problem. This problem is run on a 32-node partition of the NAS Paragon. The memory required to run this problem in the system is 7 Mbytes per node. The wall clock time is 2817 seconds to complete the run.

A single metallic ogive's parametric patch model is shown in Figure 5.15, where three symmetry planes are used so one eighth of the body is actually modeled. The ogive is illuminated by an incident wave at 9 GHz. The ogival body is a classical test case for RCS. The analytical expression for the ogive is

$$f(x) = \{\sqrt{1 - [\frac{x}{5}]^2} - \cos(22.62^o)\} \tag{5.8}$$

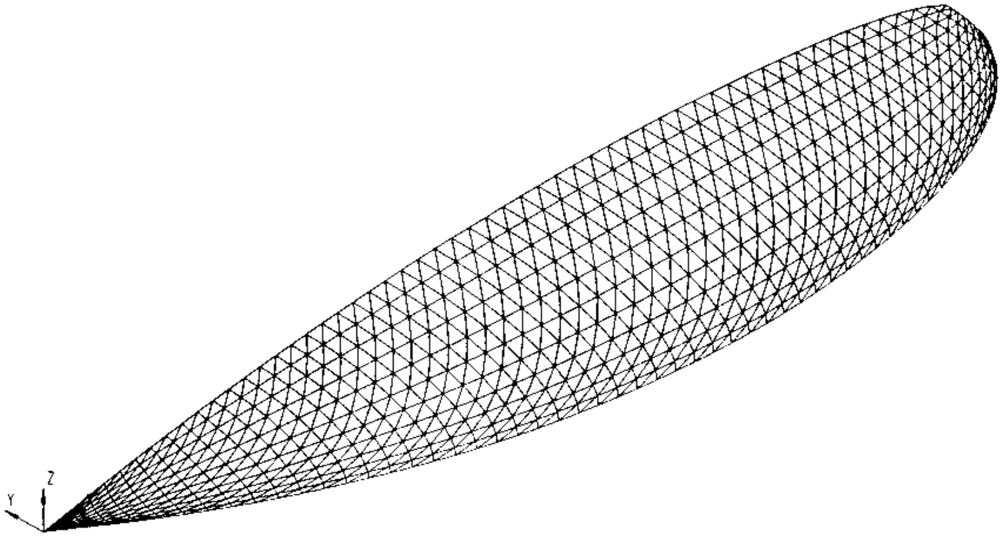


Figure 5.12: The NASA's half almond parametric geometry model

and

$$\begin{aligned} y &= \frac{f(x)\cos\phi}{1-\cos(22.62^o)} \\ z &= \frac{f(x)\sin\phi}{1-\cos(22.62^o)} \end{aligned} \qquad (5.9)$$

where x is greater than -5 inches and less than 5 inches, and ϕ is greater then $-\pi$ and less than π. The single ogive has a half angle of 22.62 degrees, a half-length of 5 inches, and a maximum radius of 1 inch. The single ogive is 10 inches long physically and about 7.6 wavelengths electrically. The single ogive is gridded into two different models. The first model utilizes three symmetry planes, so only one eighth of the single ogive needs to be modeled. This one-eighth single ogive is modeled by 2088 curved triangles with 3208 edges and 1121 nodes. The maximum edge length is $\frac{\lambda}{12}$. The second model uses no symmetry plane. It consists of about 5000 curved patches, and 7332 edges.

The monostatic radar cross section characteristics for both HH and VV polarizations are plotted in dBSM as functions of the azimuthal angle in Figures 5.16 and 5.17.

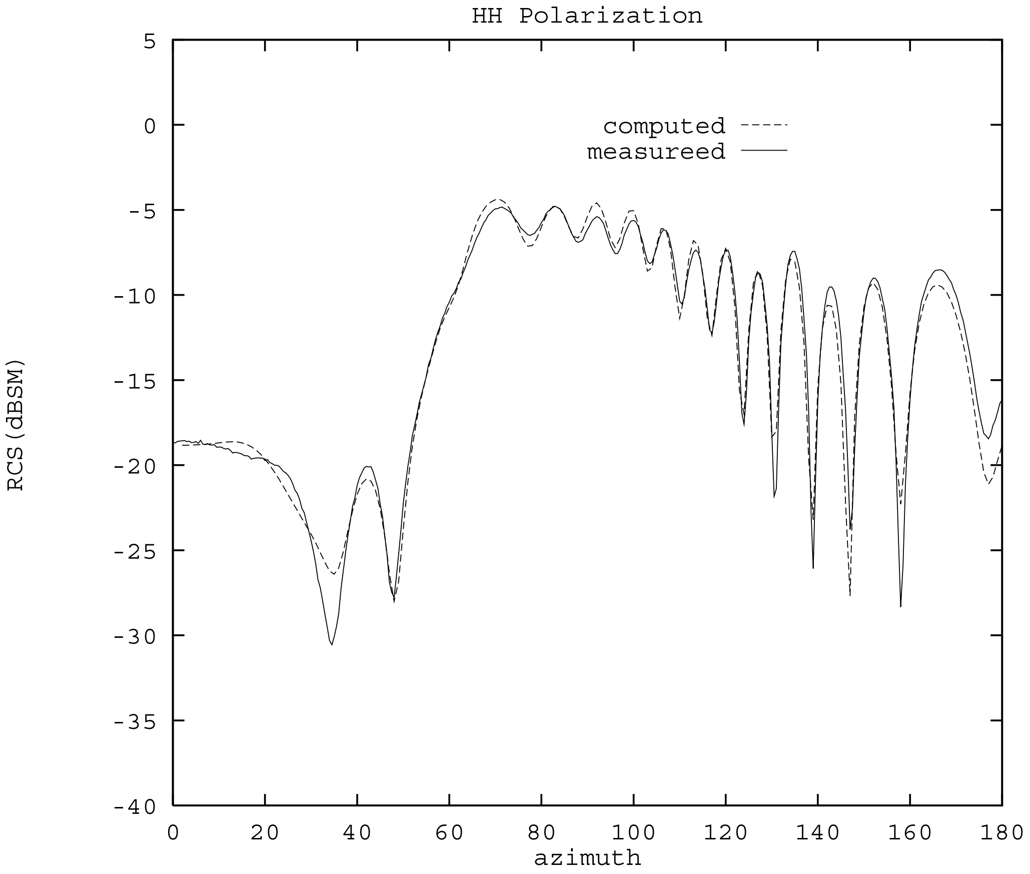


Figure 5.13: The RCS with HH polarization of a NAS almond

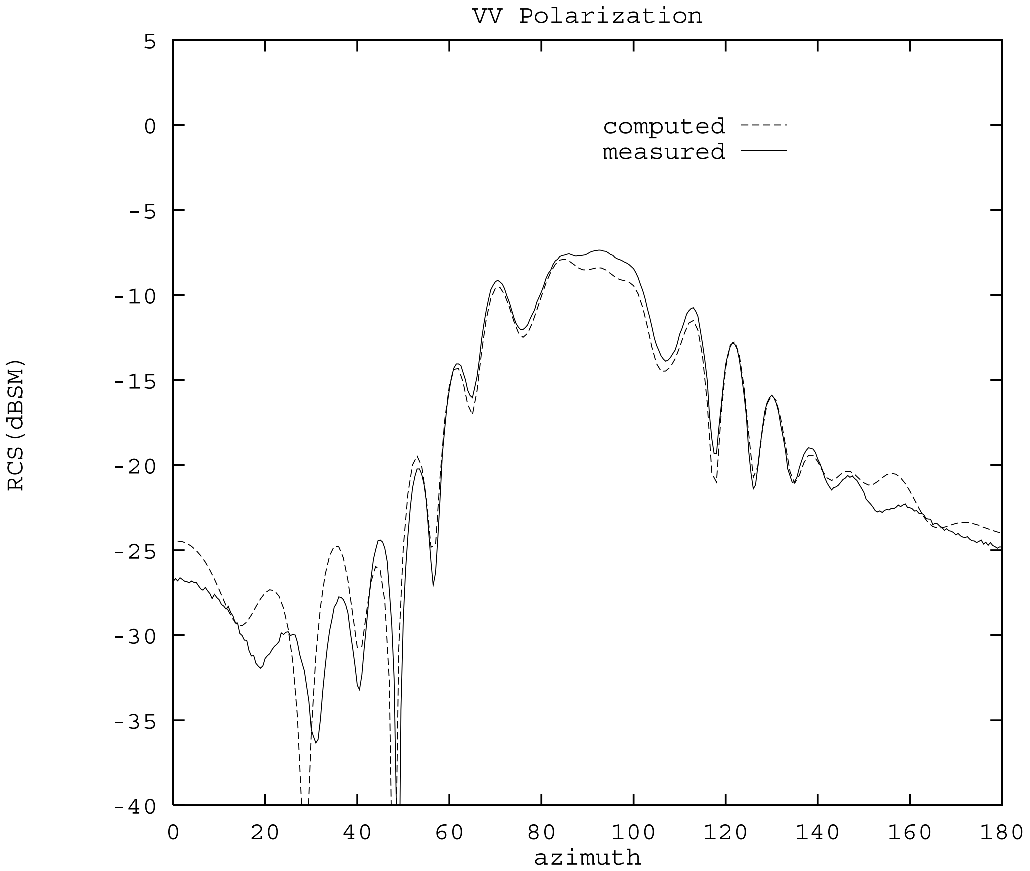


Figure 5.14: The RCS with VV polarization of a NASA almond

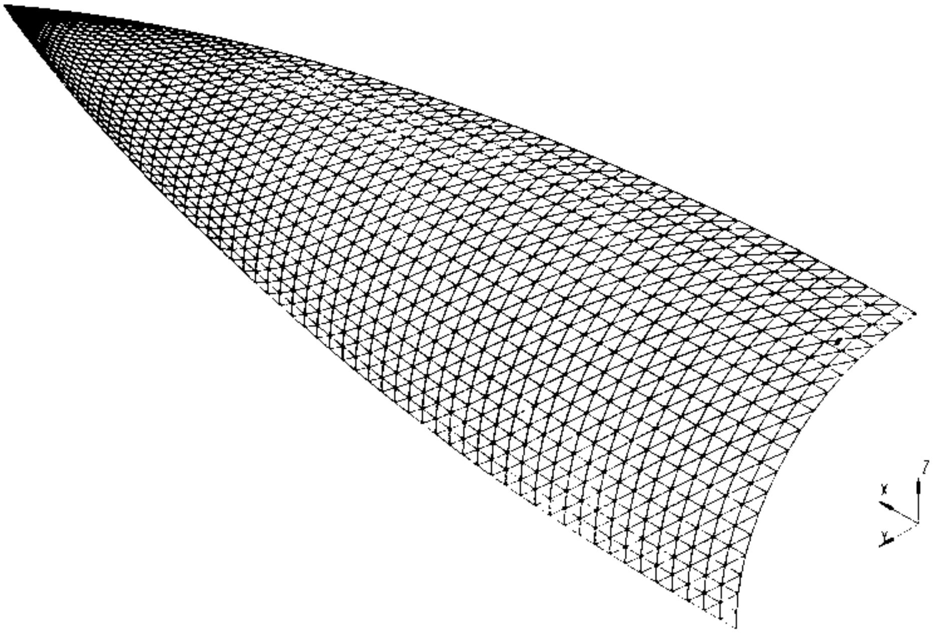


Figure 5.15: The single ogive parametric geometry model with symmetry

The elevation angle is zero. In Figures 5.16 and 5.17, and computed_without_sym denotes the model uses no symmetry planes, computed_with_sym denotes the target modeled with three symmetry planes. Although the number of the curved patches of the model without symmetry planes is much less dense than that with symmetry planes, Figures 5.16 and 5.17 shows good agreement. This demonstrates that the parametric patch method of moments with fewer unknowns still obtains good accuracy. There are 360 RHS vectors for this problem. This problem is run on a 64-node partition of the NAS Paragon. The symmetry model requires 9 Mbytes per node and takes 9648 seconds to complete the run. The model without symmetry planes requires 21 Mbytes per node and takes 3918 seconds to complete the run.

The double ogive consists of two different-size half ogives. One half ogive has a half angle of 46.4 degrees at the tip, a half length of 2.5 inches, and a maximum radius of 1 inch. The other half ogive has a half angle of 22.62 degrees at the tip, a half length of 5 inches, and a maximum radius of 1 inch. The parametric patch model of

