

# A Scalable Parallel Fast Multipole Method for Analysis of Scattering from Perfect Electrically Conducting Surfaces<sup>\*†</sup>

Bhanu Hariharan, Srinivas Aluru and Balasubramaniam Shanker

Dept. of Electrical and Computer Engineering

Iowa State University, Ames, IA, USA

bhanu@iastate.edu, aluru@iastate.edu, shanker@iastate.edu

## Abstract

In this paper, we develop a parallel Fast Multipole Method (FMM) based solution for computing the scattered electromagnetic fields from a Perfect Electrically Conducting (PEC) surface. The main contributions of this work are the development of parallel algorithms with the following characteristics: 1) provably efficient worst-case run-time irrespective of the shape of the scatterer, 2) communication-efficiency, and 3) guaranteed load balancing within a small constant factor. We have developed a scalable, parallel code and validated it against surfaces for which solution can be computed analytically, and against serial software. The efficiency and scalability of the code is demonstrated with experimental results on an IBM xSeries cluster. Though developed in the context of this particular application, our algorithms can be used in other applications involving parallel FMM.

## 1 Introduction

Computational electromagnetics (CEM) may be described as the science of solving Maxwell's equations and is applicable to the design of various devices. Technologies that are affected by its continued development include, communications and antennas, wireless, computer, biomedical, geophysical, opto-electronic, etc. Given the range of applications, continued efforts have been made to make electromagnetic field solvers more efficient. CEM solvers can be categorized into two: those that have been derived using either (i) differential or (ii) integral equations. Integral equation based methods are typically more useful for analyzing fields in open region problems. In these methods, the problem is cast as an integral equation and then reduced to a system of linear equations using the widely known Method of Moments (MoM). The memory required for storing the matrix for the linear system is  $O(N^2)$ , where  $N$  is the number of unknowns that is used to model the scatterer. Direct methods such as Gaussian elimination to solve the system of linear equations requires  $O(N^3)$  time, making it prohibitively expensive for large problems. Also, the iterative methods for the solution to the integral equation techniques require  $O(N^2)$  per iteration, not suited for large problems.

In 1993, Coifman *et. al.* [1] introduced the Fast Multipole Method (FMM) as applied to the Helmholtz equation, and when this is applied to CEM solvers it considerably alleviates both memory and computational requirements. Subsequently, this FMM augmented CEM solvers have been used in a variety of applications, including large scale scattering [1, 2, 3, 4, 5], scattering from quasi-planar objects [6, 7, 8], EMC/EMI analysis [9, 10], and mine detection techniques [11, 12]. A comprehensive survey of the state of the art may be found in [10].

The FMM-based solutions reduce run-time complexity by partitioning the computational domain into a hierarchy of subdivisions and using approximations at various levels of the subdivision to reduce overall

---

\*Research supported by NSF under CCR-9988347 and EIA-0130861.

†0-7695-1524-X/02 \$17.00 (c) 2002 IEEE

computational cost. FMM was first developed by Rokhlin in the context of solving Poisson’s equation [13], and shot into prominence with Greengard’s application of the FMM to solve the gravitational N-body problem [14]. Considerable research efforts have been directed at developing parallel algorithms and implementations for the FMM, especially in the context of the N-body problem [15, 16, 17, 18, 19, 20]. Parallel FMM based algorithms have also been applied in other areas such as molecular dynamics [21] and capacitance extraction [22]. As for CEM solvers, parallel FMM based methods have been developed for both distributed and shared memory architectures. Ottusch *et al.* [23] present a parallel CEM solver implemented in a threaded style on cache-coherent distributed shared memory architectures. Distributed memory implementations of FMM based CEM solvers using MPI are carried out by Manke *et al.* [24] and by Velamparambil *et al.* (for example, see [25]). In all of these algorithms, the general strategy is to assign nodes at each level of the tree representing the hierarchical subdivision to processors in a load balanced fashion. Velamparambil uses an interesting technique of performing a “dry run” of the algorithm to accurately estimate the work at each node of the tree, and later using this knowledge to perform accurate load balancing.