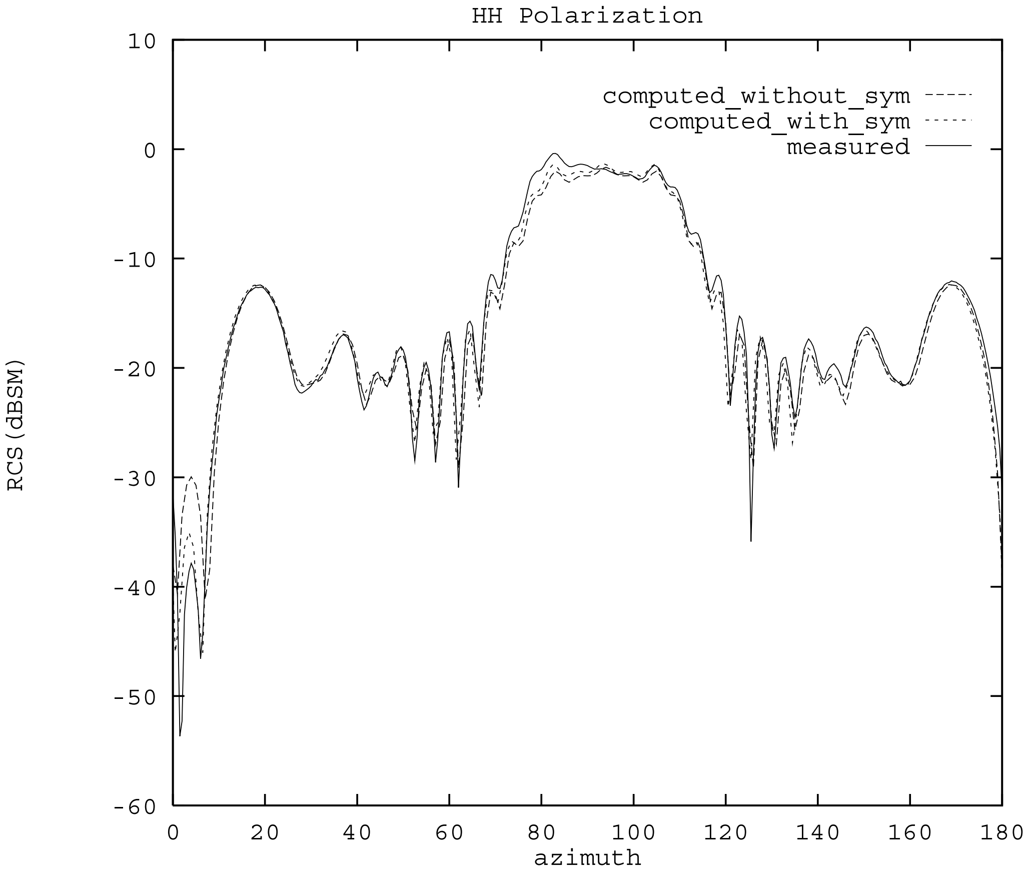


Figure 5.16: The RCS with HH polarization of a single ogive

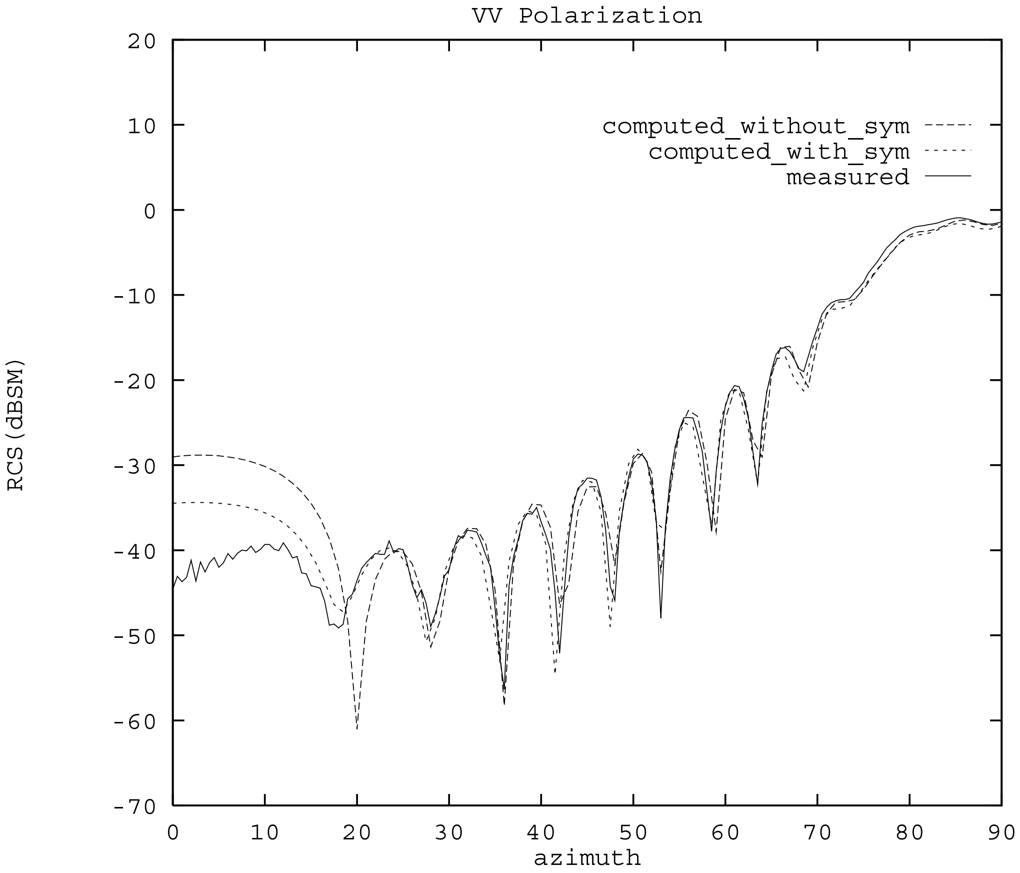


Figure 5.17: The RCS with VV polarization of a single ogive

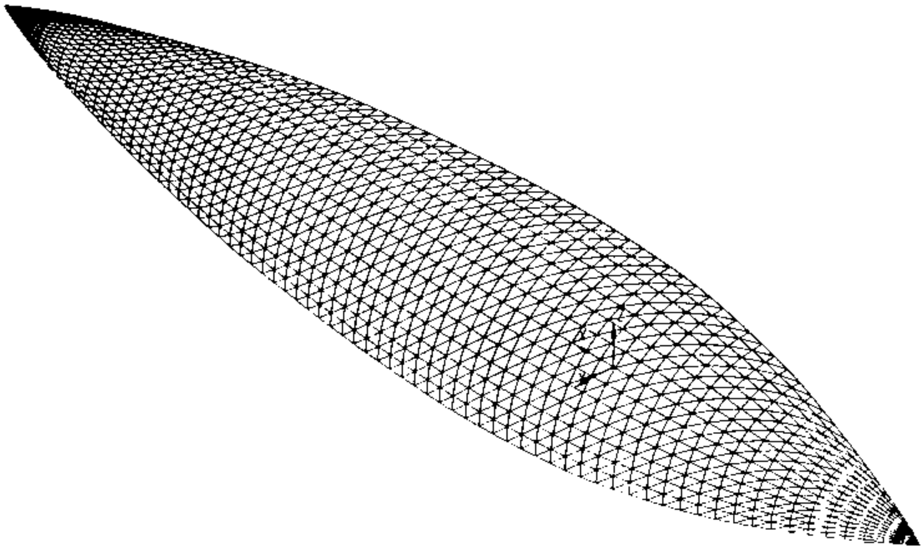


Figure 5.18: The double ogive parametric geometry model

the double ogive target is shown in Figure 5.18. The double ogive is 7.5 inches long or about 5.7 wavelengths long under the illumination of 9 GHz incident waves. The target surface is modeled by a set of 3822 curved triangles with 5772 edges and 1950 nodes.

The monostatic radar cross sections for both HH and VV polarizations are plotted in dBSM as functions of the azimuthal angle in Figures 5.19 and 5.20. The elevation angle is zero. There are 360 RHS vectors for this problem. This problem is run on a 64-node partition of NAS Paragon. The memory required to run this problem on that system is 17 Mbytes per node. The wall clock time is 2320 seconds to complete the run.

The metallic cone-sphere has a half angle of 7 degrees, and a sphere radius of 2.947 inches. The length of the cone part is 23.821 inches, and the side of the cone is tangent to the sphere, to provide a smooth transition and minimum diffraction at

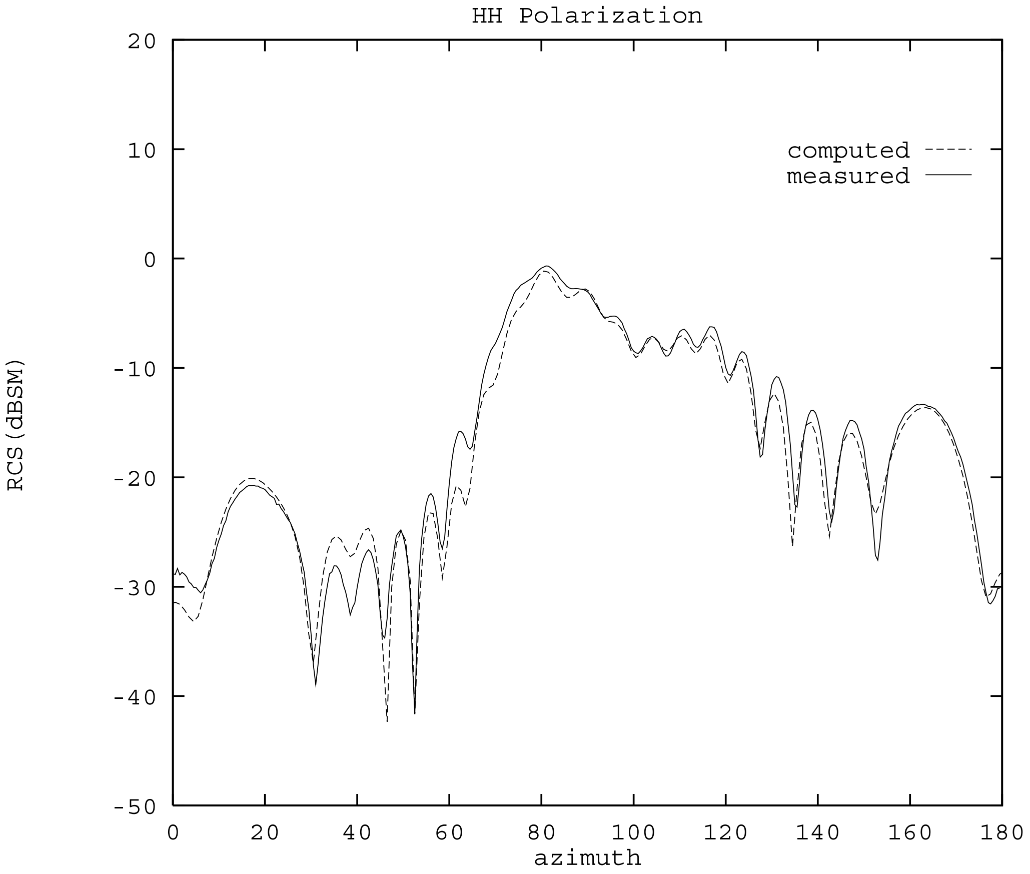


Figure 5.19: The RCS with HH polarization of a double ogive

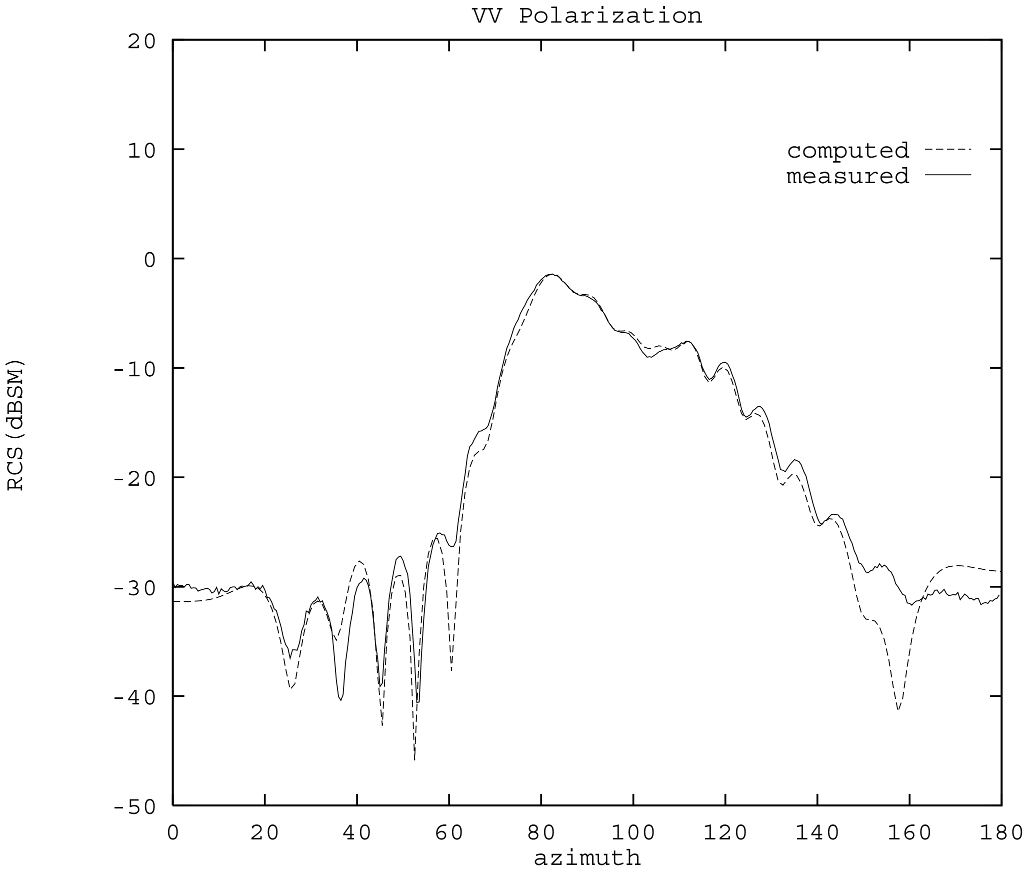


Figure 5.20: The RCS with VV polarization of a double ogive

the joint. The cone-sphere can be described as:

for -23.821 inches $< x < 0$ inches and $-\pi < \phi < \pi$,

$$y = 0.87145(x + 23.821)\cos\phi$$

$$z = 0.87145(x + 23.821)\sin\phi$$

for 0 inch $< x < 3.306$ inches and $-\pi < \phi < \pi$,

$$y = 2.947\sqrt{1 - [\frac{x - 0.359}{2.947}]^2}\cos\phi$$

$$z = 2.947\sqrt{1 - [\frac{x - 0.359}{2.947}]^2}\sin\phi$$

The cone-sphere is 27.127 inches long or about 10.33λ at 4.5 GHz. The parametric patch model of the cone-sphere is shown in Figure 5.21. There are two symmetric planes which are the XY and XZ planes. To reduce the memory and CPU time, we could only grid a quarter of the cone-sphere surface. The quarter of the cone-sphere surface is gridded into a set of 2520 curved triangles with 3884 edges and 1365 nodes. The maximum edge length is $\frac{\lambda}{8}$.

The monostatic RCS for both HH and VV polarizations are plotted in dBSM, as functions of the azimuthal angle in Figures 5.22 and 5.23. The elevation angle is zero. There are 360 RHS vectors for this problem. This problem is run on a 64-node partition of NAS Paragon. The memory required to run this problem on that system is 11 Mbytes per node. The wall clock time is 6357 seconds to complete the run.

The metallic cone-sphere with gap target is the same as the cone-sphere target, except for a gap next to the cone-sphere joint. The gap is $\frac{1}{4}$ inch deep. The difference between the cone-sphere with and without gap is only from $0 < x < 0.25$ inch, where it is defined by

$$y = 2.697\cos\phi$$

$$z = 2.697\sin\phi$$

where $-\pi < \phi < \pi$

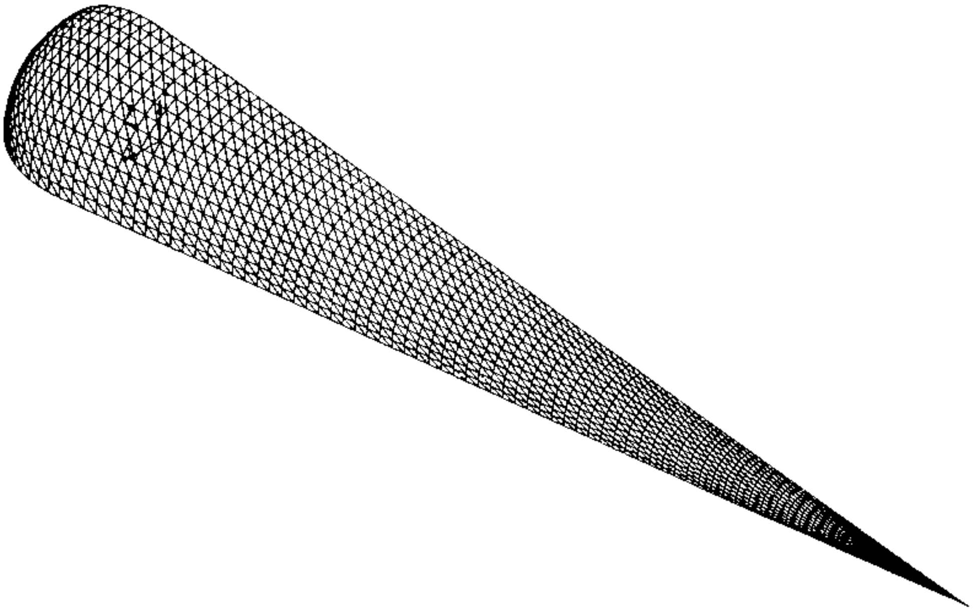


Figure 5.21: The cone-sphere parametric geometry model

The discretized cone-sphere with gap surface is shown in Figure 5.24. As with the cone-sphere without gap, only a quarter of the surface is modeled to compute the radar cross section. The quarter of the surface is modeled by a set of 2576 curved triangles with 3970 edges and 1395 nodes. The maximum edge length is $\frac{\lambda}{8}$.

The monostatic radar cross sections for both HH and VV polarizations are plotted against azimuthal angle in Figures 5.25 and 5.26. The elevation angle is zero. There are 360 RHS vectors for this problem. This problem is run on a 64-node partition of NAS Paragon. The memory required to run this problem in the system is 11 Mbytes per node. The wall clock time is 6639 second to complete the run.

The RCS of a metallic rectangular parallelpiped is computed for the purpose of testing the accuracy of the ParaMoM-MPP code. The rectangular parallelpiped is illuminated by incident waves with the frequency at 5GHz. The electrical size of the rectangular parallelpiped is $1.25\lambda \times 1.25\lambda \times 1.87\lambda$. There are two symmetry planes which are the XY and XZ planes. A quarter of the surface is gridded into a set of

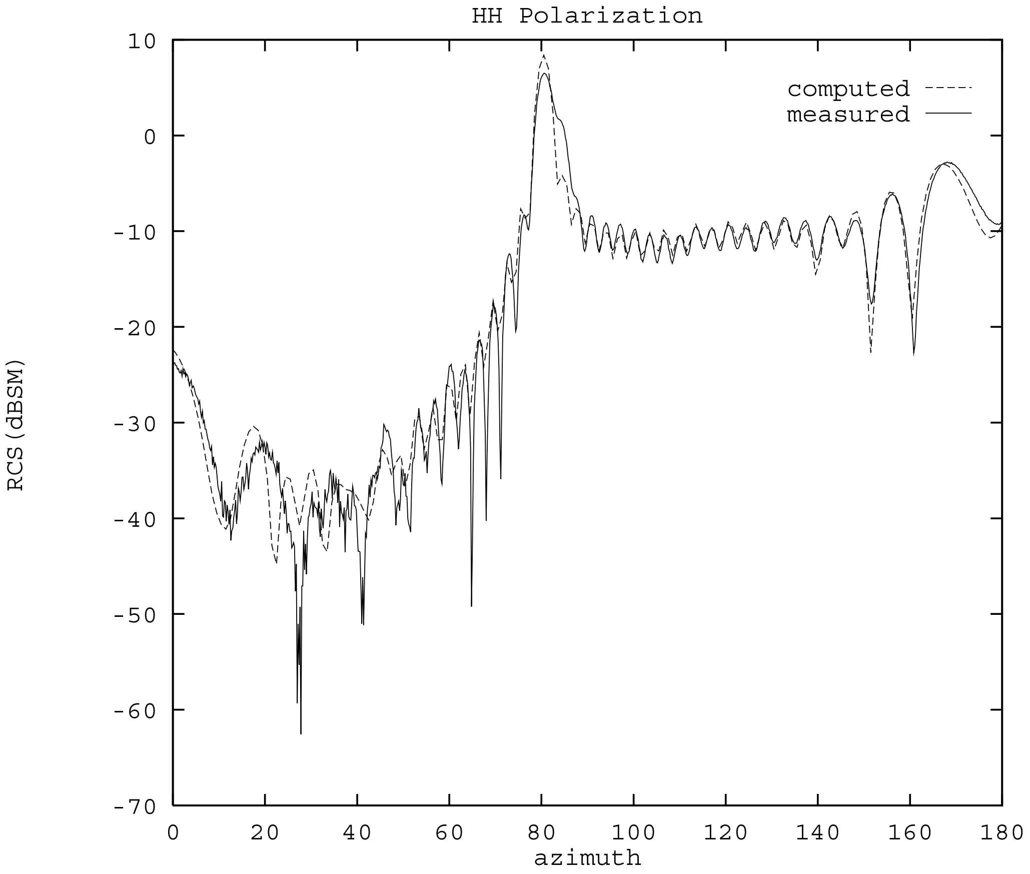


Figure 5.22: The RCS with HH polarization of a cone-sphere

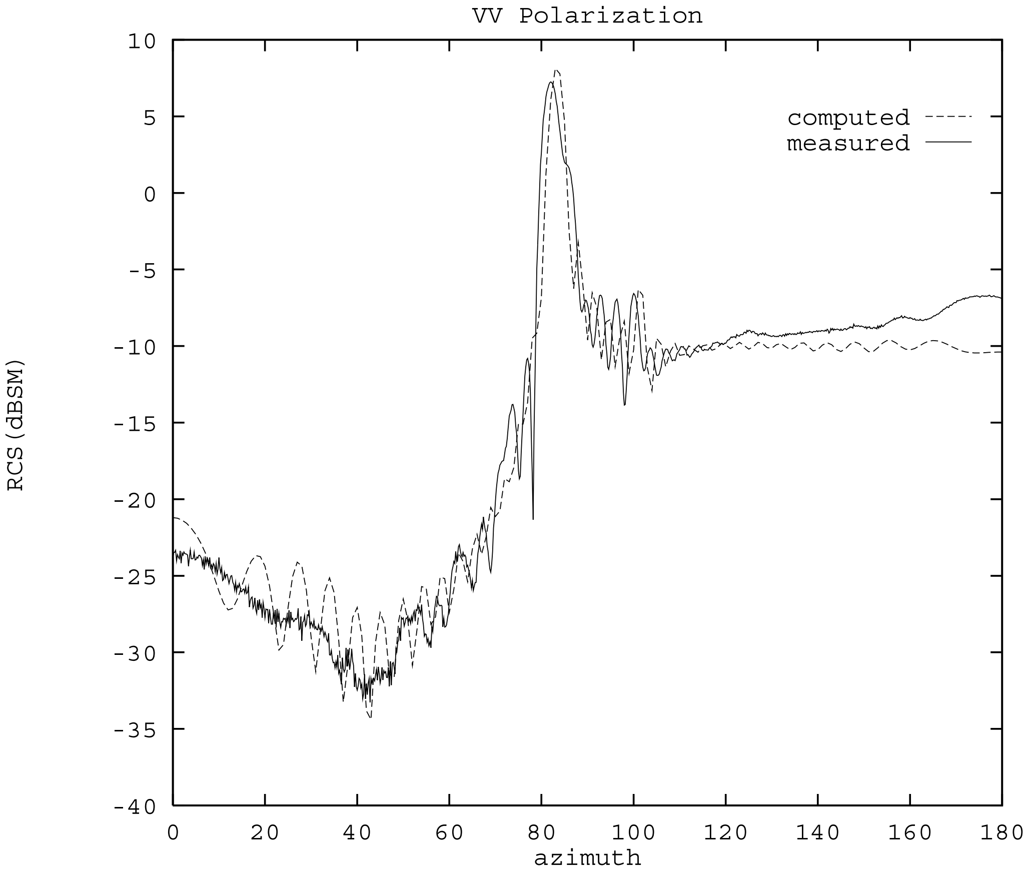


Figure 5.23: The RCS with VV polarization of a cone-sphere

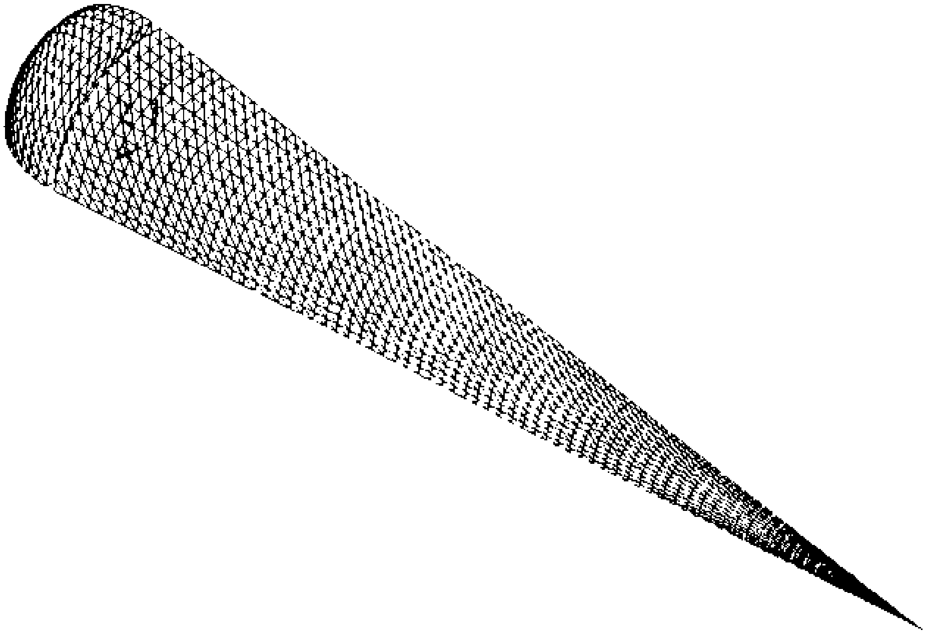


Figure 5.24: The cone-sphere with gap parametric geometry model

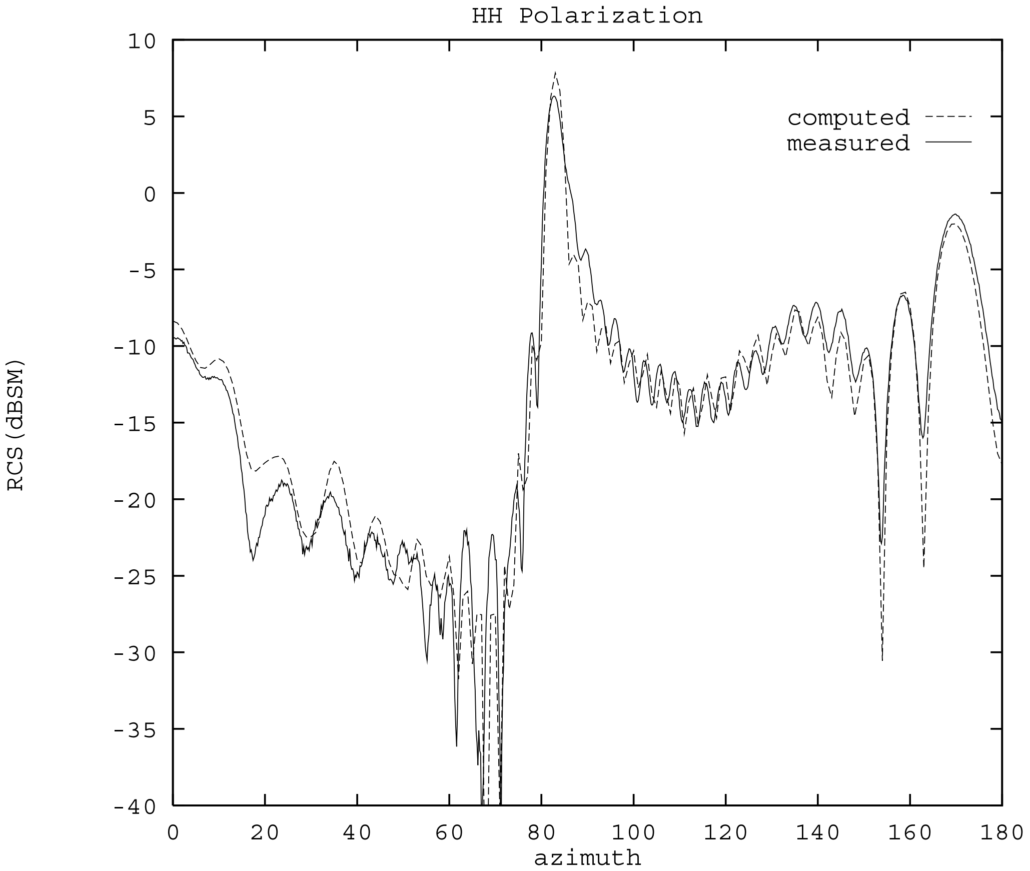


Figure 5.25: The RCS with HH polarization of a cone-sphere with gap

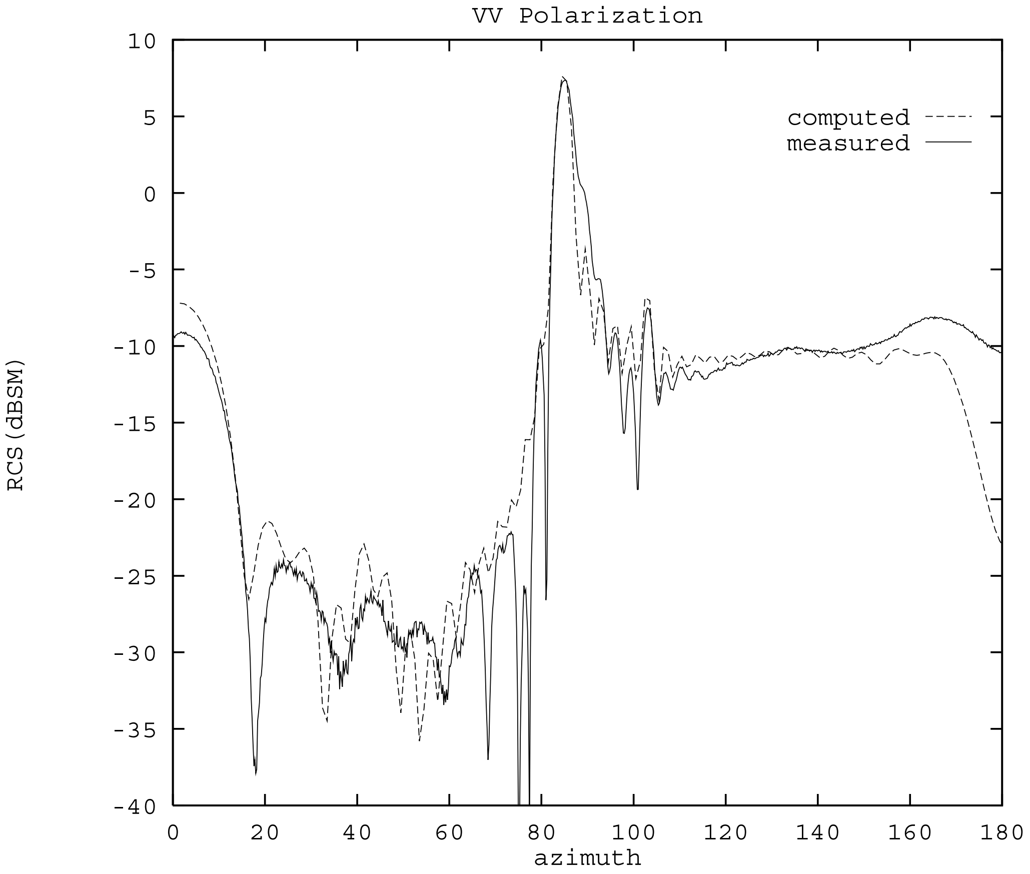


Figure 5.26: The RCS with VV polarization of a cone-sphere with gap

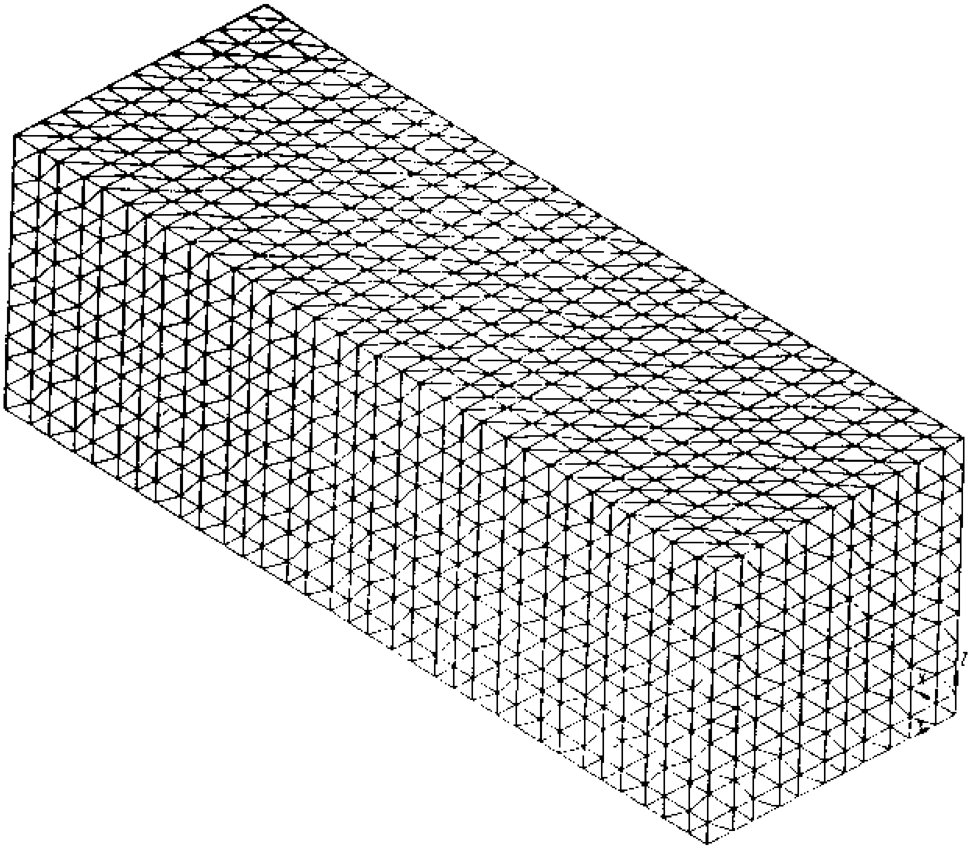


Figure 5.27: The quarter of rectangular parallelpiped parametric geometry model

1693 curved triangles with 2596 edges and 903 nodes. The maximum edge length is $\frac{\lambda}{8}$. The quarter of the rectangular parallelpiped surface which is gridded is shown in Figure 5.27.