Irrespective of the application area, a major challenge in parallel FMM is to achieve load balance and communication efficiency independent of the spatial distribution of the unknowns in the computational domain. Though some significant advances are made in developing distribution-independent algorithms [26, 27], their promise has remained largely theoretical so far. Liu [19] and Teng [20] have performed a rigorous run-time analysis for a certain class of distributions.

In this paper, we develop a parallel FMM based CEM solver for computing the electromagnetic scattering from a PEC surface. Our contributions include providing a rigorous algorithmic framework for parallel FMM and design of efficient algorithmic strategies that guarantee a balanced load irrespective of the spatial distribution of the unknowns. We overcome the limitation of level by level parallelization employed by current methods and show how to parallelize across multiple levels simultaneously while taking dependencies into account. The chief contributions of this work are:

- Provably optimal worst-case run-time as a function of the problem size independent of the spatial distribution of unknowns in the domain.
- Communication-efficient algorithms for computing the farfield (for example, reduction in the number of communication rounds to  $O(\log \log N)$  for uniform distribution).
- Development of practically efficient algorithms building on a distribution-independent framework [26].
- Experimental verification and demonstration of rigorous algorithmic ideas in solving a complex, full-scale CEM application.

Though developed in the context of a CEM solver, the parallel FMM algorithms presented in this paper can be readily applied to other applications using the FMM. The rest of this paper is organized as follows: In Section 2, we describe the problem formulation and how FMM can be used to derive a fast algorithmic strategy. Section 3 contains the details of our parallel algorithms. Experimental results are presented in Section 4 and Section 5 concludes the paper.

## 2 Problem Formulation

Consider a closed perfect electrically conducting body (PEC) whose surface is denoted by  $S$ , and resides in free space. In what follows,  $\hat{n}$  denotes an outward pointing position dependent normal to  $S$ , and  $S_{\pm}$  denote hypothetical surfaces that lie  $\epsilon^{\pm}$  on either side of  $S$ . An electromagnetic field with components  $\mathbf{E}^i(\mathbf{r})$ , and  $\mathbf{H}^i(\mathbf{r})$  impinges upon the surface  $S$ . This field creates a current  $\mathbf{J}(\mathbf{r})$ , that in turn generates scattered fields  $\mathbf{E}^s(\mathbf{r})$  and  $\mathbf{H}^s(\mathbf{r})$ . To represent the scattered fields, we begin with an expression for the magnetic vector potential  $\mathbf{A}(\mathbf{r})$  that is related to the current as

$$\mathbf{A}(\mathbf{r}) = \frac{\mu_0}{4\pi} \int_S d\mathbf{r}' \mathbf{J}(\mathbf{r}') \frac{e^{-jkR}}{R} \quad (1)$$

where  $R = |\mathbf{R}| = |\mathbf{r} - \mathbf{r}'|$ , and  $\mu_0$  and  $\epsilon_0$  are the permeability and permittivity of free space, respectively. The electric and magnetic fields can then be expressed in terms of the magnetic potential as follows

$$\mathbf{E}^s(\mathbf{r}) = -j\omega \left( \mathcal{I} + \frac{\nabla\nabla}{k^2} \right) \cdot \mathbf{A}(\mathbf{r}) \quad (2a)$$

$$\mathbf{H}^s(\mathbf{r}) = \frac{1}{\mu_0} \nabla \times \mathbf{A}(\mathbf{r}) \quad (2b)$$

The electric field integral equation (EFIE) is constructed by imposing the boundary condition that the total electric field tangential to the surface is zero. In other words,  $\hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \mathbf{E}^i(\mathbf{r}) = -\hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \mathbf{E}^s(\mathbf{r}) \quad \forall \mathbf{r} \in S, S_-, S_+$ . This condition together with (2a) yields

$$\begin{aligned} \hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \mathbf{E}^i(\mathbf{r}) &= j\omega \hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \left( \mathcal{I} + \frac{\nabla\nabla}{k^2} \right) \cdot \mathbf{A}(\mathbf{r}) \quad \forall \mathbf{r} \in S, S_-, S_+ \\ &\doteq \mathcal{L}_e \{ \mathbf{J}(\mathbf{r}) \} \end{aligned} \quad (3)$$