The monostatic radar cross sections for both HH and VV polarizations are plotted as functions of the azimuthal angle in Figures 5.28 and 5.29. There are 360 RHS vectors for this problem. This problem is run on a 32-node partition of NAS Paragon. The memory required to run this problem on that system is 9 Mbytes per node. The wall clock time is 3813 seconds to complete the run.

A perfectly conducting sphere with radius 0.8 meter is covered by a dielectric

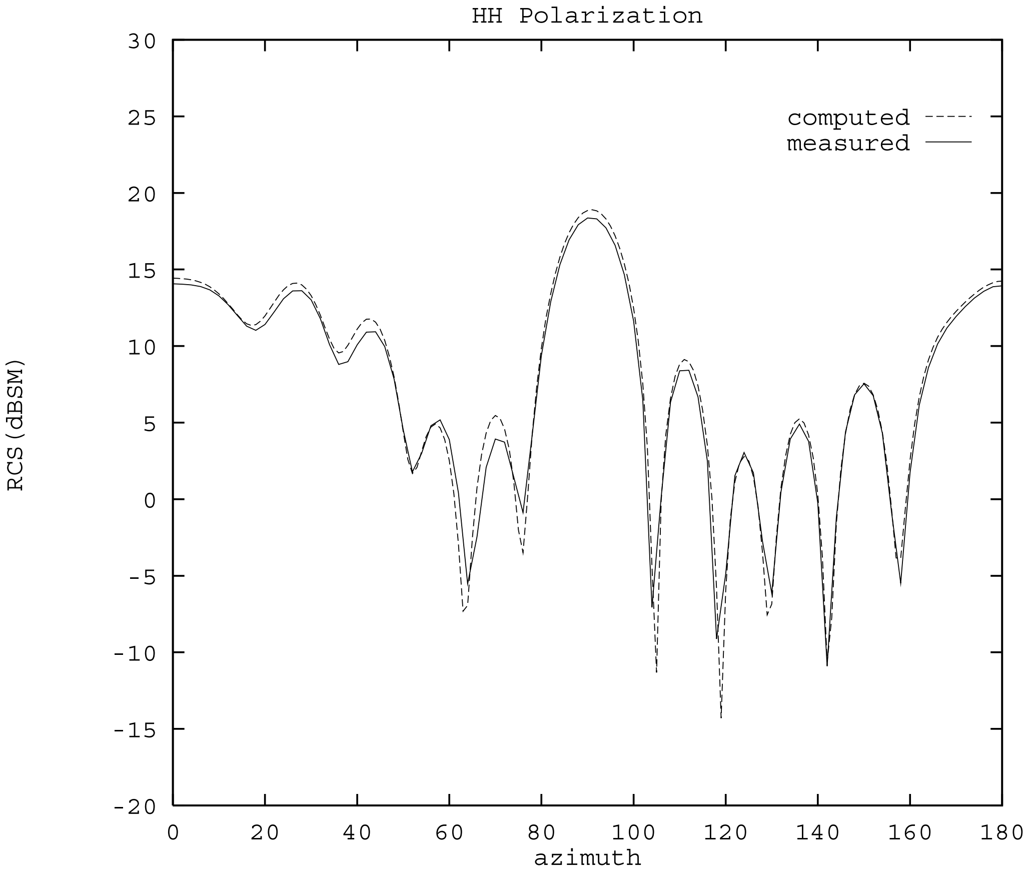


Figure 5.28: The RCS with HH polarization of a rectangular parallelpiped

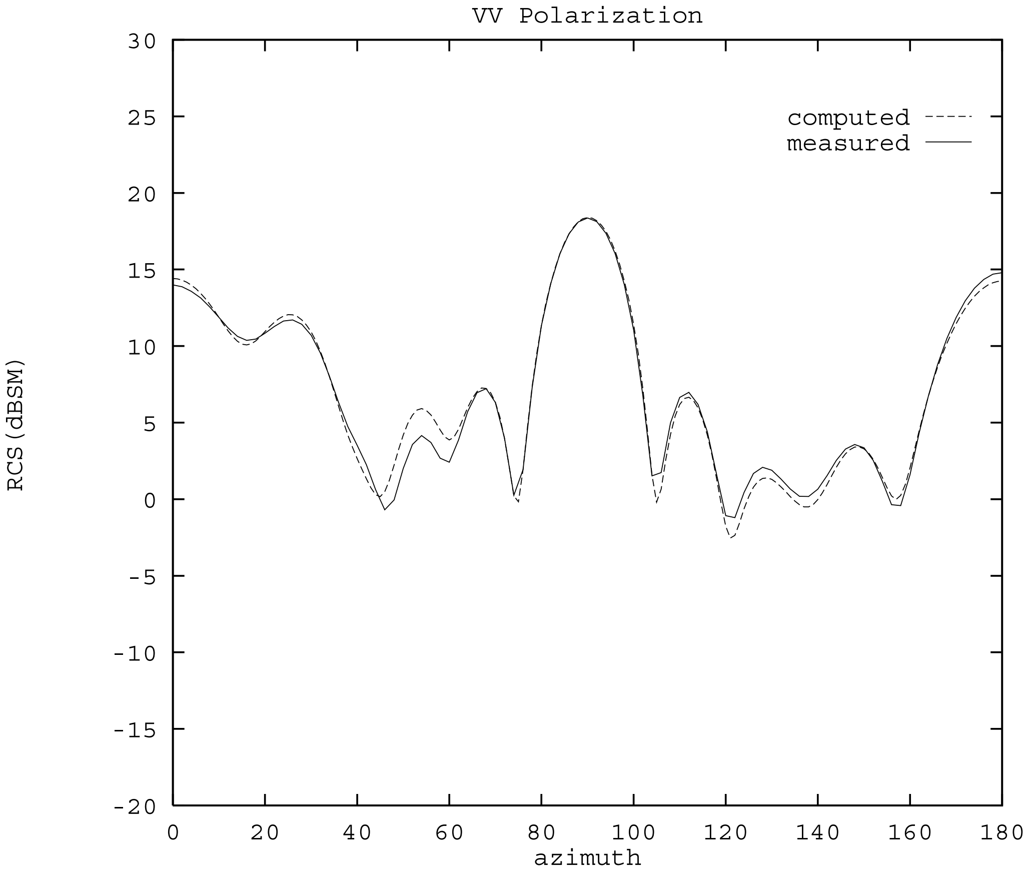


Figure 5.29: The RCS with VV polarization of a rectangular parallelpiped

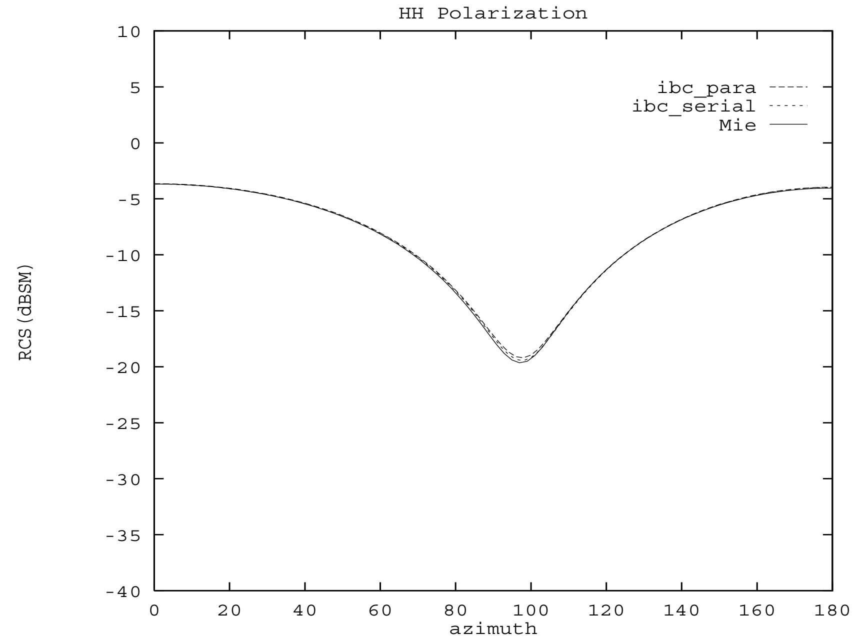


Figure 5.30: The RCS with HH polarization of a coated conducting sphere

material. Where the thickness of the coating is 0.2 meter with the permittivity $10\epsilon_0$ and the permeability μ_0, wh ere ϵ_0 and μ_0 are free space permittivity and permeability, respectively. The incide nt plane wave has a frequency of 47.77 MHz with horizontal polarization. The receiver's polarization is also horizonal. The parametric patch model of the sph ere has 182 triangular patches with 104 nodes and 286 edges.

The bistatic radar cross section for the HH polarizations is ploted in dBSM as function of azimuthal angle in Figure 5.30. There are three results shown in Figure 5.30, a Mie theory result, the modified ParaMoM code result, and the ParaMoM-MPP result. All the results are agreed. On a 32 nodes CM-5, the ParaMoM-MPP code spent 11 seconds to fill the matrix, 1.24 seconds to do the LU/Solve, and the wall clock time 28 seconds. On a SUN 10 workstation, the modified ParaMoM code spent 261 seconds to fill the matrix, 22 seconds to do the LU/Solve, and the wall clock time 287 seconds.

5.3 Discussion of Two Parallel Moment Method Codes

In this section, we discuss the parallel implementation of the PATCH code, which was developed by Johnson, Wilton, and Sharp [15] to see the similarities and the differences between the parallel PATCH code and the ParaMoM-MPP code. The similarities are listed as below:

- Both are general purpose computer codes for electromagnetic scattering from and radiation by objects of arbitrary shape. They are based on the method of moments with a surface integral formulation in the frequency domain. They use surface patches, which conform to surfaces and boundaries of general shape and allow variable patch density over the surface of the object. The basis functions for both codes are defined on pairs of adjacent patches and give a current representation free of line and point charges.

- Both have the following capabilities: computation of scattering from multiple bodies, treatment of multiple intersecting surfaces, use of symmetry planes to reduce the number of unknowns, treatment of surfaces with lumped and distributed impedance loads, and the capability for plane wave or voltage source excitation. The outcomes of these two codes are the currents on the bodies, radiated fields, far fields, and radar cross section.

- Both codes are implemented on coarse-grained MIMD distributed memory systems. Message passing and a direct dense matrix equation solver are the main tools for both codes. The main features of the parallel matrix fill algorithm, are: 1) looping over patches rather than looping over edges; 2) no communication takes place; 3) a good load balance is achieved. LU decomposition with partial pivoting is applied to factorize the moment matrix.

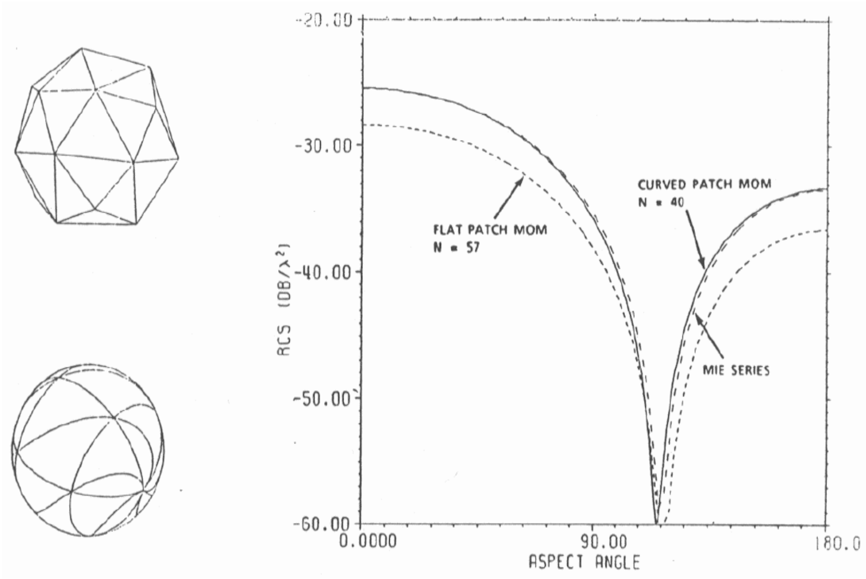
The differences between the PATCH and ParaMoM-MPP code are listed below:

- PATCH uses the popular Rao-Wilton-Glisson [14] technique with flat facets. ParaMoM-MPP uses Wilkes' and Cha's basis function which is defined in terms of any (set of two) arbitrary curvilinear surface coordinates and conforms to the exact surface curvature of the parametric surface at hand. Figure 5.31 shows the comparison of performance on a 0.4λ circumference ($ka = 0.4$) sphere, between the popular RWG technique with flat facets and that using ParaMoM. The improvement in accuracy with similar matrix size is impressive and that leads to reduction in matrix size, for a specified error tolerance. Cha's group provides a bar chart in Figure 5.32 showing the matrix size comparison between the RWG code and ParaMoM 1.0, for several textbook shapes, at a 0.5 dB average error tolerance. A better than two-to-one reduction in matrix size is obtained with the use of curved patches.

- PATCH only uses the electric field integral equation formulation. ParaMoM-MPP has used not only the electric field integral equation formulation but the magnetic field integral equation and the combined field integral equation as well.

- In matrix fill, PATCH has two redundant computations of the magnetic vector potential and electric scalar potential for each source-field patch pair, but ParaMoM-MPP has only an average of one redundant computation of Green's function for each source-field patch pair. PATCH does not precompute the arrays required by the matrix fill. They are computed along with the progress of the fill process. The ParaMoM-MPP matrix fill algorithm is much more efficient than that of PATCH.

- Compared to PATCH, ParaMoM-MPP is implemented on more architectures, so that it has better portability.



Bistatic RCS of Sphere with ka = 0.4, HH polarization

Figure 5.31: Comparison of accuracy on a small sphere

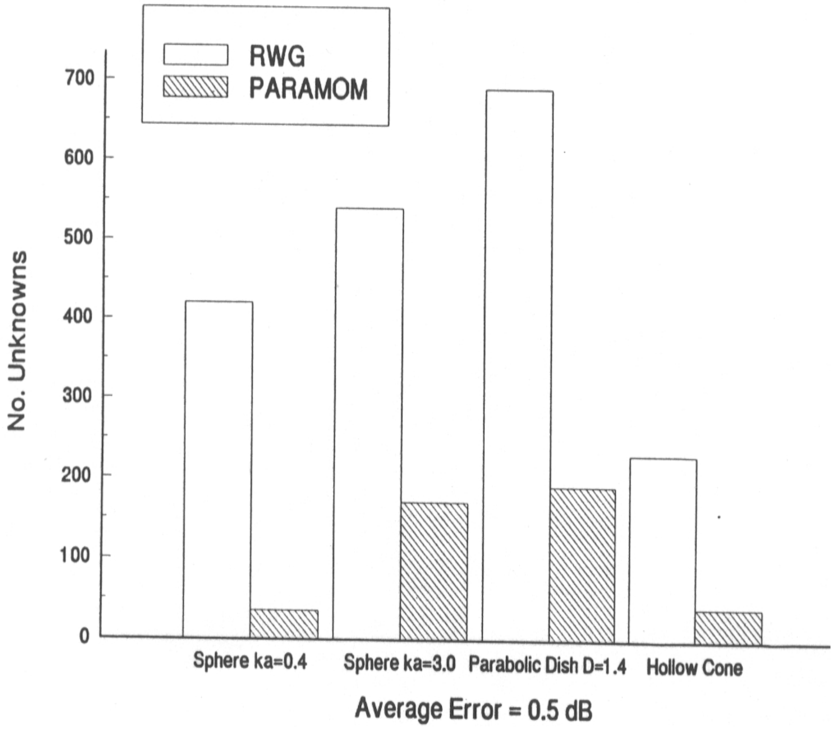


Figure 5.32: Comparison of matrix size for a specified (0.5 dB) error tolerance

Chapter 6

Conclusion

In this thesis, we described Wilkes' and Cha's Parametric patch model and their basis function which is defined on a typical patch. Each patch is a nonplanar surface, that is a triangle in curvilinear coordinate space. The EFIE, MFIE, and CFIE formulations are derived for calculating scattering by 3-D arbitrarily shaped conducting bodies with and without dielectric material coating. A sequential computer program has been written based on the formulation developed in Chapter 3. This code has been merged into the ParaMoM code previously developed by Cha's group at SRC based on the formulation described in Chapter 2. The ParaMoM program calculates the Radar Cross Section (RCS) conducting bodies whose surfaces are arbitrarily shaped with or without dielectric coating.

If the unmodified ParaMoM code was parallelized, the large setup memory required would make it impractical for MIMD distributed computers. ParaMoM was modified to reduce the setup memory to a manageable level for a MIMD distributed computer. The parallel implementation of the ParaMoM code is called ParaMoM-MPP. ParaMoM-MPP is implemented on three different MIMD distributed memory architectures: the Thinking Machine's CM-5, Intel machines, and IBM SP-1. In order to demonstrate the performance of the ParaMoM-MPP code, a series of tests were run to obtain performance data. The performance data listed in the previous chapter shows good performance of all three implementations. To demonstrate the accuracy of the formulation derived in the thesis, ParaMoM-MPP has been used to compute the scattering for the EMCC testing cases. The ParaMoM-MPP RCS results agree well with measured results.

From running EMCC test cases, we observe that exploiting the target's geometric symmetry gives ParaMoM-MPP an advantage in RCS prediction applications. In the EMCC test cases, we take this advantage to reduce problem size by a maximum factor of eight. In a distributed MIMD system, there is only a fixed amount of RAM for each node and no virtual memory in the system. The above-mentioned reduction in setup memory requirement enables the given system to solve a bigger problem.

The parametric patch model requires fewer unknowns than the flat patch model under the same error tolerance. This is demonstrated by Wilkes and Cha in Figures 5.31 and 5.32 in the previous chapter. To demonstrate this using ParaMoM-MPP, we choose the EMCC single ogive target. First, three symmetry planes are used so that only one eighth of the single ogive is modeled by 2088 curved triangular patches. There are eight sub-problems to solve. The average number of equations in each sub-problem is about 3150. Second, the single ogive is modeled without exploiting any symmetry. The dimension of the moment matrix for the whole ogive is chosen to be 7332. The total allocated memory to run this problem is 21 Mbytes per node. The RCS with both HH and VV polarizations with and without symmetry plane are shown in Figures 5.16 and 5.17. This result shows that the number of patches could be much less. This result gives clear vision about the advantage of taking the symmetry into consideration in terms of the memory per node requirement. It is very important for a distributed memory MPP system.

In order to take advantage of state-of-the-art computer technology as much as we could, we designed parallel algorithms for all the components of the ParaMoM code. We implemented a coordinated parallelism algorithm which takes advantage of both the message passing paradigm and data parallel paradigm on Thinking Machine's CM-5. We implemented algorithms which used the NX message passing library [68] or PVM [69] and ScaLAPACK libraries on the Intel computers. The PVM (Parallel virtual machine) and ScaLAPACK libraries are used to accomplish message passing and direct LU matrix solution, respectively. PVM automatically converts data formats among computers from different vendors, making it possible to run an application over a collection of heterogeneous computers. To demonstrate portability, the PVM Intel version of ParaMoM-MPP was ported to the IBM SP-1 with little effort. We also ported the PVM Intel version of ParaMoM-MPP to a FDDI-based Gigswitch-connected DEC alpha farm.

In general, high performance and good speed-up are achieved in these parallel implementations. The performance is scaled well with the problem size and machine size. Highly portable code development across three parallel systems with diverse configurations and hardware capabilities has been achieved by using the BLACS/ScaLAPACK on the distributed system, which is eventually achieved by using PVM message-passing programming interface. This code can further be easily ported to a heterogeneous computing environment. The systems which are used to run ParaMoM-MPP represent most of the state-of-the-art architectures in today's massively parallel processing technology. We may summarize the relative performance of various parallel machines as follows.

The CM5 Implementation

The flexibility given to a user to choose among programming models is an important feature of CM-5, since it lets users choose the technique that is best, not only for their application, but for each part of their application. The ParaMoM-MPP implementation is a good example. The matrix fill portion of ParaMoM-MPP is optimized by slab data decomposition. Utilizing the edge numbering scheme to loop over patches rather than basis functions minimizes the redundant computation to one Green's function for each source-field patch pair. The matrix fill algorithm has no internode communication and very good load balancing. The fill algorithm is scalable as discussed in Chapter 5. The data parallel program for factoring and solving the matrix equation is written in CM Fortran interfaced with the CMSSL (Connection Machine Scalable Scientific Library). The factor/solve implementation is optimized because: 1). It has good load balance (block cyclic data decomposition); 2). CM Fortran utilizes the full power of four vector units within each CM-5 node; the peak performance for the CM-5 scalar nodes is 5 Mflops but using the 4 vector units the peak performance increases to 128 Mflops; 3) The CMSSL Library is a well-written and well-respected piece of software. The performance of factor/solve is the best on the CM-5 among the three platforms. The I/O performance is excellent to both SDA and Data Vault systems. The CM-5 implementation can be ported to all CM-5 partitions without change.

The drawback of the CM-5 implementation is that the matrix fill is relatively poor. Fortran 77 can not utilize the vector units in the CM-5 nodes, therefore the performance is expected to be poor since CM-5 SPARC2 node has only 5 Mflops peak performance. Although it is possible to utilize the vector units by implementing in CM DPEAC (a CM assemble statements), this is not expected to improve the fill performance much. Because of its nature, the problem is very difficult to vectorize and pipeline efficiently.

The Intel Paragon Implementation

Both PVM and NX (the Intel message passing library) are implemented on various Intel machines, such as the iPSC/860, the Touchstone Delta, and the Paragon. These implementations are very flexible. Users can configure virtually the architecture. The BLACS gives good communication libraries and the ScaLAPACK provides scalable factor/solve functions. Combining the above with powerful Paragon nodes, the ParaMoM-MPP code gives good performance for both matrix fill and factor/solve. For matrix fill, a slab data decomposition not only minimizes the amount of redundant computation of Green's function for each source-field patch as with the CM-5 implementation but achieves good load balancing as well. However, the block scattering data decomposition gives better load balancing for LU with partial pivoting. Since our Intel implementation is in one piece, it is impossible to achieve load balancing for both matrix fill and factor/solve without data reshuffling. We not only implemented ParaMoM-MPP to optimize matrix fill (or slab data decomposition) but also implemented it to use I/O with a parallel file system as a buffer to reshuffle data in each node. We observe that the same total wall clock time was used to run the code for the same application. The number of factor/solve Mflops achieved after data reshuffling is almost double the number without reshuffling. We note that the I/O performance is relatively poor on the NAS Paragon, but that may be in part due to the relatively low number of I/O nodes in the NAS Paragon configuration. Portability is achieved by the PVM library, which is available to many systems.

The IBM SP-1 Implementation

The IBM SP-1 has the best scalar node performance among these three platforms, and as a result it gives the best matrix fill performance per node for the platforms tested. The SP-1 system is still a new machine and in the test stage; the system software is not yet mature. It is believed that the IBM SP series can be very powerful once the software is further developed.

The DEC Alpha Farm Implementation

For testing purposes, we also implemented ParaMoM-MPP on a FDDI-based Gigswitch-connected DEC Alpha workstation cluster. A distributed system provides higher performance and larger memory per node than MPP systems. This makes the high performance distributed computing (HPDC) approach to electromagnetic application feasible and attractive to solve modest sized problems. However, the communication bandwidth on a typical network does not scale with increased numbers of nodes. Therefore the workstation cluster is probably not a practical solution for solution of large problems since the LU factor performance will be limited by the communication bottleneck.

Our contributions are summarized below:

- Using Wilkes' and Cha's [2] parametric surface patch model, an electromagnetic scattering formulation for three-dimensional arbitrarily shaped conducting bodes with or without dielectric material coating was derived.

- The ParaMoM code was parallelized to run on three massively parallel architectures.
- The ParaMoM code was extended to treat dielectric coating.

Desirable features of the ParaMoM-MPP implementation are listed below:

Accuracy

The ParaMoM-MPP code is functionally identical to the mature, stable, and well-validated ParaMoM code. Results that validate ParaMoM-MPP are presented in Chapter 5.

Efficient Parallel Implementation

The overall performance of the parallel code is very good on the target architectures. The solution time for relatively large problems is two orders of magnitude shorter on typical MPPs than on the fastest state-of-the-art single-processor workstation.

Portability

The parallel ParaMoM code was implemented on multiple MPP architectures. Differences between the implementations are superficial.

Scalability

Empirical and analytical results demonstrate that excellent efficiency may be obtained on very large problems and large MPP configurations.

Flexibility to Incorporate Future Developments

The matrix filling algorithm utilizes a flexible data partitioning scheme that can easily be interfaced to a variety of matrix solvers. The out-of-core fill algorithm gives the ability to fill the matrix either in-core or out-of-core. This algorithm has been incorporated and tested on the CM-5 code.

Delivery of Useful MPP Codes on Multiple Platforms

The ParaMoM-MPP code has been fully tested on three basic configurations: Intel, CM-5, and IBM SP-1.

Future Work

Our work demonstrates that parallel computing and advanced solution techniques are equally important to successfully achieve full-scale aircraft RCS prediction. Our work is a very successful example of combining state-of-the-art massively parallel processing technology with state-of-the-art computational electromagnetic techniques. This approach will lead the way to achieve RCS prediction of a full-scale aircraft. To illustrate this, we examine the case of the VFY218. The full-scale VFY218 has a total surface area, including engine ducts and exhaust, of approximately 200 m^2. An exact method of decomposition can be used to isolate the interior solutions (ducts, exhaust) from the exterior solution (outside the aircraft). This will reduce the surface area of the problem to 160 m^2. One symmetry plane can be used to halve the 160 m^2 to 80 m^2. We now have 900 λ^2 at 1 GHz and 225 λ^2 at 500 MHz. At 500 MHz, a 10 points per wavelength sampling rate creates a moment matrix of dimension 45,000. An out-of-core implementation on a state-of-the-art IBM 512 SP-2 system–which will be installed at the Cornell Theory Center later in 1994–can handle this case. SRC is developing an advanced method capable of accurate prediction at a sampling rate around 6 points per wavelength. This method will be in ParaMoM 2.0. At 1 GHz, this sampling rate leads to a matrix order of 65,000 for a VFY218. When ParaMoM 2.0 is developed, the parallel implementation will achieve the RCS prediction of a full-scale VFY218. Practical RCS prediction using numerical methods is realistic with massively parallel processing technology.

It is possible to combining the MoM with other numerical techiques to develop a new algorithm. The new algorithm may reduce the computation from $O(N^3)$ to

$O(N \log N)$.

Appendix A

A Normalized Local Area Coordinate System

The integral considered in Chapters 2 and 3 is

$$I(x,y) = \int_T f(x,y,u,v)du\,dv \tag{A.1}$$

where T is a flat triangle in the parametric (u,v) space with an area of A. The most convenient way to evaluate the integral in the parametric (u,v) space is to transform coordinates to a local system of area coordinates (Chapter 8 [49]) within triangle T. The triangle T is divided into three regions of area A_1, A_2, and A_3 which are constrained to satisfy $A_1 + A_2 + A_3 = A$ shown in Figure (A.1). We define the nomalized area coordinates as

$$\begin{aligned} \xi &= \frac{A_1}{A} \\ \zeta &= \frac{A_2}{A} \\ \gamma &= \frac{A_3}{A} \end{aligned} \tag{A.2}$$

which, because of the area constraint, must satisfy

$$\xi + \zeta + \gamma = 1 \tag{A.3}$$

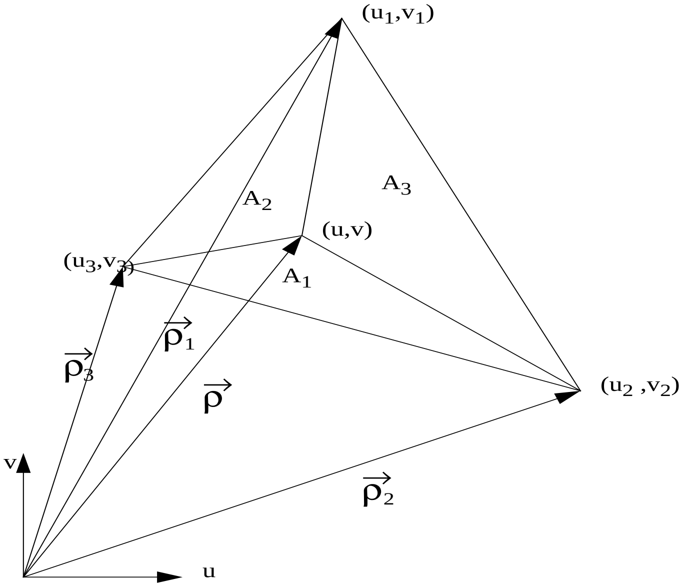


Figure A.1: Definitions of areas used in defining area coordinates.

Only two of the normalized area coordinates can be considered as independent variables. Each point (u, v) within triangle T can be represented in terms of ξ, ζ, and γ and its vertices as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \xi \begin{pmatrix} u_1 \\ v_1 \end{pmatrix} + \zeta \begin{pmatrix} u_2 \\ v_2 \end{pmatrix} + \gamma \begin{pmatrix} u_3 \\ v_3 \end{pmatrix} \tag{A.4}$$

where $\begin{pmatrix} u \\ v \end{pmatrix}$, $\begin{pmatrix} u_1 \\ v_1 \end{pmatrix}$, $\begin{pmatrix} u_2 \\ v_2 \end{pmatrix}$, and $\begin{pmatrix} u_3 \\ v_3 \end{pmatrix}$ are points in vector form in the (u, v) space. They are denoted by $\vec{\rho}$, $\vec{\rho}_1$, $\vec{\rho}_2$, and $\vec{\rho}_3$ as shown in Figure A.1. Substituting these into (A.4) and using (A.3) to eliminate γ, we obtain

$$\vec{\rho} = \xi\vec{\rho}_1 + \zeta\vec{\rho}_2 + (1 - \xi - \zeta)\vec{\rho}_3 \tag{A.5}$$

Differentiating $\vec{\rho}$ in (A.5) with respect to ξ and ζ, respectively, gives

$$\frac{\partial\vec{\rho}}{\partial\xi} = \vec{\rho}_1 - \vec{\rho}_3 = \mid \vec{\rho}_1 - \vec{\rho}_3 \mid \hat{\xi}$$

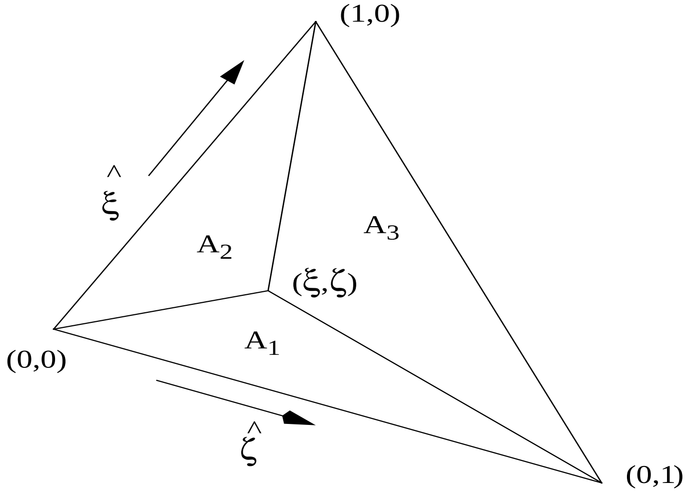


Figure A.2: The local area system of coordinates.

$$\frac{\partial\vec{\rho}}{\partial\zeta} = \vec{\rho}_2 - \vec{\rho}_3 = \mid \vec{\rho}_2 - \vec{\rho}_3 \mid \hat{\zeta} \tag{A.6}$$

where $\hat{\xi}$ and $\hat{\zeta}$ are the unit vectors of ξ and ζ, respectively, shown in Figure A.2.

The Jacobian for the local area coordinates is given by

$$\mathsf{J} = \mid \frac{\partial\vec{\rho}}{\partial\xi} \times \frac{\partial\vec{\rho}}{\partial\zeta} \mid = \mid \vec{\rho}_1 - \vec{\rho}_3 \mid \mid \vec{\rho}_2 - \vec{\rho}_3 \mid \mid \hat{\xi} \times \hat{\zeta} \mid = 2A \tag{A.7}$$

so that, it can easily be shown that the surface integral over T in (A.1) transforms as follows:

$$I(x,y) = \int_T f(x,y,u,v)ds = 2A\int_0^1\int_0^{1-\xi} f(x,y,u(\xi,\zeta),v(\xi,\zeta))d\zeta d\xi. \tag{A.8}$$

where $u(\xi,\zeta)$ and $v(\xi,\zeta)$, given by (A.4), are linear functions of ξ and ζ.

Appendix B

Definition of Scattering Angles

The azimuth angle and the elevation angle used in the RCS plots are defined in Figure B.1, where AZ denotes the azimuth angle, El denotes the elevation angle, and RLOS denotes the line of sight determined by AZ and EL.

In Figure B.2, T denotes the transmitter; R is the receiver; β denotes the fixed bistatic angle; β_1 is the initial bistatic angle; β_2 is the final bistatic angle; AZ is the initial azimuth angle; and EQLOS is the fixed equivalent line of sight.

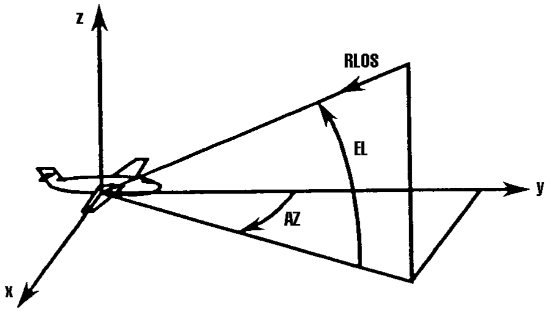
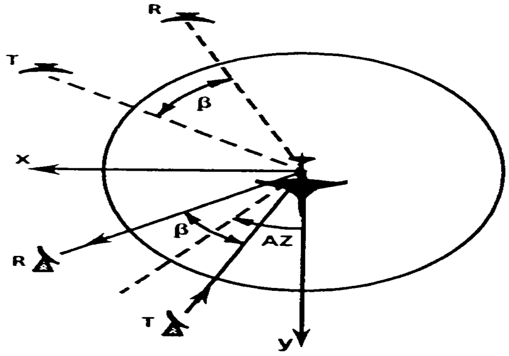
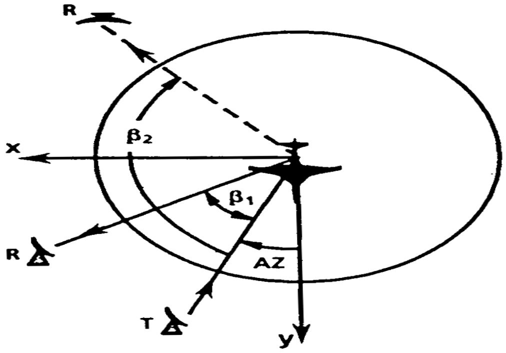


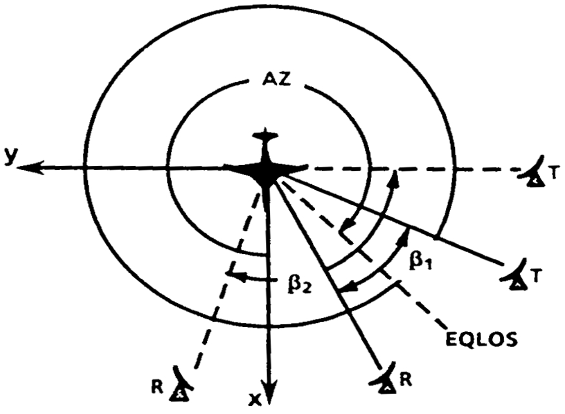
Figure B.1: Definition of azimuth, elevation, and RLOS angles



(a). Monostatic or fixed bistatic angle



(b). Bistatic angle with fixed transmitter



(c). Bistatic angle with fixed equivalent line of sight

Figure B.2: Definition of monostatic and bistatic angles for different RCS modes

Appendix C

Overview of Three MPP Architectures

This appendix is a simple overview of three coarse-grained parallel architectures. They are the Thinking Machine Corporation CM-5, Intel Supercomputing Division's Paragon, and the IBM SP-1.

C.1 The CM-5 System Overview

This section gives a brief overview of the CM-5 system. A CM-5 system contains four major parts:

- processing nodes
- control processors
- I/O nodes
- networks

The CM-5 is available in configurations of 32 to 1024 processing nodes, each node being a RISC microprocessor with optional attached vector units. Current implementations use a SPARC microprocessor with 32M to 128M bytes of memory and four optional vector units. Each processing node operates at 33 MHz and is rated at 22 Mips and 5 MFlops. When equipped with vector units, each node is rated at 128 Mips (peak) and 128 MFlops (peak).

The processing nodes can be divided into several partitions and each partition contains a control processor. The control processors run a UNIX-based operating system. They can download programs into the processing nodes in their partition, control the program execution on the processing nodes, and handle sequential I/O for processing nodes. The control processors also participate in computation when the program is executed in the host-node programming mode.

The I/O nodes, which also connect to the network, handle high performance parallel I/O for processing nodes. Besides ordinary NFS file systems, the I/O nodes support all HIPPI (High Performance Parallel Interface), SDA (Scalable Disk Array) and VME interfaces, thus allowing connections to a wide range of computers and I/O devices. A CMIO interface supports mass storage devices such as the Data Vault. The peak throughput is from 20 to 200 Mbyte/sec.

The CM-5 internal networks (Figure C.1) include two components, a data network and a control network. The data network contains two channels, a left data network and a right data network. The control network contains three different networks, a broadcast network, a combine network, and a global network. The CM-5 has a separate diagnostics network, which is visible only to the system administrator, to detect and isolate errors throughout the system.

The data network, which is a 32-bit-wide data path, provides high performance data communications among all system components. The network has a peak bandwidth of 5M bytes/sec for node-to-node communication. However, if the destination is within the same cluster of 4 or 16, it can give a peak bandwidth of 20M bytes/sec and 10M bytes/sec, respectively. The topology of the CM-5 data network is a fat tree and the communication mechanism is worm-hole. Figure C.2 shows the data network with 16 nodes. The fat tree topology allows more than one path to be used for data transmission. The data path will not be blocked unless all links are occupied. Data

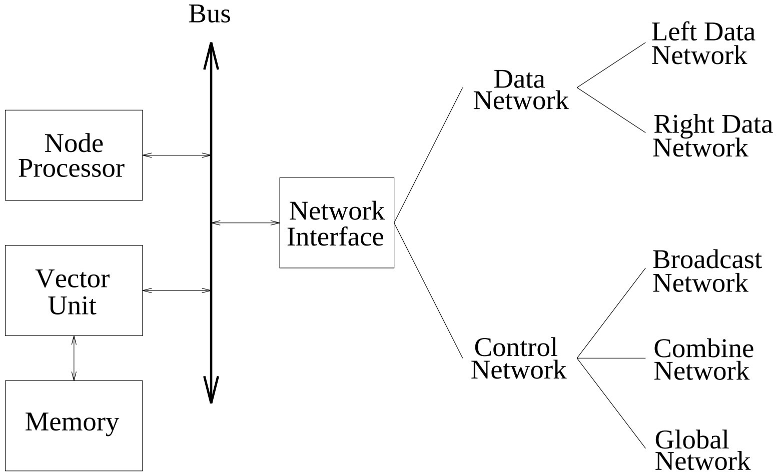


Figure C.1: CM-5 network interface.

packets can choose any link that connects to its destination during data transmission, which reduces the possibility of link contentions with other data packets. Each packet is a size of 5 words long, which means that a large message must be divided into many packets for transmission. Data packets from other nodes can be sent via the same link interactively. Data packets from different nodes can share the same link and thus will not be blocked by the large message. Link contention will not be a problem on CM-5 unless the amount of data has exceeded the capacity of the data network. Since data packets may be sent via different links, the receiving order may be different from the sending order. A sequential mechanism is needed when transmitting data larger than 5 words.

The control network handles operations requiring the cooperation of many or all processors. The broadcast network broadcasts messages to all nodes. The combine network supports parallel prefix, reduction operations, and network-done tests; it accelerates the cooperative mathematical and logic operations. The global network handles the synchronization for the nodes of CM-5; it supports both synchronous and asynchronous interfaces to perform synchronization among processing nodes.

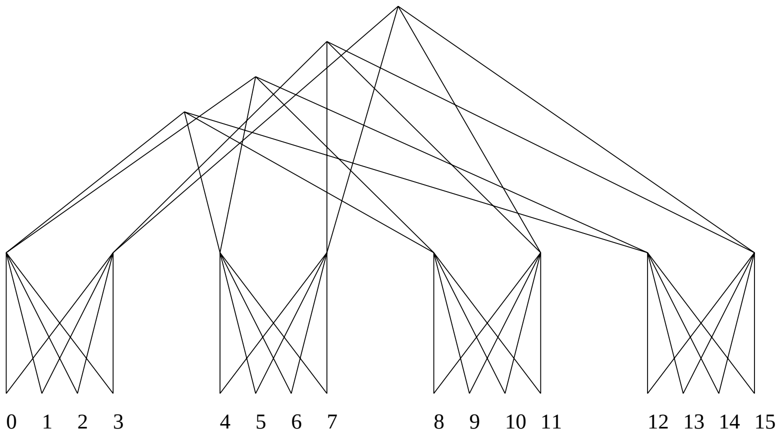


Figure C.2: CM-5 data network with 16 nodes.

CMMD, which provides a full range of message-passing facilities, is the standard communication library of CM-5. CMMD provides both host-node and hostless programming modes. The front-end control processor acts as the host when the host-node programming mode is used.

CMMD provides functions for

- “Cooperative” concurrent message passing, in which synchronization occurs only between matched sending and receiving nodes and only during the act of communication.
- Asynchronous, interrupt-driven, message passing.
- Global message passing, which involves and synchronizes all nodes within the partition.

I/O operations can be done by each node independently or by all nodes working globally.

The CM-5 also provides active message CMAML, which is an asynchronous communication mechanism with the following underlying scheme: Each message header contains the address of a user-level handler that is executed at the receiving node upon message arrival with the message body as argument(s).

C.2 The CM-5 at NAS

This section describes the hardware and software configuration of the Thinking Machine Corporation (TMC) Connection Machine 5 (CM-5) system at NAS.

The CM-5 Hardware The CM-5 at NAS is a distributed-memory multiprocessor computer. It has the following features.

- 128 compute nodes; each node consists of a SPARC processor and four vector processors and can give a theoretical peak performance of 128 MFLOPS. The theoretical total peak performance of the 128 node system is about 16 GFLOPS.
- Each node has 32 MegaBytes (MB) of local memory for a total of 4 GigaBytes (GB) of memory.
- The nodes can be organized into a single partition or multiple partitions. A partition can be as small as 32 nodes or as large as the entire computer.
- Each partition has a control processor called the Partition Manager (PM) that governs the allocation of parallel resources, performs the inherently sequential part of data-parallel programs, and acts as an interface to external networks. Each partition manager is a SPARC processor without vector units.
- The CM-5 is connected to the outside world by ethernet and HiPPI. HiPPI was recently installed (5/94) and is currently undergoing testing. It should be available soon for fast file transfer to long term storage.

The CM-5 Internal Communication Networks The CM-5 has two internal networks that support interprocessor communication. They are:

- **Control Network:** The control network provides tightly coupled communications services. It is optimized for low latency and supports synchronization and broadcast operations. It has latency of 2-5 microseconds.

- **Data Network:** The data network provides loosely coupled communications services. It is optimized for high bandwidth and supports point-to-point communication. The data network interconnect is a fat tree. It has a peak bandwidth of 20 MB/s between any two nodes (one direction).

Measured performance or point-to-point communication (which uses the data network) has about 80 microseconds latency (startup time) and 8.7 MB/sec bandwidth.

Global operations generally use the control network and are extremely fast, as long as they do not involve large arrays. Global synchronizations and integer reductions take about 5 microseconds. Global floating point reductions take about 25 microseconds. A broadcast of a single word to all nodes takes about 8 microseconds. An all-to-all gather (CMMD_concat_with_nodes()) takes 42 microseconds for a single word on each node. The broadcast and concatenation operations take longer for longer messages, reaching an aggregate bandwidth of 25 MB/sec and 34 MB/sec on 32 nodes, respectively. The preceding numbers were obtained with CMMD 3.1 final.

The CM-5 I/O System

- The CM-5 at NAS is a Scalable Disk Array (SDA) with 48 GB of fast parallel disk space.

  The SDA is a RAID 3 device. Multiple (up to 48) small disks operate together as a unit and files are distributed (striped) over the small disks. There is one parity drive. From the software point of view, the SDA looks like an ordinary UNIX device with enhancements for large ($>$ 4 GB) files and fast parallel I/O.

- The CM-5 has three HIPPI (High Performance Parallel Interface) interfaces to link the CM-5 and its storage devices to other supercomputer systems such as the CRAY Y-MP C90. The HIPPI interfaces will not be operational until mid 1994.

- The NAS CM-5 does not have Datavault.

The CM-5 Software and Programming Environment

- CMOST: The CM-5 operating system, CMOST, is an enhanced version of the UNIX operating system. It is binary-compatible with SunOS 4.1 (for scalar programs). The enhancements provide support for parallel programming. The partition manager runs a full version of CMOST; the processing nodes run a CMOST microkernel.

- Prism: The Prism programming environment is an integrated-graphical environment for program editing, debugging, data visualization and performance analysis. Prism operates on workstations running the X window system. A commands-only version is also available for users without access to X window.

- CM/AVS: CM/AVS is a graphical user interface that adopts and extends the Application Visualization System (AVS) to the CM-5. A user can use CM/AVS to build distributed-visualization applications that involve operations such as filtering, graphing, volume rendering, and animation. CM/AVS is not generally available. If you need to use it, contact NAS User Services.

Data Parallel Languages

- CM Fortran: CM Fortran is a standard Fortran compiler supplemented with the array-processing extensions of Fortran 90.

- C*: A version of standard C with extensions to support data parallel programming.

- *Lisp: The data parallel version of Lisp. Currently only the *Lisp interpreter is available for the CM-5.

- The CMAX translator is available to assist in converting serial Fortran 77 source code into data parallel CM Fortran source code.

Data Parallel Processing: From the software perspective, an array object refers to all the data elements of the array simultaneously. From the hardware perspective, the separate operations on the array's elements are all performed simultaneously.

CM Fortran: The CM Fortran language is an implementation of Fortran 77 supplemented with array-processing extensions from the ANSI and ISO (draft) standard Fortran 90. These array-processing features map naturally onto the data parallel architecture of the Connection Machine (CM) system, which is designed for computations on large data sets. CM Fortran thus combines:

- The familiarity of Fortran 77, still the language of choice for scientific computing;
- The expressive power of Fortran 90, which offers a rich selection of operations and intrinsic functions for manipulating arrays;
- The computational power of the CM system, which brings thousands of processors to bear on large arrays, processing all the elements in unison.

In Fortran 77, operations are defined only on individual scalars. Operating on an array requires stepping through its elements, explicitly performing the operation on each one. With Fortran 90 constructions, it is not necessary to reference array elements separately by means of subscripts, and it is not necessary to write DO loops or other such control constructs to have the operation repeated for each element. It is sufficient simply to name the array as an operand or argument.

CM Fortran implements the array-processing features of Fortran 90. Features of Fortran 90 in the major areas other than array processing — such as pointers, structures, modules, and precision control — are not part of CM Fortran.

CM Fortran Utility Library: The Utility Library provides convenient access from CM Fortran to the capabilities of lower-level CM software. The purpose is typically to achieve functionality or performance beyond what is currently available from the compiler. CM Fortran programmers can use Utility Library procedures in situations where one is normally tempted to make explicit calls to lower-level software. The CM Utility Library provides parallel I/O which is compatible with the parallel I/O of the

CMMD library and the CM Fortran timer.

Software Libraries

- CMSSL Scientific and Mathematical Library: The CM Scientific Software Library (CMSSL) provides routines for performing data parallel versions such as numerical linear algebra, FFTs, ordinary differential equations, optimization, random number generation, and statistical analysis. The library also provides optimized communication functions important to structured- and unstructured-grid computations for the numerical solution of partial differential equations and optimization problems such as: communication compiling, partitioning, polyshifting, gathering and scattering, and all-to-all broadcasting and reduction.
- CMMD Communication Library: The Connection Machine communication library, CMMD, supports node level programming (message passing) by providing routines for fast and efficient communication between processing nodes.
- CMX11 Visualization Library: This library provides routines to transfer parallel data between the CM-5 and any workstation running X11. The library can be called from CM Fortran and C*.

C.3 The Intel Paragon System at NAS

This section describes the hardware and software configuration of the Intel Paragon at NAS.

The Paragon is a 227-node i860/XP-based distributed memory multiprocessor with a mesh interconnection network. It supports the message-passing programming model with the NX communication library. It will eventually replace the NAS iPSC/860 (hypercube).

Hardware: The NAS Paragon (grace) is a distributed-memory multiprocessor. The hardware configuration is as follows:

- Nodes: 208 computing nodes have 32 MB memory per node used for running parallel applications. 6 Service nodes have 32 MB memory per node for users running interactive jobs. 8 I/O nodes have 32 MB memory per node connected to RAID disks. 2 ethernet nodes have 16 MB memory per node connected to an external ethernet network. 2 HiPPI nodes have 32 MB memory per node. 1 FDDI node has 16 MB memory connected to an external FDDI network.
- Networks: FDDI is currently the default, Ethernet is accessible as grace-ec, and HIPPI is not yet available.
- Disk: 9 RAIDs are at 4.8 MB/RAID for 43 Gb total unformatted space. The allocation is as follows:
  - System and Swap: 14 GB.
  - Users: Home Directories 4 GB and PFS 22 GB.
- Descriptions:
  - Each node has two i860/XP RISC processors. Currently only one processor can be used. Peak performance of the i860/XP is 75 MFLOPS (double precision), giving a theoretical peak performance of almost 16 GFLOPS for the entire computer. Typical performance for real CFD codes is between 5 and 10 MFLOPS per node.
  - Each computing node has 32 MB of local memory. Of this, approximately 22 MB is available to user programs.
  - The Parallel File System (PFS) provides scratch space for fast parallel I/O. It provides functionality equivalent to the hypercube CFS.
  - 6 nodes comprise the service partition and provide an interface to the outside world, serving as a “front end” to the computer. These service

  nodes run interactive jobs, such as shells and editors. They appear as one computer running Unix. Eventually (when the software is fixed) processes will be migrated between processors for load balancing.

  - 208 nodes comprise the compute partition, and run parallel applications.
  - The nodes are connected by a network with the topology of a two-dimensional mesh. Messages are packetized and routed using dimension-order routing. Messages are automatically routed through intervening nodes without interrupting the processor.

    Performance: Under the current version of the operating system the latency (startup time) for point-to-point messages is about 120 microseconds and the bandwidth is about 35 MB/sec. These are roughly independent of the source and destination nodes. Global operations are implemented using point-to-point communication and have correspondingly high latencies which vary with partition size.

    The capability of the hardware is about 5 times better than these numbers and we expect to see them improve with subsequent operating system releases and possible hardware upgrades.

  - The Paragon is known as "grace.nas.nasa.gov" and can be accessed via telnet or rlogin.

Software

- OSF/1: The Paragon runs OSF/1 from the Open Software Foundation. OSF/1 is an emerging Unix standard which contains features from both System V and Berkeley Unix.

  OSF/1 runs on top of the Mach 3.0 kernel from Carnegie Mellon University. OSF and Mach on the Paragon have both been been extended to support a distributed memory environment.

  Each node runs its own copy of OSF/1. This takes up about 10 megabytes per node.

- SUNMOS: NAS system developers are currently testing a second operating system for the Paragon–SUNMOS. SUNMOS can run concurrently with OSF but reduces the number of compute nodes available to OSF applications. It is much less flexible than OSF but provides better performance for message passing. SUNMOS does not run during regularly scheduled time, but may be requested during dedicated time. For more information on SUNMOS, see the nasgopher file "sunmos".
- NX: NX is a library of routines for message passing and management of parallel jobs. It is nearly identical to the NX message passing library on the Intel iPSC/860 (hypercube).
- NQS: NQS is used for submitting batch jobs that run at night.
- ipd: ipd is a debugger which handles parallel applications.
- PVM: Parallel Virtual Machine 3.2 is available.

C.4 The IBM SP-1 System at ANL

The IBM SP-1 is a new parallel computer designed to make the best use of IBM's powerful RISC technology combined with a high-speed switch. Special features of this SP-1 are:

- large memory per node (128 MBytes),
- local disks on each node (1 GByte),
- full Unix on each node (IBM AIX 3.2.4),
- high-performance nodes,
- high-performance switch,
- high I/O bandwidth of nodes.

The hardware configuration of the IBM SP-1 at Argonne National Labortroy (or the Argonne SP-1) consists of 128 nodes and two compile servers. Each node is essentially an RS/6000 model 370. This model has a 62.5 MHz clock speed, a 32-KB data cache, and a 32-KB instruction cache. Key features of this system are:

1. 128 Mbytes of memory per node
2. GBytes local disk on each node (400 Mbytes available to users, the rest for paging)
3. Full Unix on each node (IBM AIX 3.2.4)
4. Each node accessible by Ethernet from the internet
5. High-performance Omega switch (50 μ sec latency, 8.5 Mbytes/sec bandwidth when using EUI-H)

In addition, the ANL SP-1 will soon have a large high-performance fill system (220 Gbytes of RAID disk and a 6-T Byte automated tape library).

The peak performance of each node is 125 Mflops (1674-bit floating point add and 1 floating point multiply in each clock cycle). In practice, each node can achieve between 15 and 70 Mflops on Fortran code. Higher performance can be reached by using BLACS or ESSL routines.

Each SP-1 node is running a full Unix; most of the usual Unix tools are available. Users may log directly into any SP-1 node using telnet, rlogin, or rsh. The software of the ANL SP-1 includes multiple parallel programming environments, IBM's ESSL library, and performance debugging tools.

Communication between nodes can be carried out by many ways which one can choose. Most users will not use these directly; rather they will use one of the portable programming libraries. However, as the programming libraries use these transport layers to actually accomplish the communication, it is important to understand them so that the proper transport layer can be chosen.

The available transport layers are Ethernet, IP, Switch/IP, and EUI-H. Only the first two support multiple parallel jobs on the same node. Both versions of EUI can only run one process per node. In addition, EUI-H is incompatible with EUI and Switch/IP on the same nodes (though the SP-1 can be configured so that EUI-H runs on some nodes and EUI and Switch/IP run on the others; this is a common daytime configuration at ANL.

Using the Ethernet transport layer with all nodes connected by Ethernet, the SP-1 looks just like a collection of workstations. This method does suffer from the same drawbacks as any Ethernet-connected system: high latency (about 1 μsec) and low bandwidth (1 MByte/sec is shared among all processors). The IP transport layer provides enhanced performance to code written using Unix sockets for interprocessor communication.

EUI is IBM's message-passing interface to the high-performance switch. There are two versions: one that works with the Parallel Operating Environment (POE) and one that does not. EUI refers to the POE version. POE supports a parallel symbolic debugger (xpdbx) and a performance visualization tool (vt). However, its performance is inferior to EUI-H (latency is about 405 μsec). In addition, at most 64 nodes are available to EUI. EUI-H is an experimental, low-overhead implementation of the EUI interface. It does not support either xpdbx or vt. It is difficult to provide standard input to EUI-H programs. Also, it is not possible to produce gprof-style profiling information from EUI-H programs. All 128 nodes may be accessed using EUI-H when the machine is configured for that. Note that this version of EUI-H is the version of IBM Research; it contains only the Fortran bindings for EUI.

IP and EUI applications may share the switch; multiple IP applications may share both nodes and the switch. IP and EUI run under the "Parallel Operating Environment," or POE. POE includes a number of tools, such as a parallel debugger and ParaGraph-like visualization tool (vt). These two transport layers share a common interface to the switch known as lightspeed.

Parallel Libraries:

- Chameleon: Chameleon is a lightweight, portable message-passing system. It provides access to a wide range of transport layers, including EUI, EUI-H, PVM, and P4. Chameleon provides a common startup model that simplifies choosing a transport layer.
- Fortran M: Fortran M is a small set of extensions to Fortran that supports a modular approach to the construction of sequential and parallel programs. Fortran M programs use channels to connect processes which may be written in Fortran M or Fortran 77. Processors communicate by sending and receiving messages on channels. Channels and processes can be created dynamically, but programs remain deterministic unless specialized nondeterministic constructs are used.
- MPI: MPI (Message-Passing Interface) is a new message-passing system "standard" that has recently been defined by a broadly based group of parallel computing vendors, library writers, and users. The current draft is now in the public-comment stage.

Bibliography

[1] Roger F. Harrington, *Matrix Methods for Field Problems*, Proc. IEEE, Vol. 55, No. 2, pp. 136–149, Feb. 1967.

[2] Debra L. Wilkes and Chung-Chi Cha, *Method of Moments Solution with Parametric Curved Triangular Patches*, IEEE Antennas and Propagation International Symposium Digest, pp. 1512–1515, 1991.

[3] Roger F. Harrington, Field Computation by Moment Methods, Macmillan, New York, Reprinted by IEEE Press, Piscataway, NJ, 1993.

[4] J. H. Richmond, *A wire-grid model for scattering by conducting bodies*, IEEE Trans. Antennas Propagat., Vol. AP-14, No. 6, pp. 782–786, Nov. 1966.

[5] D. C. Kuo, H. H. Chao, J. R. Mautz, B. J. Strait, and R. F. Harrington, *Analysis of Radiation and Scattering by Arbitrary Configurations of Thin Wires*, IEEE Trans. Antennas Propagat., Vol. AP-20, pp. 814–815, Nov. 1972.

[6] X. C. Yuan, *Electromagnetic Coupling into Slotted TE and TM Cylindrical Conductors by the Pseudo-Image Method*, Ph.D. Dissertation, Syracuse University, Syracuse, NY, 1987.

[7] J. R. Mautz and R. F. Harrington, *Radiation and Scattering from Bodies of Revolution*, Appl. Sci. Res., Vol. 20, pp. 405–435, June 1969.

[8] R. F. Harrington and J. R. Mautz, *Radiation and Scattering from Loaded Bodies of Revolution*, Appl. Sci. Res., Vol. 26, pp. 209–217, June 1971.

[9] G. J. Burke and A. J. Poggio, *Numerical Electromagnetics Code (NEC)—Method of Moments*, Technical Document 116, AFWL-TR-76-320, Naval Ocean Systems Center, San Diego, CA, July 1977.

[10] E. K. Miller, and F. J. Deadrick, *Some computational aspects of thin-wire modeling*, in Numerical and Asymptotic Techniques in Electromagnetics, R. Mittra, Ed., New York: Springer-Verlag, 1975, Chapt. 4.

[11] K. S. H. Lee, L. Marin, and J. P. Castillo, *Limitations of wire-grid modeling of a closed surface*, IEEE Trans. Electromagn. Compat., Vol. EMC-18, No. 3, pp. 123–129, Aug. 1976.

[12] A. W. Glisson and D. R. Wilton, *Simple and efficient numerical Methods for Problems of Electromagnetic Radiation and Scattering from Surfaces*, IEEE Trans. Antennas Propagat., Vol. AP-28, No. 5, pp. 593–603, Sept. 1980.

[13] A. W. Glisson, *On the development of numerical techniques for treating arbitrarily-shaped surfaces*, Ph.D. dissertation, Univ. of Mississippi, 1978.

[14] S.M. Rao, D. R. Wilton and A. W. Glisson, *Electromagnetic Scattering by Surfaces of Arbitrary Shape*, IEEE Trans. Antennas Propagat., Vol. AP-30, pp. 409–418, May 1982.

[15] W. A. Johnson, D. R. Wilton, and R. M. Sharpe, *Modeling scattering from and radiation by arbitrary shaped objects with the electric field integral equation triangular surface PATCH code*, Electromagnetics, Vol. 10, Nos. 1–2, pp. 41–63, Jan.-June 1990.

[16] S. M. Rao, *Electromagnetic Scattering and Radiation of Arbitrarily-Shaped Surfaces by Triangular patch Modeling*, Ph.D. dissertation, Univ. of Mississippi, 1980.

[17] S. M. Rao, and D. R. Wilton, *E-Field, H-Field, and Combined Field Solution for Arbitrarily Shaped Three-Dimensional Dielectric Bodies*, Electromagnetics, Vol. 10, pp. 407–421, 1990.

[18] Roberto D. Graglia, *The Use of Parametric Elements in the Moment Method Solution of Static and Dynamic Volume Integral Equations*, IEEE Trans. Antennas Propagat., Vol. 36, No. 5, pp. 636–646, May 1988.

[19] Roberto D. Graglia, P. L. E. Uslenghi, and R. S. Zich, *Moment Method with Isoparametric Elements for Three-Dimensional Anisotropic Scatterers*, Proc. IEEE, Vol. 77, No. 5, pp. 750–760, May 1989.

[20] M. I. Sancer, R. L. McClary, and K. J. Glover, *Electromagnetic Computation Using Parametric Geomatry*, Electromagnetics, Vol. 10, pp. 85–103, 1990.

[21] Stephen Wandzura, *Electric Current Basis Functions for Curved Surfaces*, Electromagnetics, Vol. 12, pp. 77–91, 1992.

[22] Nathan J. Champagne II, Jeffery T. Williams, and Donald R. Wilton, *The use of curved segments for numerically modeling thin wire antennas and scatterers*, IEEE Trans. Antennas Propagat., Vol. 40, pp. 682–689, No. 6, June 1992.

[23] J. R. Mautz and R. F. Harrington, *H-field, E-field, and Combined-field Solutions for Conducting Bodies of Revolution*, AEÜ, Vol. 32, pp. 157–164, April 1978.

[24] J. R. Mautz and R. F. Harrington, *Electromagnetic Scattering from a Homogeneous Material Body of Revolution*, AEÜ, Vol. 33, pp. 71-80, Feb. 1979.

[25] R. F. Harrington, *Boundary Integral Formulations for Homogeneous Material Bodies*, J. Electro. Waves Applic., Vol. 3, No. 1, pp. 1–15, 1989.

[26] Q. Chen, *Electromagnetic Modeling of three-Dimensional Piecewise Homogeneous Material Bodies of Arbitrary Composition and Geometry*, Ph.D. dissertation, Univ. Houston, Houston, TX, May 1990.

[27] J. R. Mautz and R. F. Harrington, *Generalized Network Parameters, Radiation, and Scattering by Conducting Bodies of Revolution*, (Computer Program Description), IEEE Trans. Antennas Propagat., Vol. AP-22, pp. 630–631, July 1974.

[28] J. R. Mautz and R. F. Harrington, *Radiation and Scattering from Loaded Bodies of Revolution*, (Computer Program Description), IEEE Trans. Antennas Propagat., Vol. AP-23, p. 594, July 1975.

[29] J. R. Mautz and R. F. Harrington, *An Improved E-Field Solution for a Conducting Body of Revolution*, AEÜ, Vol. 36, pp. 198–206, May 1982.

[30] J. R. Mautz and R. F. Harrington, *Electromagnetic Coupling to a Conducting Body of Revolution with a Homogeneous Material Region*, Electromagnetics, Vol. 2, pp. 257–308, Oct.-Dec. 1982.

[31] G. C. Fox and Ian G. Angus, Solving Problems on Concurrent Processors, Prentice Hall, New Jersey, 1988.

[32] G. C. Fox, and A. Frey, *High performance parallel supercomputing application, hardware, and software issues for a teraflop computer*, Caltech Concurrent Computation Program Paper, C3P-451C, Nov. 1988.

[33] G. C. Fox, *1989 - The first year of the parallel supercomputer*, Caltech Concurrent Computation Program Paper, C3P-769, June 1990.

[34] T. Cwik, Robert van de Geijn, and J. Patterson, *Application of massivly parallel computation to integral equation models of electromagnetic scattering*, J. Opt. Soc. Am. A, Vol. 11, No. 4, pp. 1538–1545, April 1994.

[35] Tom Cwik, *Parallel Decomposition Methods for the Solution of Electromagnetic Scattering Problems*, Electromagnetics, Vol. 12, pp. 343–357, 1992.

[36] Tom Cwik, Jonathan Partee, and Jean Patterson, *Method of Moment Solutions to Scattering Problems in a Parallel Processing Environment*, IEEE Trans. Magnetics, Vol. 27, No. 5, pp. 3837–3840, Sept. 1991.

[37] Tom Cwik, Jean Patterson and David Scott, *Electromagnetic Scattering Calculations on the Intel Touchstone*, IEEE 1992.

[38] Jean E. Patterson, Tom Cwik, Robert D. Ferraro, Nathan Jacobi, Paulett C. Liewer, Thomas G. Lockhart, Gregory A. Lyzenga, Jay W. Parker, and Diglio A. Simoni, *Parallel Computation Applied to Electromagnetic Scattering and Radiation Analysis*, Electromagnetics, Vol. 10, pp. 21–39, 1990.

[39] Ruel H. Calalo, William A. Imbriale, Nathan Jacobi, Paulett C. Liewer, Thomas G. Lockhart, Gregory A. Lyzenga, James R. Lyons, Farzin Manshadi and Jean E. Patterson, *Hypercube Matrix Computation Task Report for 1986-1988*, JPL Publication 88-31, August 1, 1988.

[40] Ruel H. Calalo, Tom Cwik, Robert D. Ferraro, William A. Imbriale, Nathan Jacobi, Paulett C. Liewer, Thomas G. Lockhart, Gregory A. Lyzenga, Stephanie Mulligan, Jay W. Parker, Jean E. Patterson, *Hypercube Matrix Computation Task Research in Parallel Computational Electromagnetics*, Report for 1988-1989, JPL Nov. 1, 1989.

[41] Ruel H. Calalo, Tom Cwik, Robert D. Ferraro, William A. Imbriale, Nathan Jacobi, Paulett C. Liewer, Thomas G. Lockhart, Gregory A. Lyzenga, Stephanie Mulligan, Jay W. Parker, Jean E. Patterson, *Research in Parallel Computational Electromagnetics Hypercube Matrix Computation Task*, Quarterly Review, JPL, Nov. 29, 1989.

[42] Ruel H. Calalo, Tom Cwik, Robert D. Ferraro, William A. Imbriale, Nathan Jacobi, Paulett C. Liewer, Thomas G. Lockhart, Gregory A. Lyzenga, Stephanie Mulligan, Jay W. Parker, Jean E. Patterson, *Research in Parallel Computational*

Electromagnetics Hypercube Matrix Computation Task, Quarterly Review, JPL, April 6, 1990.

[43] T. Cwik, J. Partee, and J. patterson, *Integral Equation Solutions to Radiation and Scattering Problems Using Coarse-Grained Parallel Processor*, PIER Vol. 7, pp. 157-195, T. Cwik and J. Patterson ed. (EWM Publishing, Cambridge, MA. 1993.

[44] S.D. Gedney, A. F. Peterson, and R. Mittra, *The Moment Method Solution of Electromagnetic Scattering Problems on MIMD and SIMD Hypercube Supercomputers*, PIER Vol. 7, pp. 197-246, T. Cwik and J. Patterson ed. (EWM Publishing, Cambridge, MA. 1993.

[45] S. D. Gedney and R. Mittra, *The use of the FFT for the efficient solution of the problem of electromagnetic scattering by a body of revolution*, IEEE Trans. Antennas Propagat., Vol. 38, pp. 313–322, 1990.

[46] A. W. Glisson and D. R. Wilton, *Simple and efficient numerical techniques for treating bodies of revolution*, *University of Mississippi Engineering Experiment Station* Technical Report No. 105, 1979.

[47] Roger F. Harrington, Time-Harmonic Electromagnetic Fields, McGraw-Hill, 1961.

[48] J. Van Bladel, Electromagnetic Fields, McGraw-Hill, 1964.

[49] O. C. Zienkiewicz, The Finite Element Method (The third, expanded and revised edition of the Finite Element Method in Engineering Science), Chapter 8, McGraw-Hill, 1977.

[50] Debra L. Wilkes, *The Treatment of Singularity in MoM*, Private Communication, 1993.

[51] N. Morita, N. Kumagai, and J. R. Mautz, Integral Equation Methods for Electromagnetics, Artech House, 1990.

[52] Donald R. Wilton, *Review of Current Status and Trends in the Use of Integral Equations in Computational Electromagnetics*, Electromagnetics, Vol.12, pp. 287–341, 1992.

[53] J. R. Mautz and R. F. Harrington, *H-Field, E-Field, and combined-field solutions for conducting bodies of revolution*, Technical Report TR-77-2, Dept. of Electrical and Computer Engineering, Syracuse University, Syracuse, NY 13244, Feb. 1977.

[54] J. J. H Wang, V. K. Tripp, and J. E. Tehan, *The Magnetically Coated Conducting Surface as a Dual Conductor and Its Application to Antennas and Microwaves*, IEEE Trans. Antennas Propagat., Vol. 38, No. 7, pp. 1069–1077, July 1990.

[55] H. A. Ragheb, L. Shafai, and M. Hamid, *Plane Wave Scattering by a Conducting Elliptic Cylinder Coated by a Nonconfocal Dielectric*, IEEE Trans. Antennas Propagat., Vol. 39, No. 2, pp. 218–223, Feb. 1991.

[56] Xiaoyi Min, Wiemin Sun, Wang-jie Gesang, and Kun-Mu Chen, *An Efficient Formulation to Determine the Scattering Characteristics of a Conducting Body with Thin Magnetic Coatings*, IEEE Trans. Antennas Propagat., Vol. 39, No. 4, pp. 448–454, April 1991.

[57] S. M. Rao, C. C. Cha, R. L. Cravey and D. L. Wilkes, *Electromagnetic Scattering from Arbitrary Shaped Conducting Bodies Coated with Lossy Materials of Arbitrary Thickness*, IEEE Trans. Antennas Propagat., Vol. 39, No. 5, pp. 627–631, May 1991.

[58] M. A. Leontovich, Investigations on Radiowave Propagation, Part II, Moscow: Academy of Sciences, 1948.

[59] T. B. A. Senior, *Impedance Boundary Conditions for Imperfectly Conducting Surfaces*, Appl. Sci. Res., Vol. 8(B), pp. 418–436, 1960.

[60] T. B. A. Senior, *A Note on Impedance Boundary Conditions*, Can. J. Phys., Vol. 40, pp. 663–665, 1962.

[61] T. B. A. Senior, *Approximate Boundary Conditions*, IEEE Trans. Antennas Propagat., Vol. AP-29, No. 5, pp. 826–829, Sept. 1981.

[62] K. M. Mitzner, *An Integral Equation Approach to Scattering from a Body of Finite Conductivity*, Radio Science, Vol. 2, No. 12, pp. 1459-1470, Dec. 1967.

[63] Louis N. Medgyesi-Mitschang and John M. Putnam, *Integral Equation Formulations for Imperfectly Conducting Scatterers*, IEEE Trans. Antennas Propagat., Vol. AP-33, No. 2, pp. 206–214, February 1985.

[64] M. J. Flynn, *Some computer organizations and their effectiveness*, IEEE Trans. Comput., Vol. C-21, pp. 948–960, 1972.

[65] G. M. Amdahl, *Validity of Single-Processor Approach to Achieving Large-Scale Computing Capability*, Proc. AFIPS Conf., pp. 483–485, Reston, VA., 1967.

[66] Syracuse Research Corporation, Parametric Method of Moments (ParaMoM) RCS Prediction Package User's Manual, Version 1.0, SRC Technical Report TD 92-1321, October 1992.

[67] Thinking Machines Corporation, CMSSL for CM Fortran, Version 3.1, June 1993.

[68] Intel Corporation, iPSC/2 and iPSC/860 User's Guide, April 1991.

[69] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam, PVM 3 User's Guide and Reference Manual, Oak Ridge National Laboratories report ORNL/TM-12187, May 1993.

[70] J. Choi, J. J. Dongarra, R. Pozo, and D. W. Walker, *Scalapack: A scalable linear algebra library for distributed memory concurrent computers*, In Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation, pp. 120–127. IEEE Computer Society Press, 1992.

[71] J. J. Dongarra and R. A. van de Geijn. *Two-dimensional basic linear algebra communication subprograms*, Technical Report LAPACK working note 37,

Computer Science Department, University of Tennessee, Knoxville, TN, October 1991.

[72] Jack J. Dongarra, Robert A. van de Geijn, and David W. Walker, *A Look at Scalable Dense Linear Algebra Libraries*, Oak Ridge National Laboratories report ORNL/TM-12126, July 1992.

[73] Thinking Machines Corporation, CM-5 I/O System Programming Guide, Version 7.2, September 1993.

[74] Thinking Machines Corporation, CMMD Reference Manual Version 3.0, May 1993.

[75] Thinking Machines Corporation, CM Fortran Language Reference Manual, Version 2.1, January 1994.

[76] Thinking Machines Corporation, CM Fortran User's Guide Version 2.1, January 1994.

[77] Thinking Machines Corporation, CMMD User's Guide, Version 3.0, May 1993.

[78] Thinking Machines Corporation, CM-5 Technical Summary November 1993.

[79] Thinking Machines Corporation, CM-5 User's Guide Version 7.2, August 1993.

[80] Intel Corporation, Paragon^{TM} ProSolver^{TM}-DES Manual, December 1993.

[81] Alex C. Woo, Helen T. G. Wang, Michael J. Schuh, and Michael L. Sanders, *Benchmark Plate Radar Targets for the Validation of Computational Electromagnetics Programs*, IEEE Antennas Propagat. Magazine, Vol. 34, No. 6, pp. 52–56, Dec. 1992.

[82] Alex C. Woo, Helen T. G. Wang, Michael J. Schuh, and Michael L. Sanders, *Benchmark Plate Radar Targets for the Validation of Computational Electromagnetics Programs*, IEEE Antennas Propagat. Magazine, Vol. 35, No. 1, pp. 84–89, Feb. 1993.

[83] Jack J. Dongarra, Robert A. van de Geijn, and R. Clint Whaley, A Users' Guide to the BLACS, September 1993.

Biographical Data

Name:	Xianneng Shen
Date and Place of Birth:	January 1958, Jinhua, China
Degrees Awarded:	M.S. in Computer Engineering, 1993 Syracuse University, Syracuse, NY, USA M.S. in Electrical Engineering, 1985 Chengdu Institute of Radio Engineering, Chengdu, China B.S. in Electrical Engineering, 1982 Chengdu Institute of Radio Engineering, Chengdu, China

Professional Experience:	Graduate Assistant, Jan. 1993 - May 1994 NPAC at Syracuse University
	Graduate Assistant, Sept. 1987 - May 1992 ECE Department at Syracuse University
	Visiting Researcher, Oct. 1986 - Aug. 1987 ECE Department at Syracuse University
	Lecturer, July 1985 - Sept. 1986 Chengdu Institute of Radio Engineering, Chengdu, China