As is well known, this equation holds true for both open and closed structures. Likewise, the magnetic field integral equation (MFIE) can be derived using the condition that the total magnetic field tangential to  $S_-$  vanishes. Thus,

$$\begin{aligned} \hat{\mathbf{n}} \times \mathbf{H}^i(\mathbf{r}) &= -\frac{1}{4\pi} \hat{\mathbf{n}} \times \nabla \times \mathbf{A}(\mathbf{r}) \quad \forall \mathbf{r} \in S_- \\ &\doteq \mathcal{L}_h \{ \mathbf{J}(\mathbf{r}) \} \end{aligned} \quad (4)$$

In what follows, we shall use the following notation for convenience:  $\mathcal{V}_e \{ \mathbf{E}^i(\mathbf{r}), \mathbf{H}^i(\mathbf{r}) \} = \hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \mathbf{E}^i(\mathbf{r})$  and  $\mathcal{V}_h \{ \mathbf{E}^i(\mathbf{r}), \mathbf{H}^i(\mathbf{r}) \} = \hat{\mathbf{n}} \times \mathbf{H}^i(\mathbf{r})$ . It is well known that the solutions to the EFIE and the MFIE are not unique when analyzing scattering at frequencies that are close to the interior resonance frequencies of the body. This is typically overcome using the combined field integral equation (CFIE), a linear combination of the EFIE and the MFIE. The solution to the latter is unique at resonance frequencies of a closed object. CFIE is prescribed as

$$\mathcal{V}_c \{ \mathbf{E}^i(\mathbf{r}), \mathbf{H}^i(\mathbf{r}) \} = \mathcal{L}_c \{ \mathbf{J}(\mathbf{r}) \} \quad \forall \mathbf{r} \in S_- \quad (5)$$

where  $\mathcal{V}_c \{ \mathbf{E}^i(\mathbf{r}), \mathbf{H}^i(\mathbf{r}) \} = -\beta/\eta_0 \mathcal{V}_e \{ \mathbf{E}^i(\mathbf{r}), \mathbf{H}^i(\mathbf{r}) \} + \mathcal{V}_h \{ \mathbf{E}^i(\mathbf{r}), \mathbf{H}^i(\mathbf{r}) \}$  and  $\mathcal{L}_c \{ \mathbf{J}(\mathbf{r}) \} = -\beta/\eta_0 \mathcal{L}_e \{ \mathbf{J}(\mathbf{r}) \} + \mathcal{L}_h \{ \mathbf{J}(\mathbf{r}) \}$ . In the above equations,  $\eta_0 = \sqrt{\mu_0/\epsilon_0}$  is the intrinsic impedance of the free space and is introduced for scaling purposes, and  $\beta$  is a positive (real) constant

that is greater than zero. Solution to (3), (4), and (5) is usually obtained using MoM. The first step in using this method is to represent the current using a set of basis functions, i.e.,

$$\mathbf{J}(\mathbf{r}) = \sum_{n=1}^N I_n \mathbf{S}_n(\mathbf{r}) \quad (6)$$

The basis functions chosen in our analysis are the RWG functions that require the surface  $S$  to be represented using a set of flat triangular panels and one basis function  $\mathbf{S}_n(\mathbf{r})$  is associated with each edge joining two triangles:

$$\mathbf{S}_n(\mathbf{r}) = \begin{cases} \frac{l_n}{2A_n^+} \boldsymbol{\rho}_n^+(\mathbf{r}) & \text{for } \mathbf{r} \in \Gamma_n^+ \\ \frac{l_n}{2A_n^-} \boldsymbol{\rho}_n^-(\mathbf{r}) & \text{for } \mathbf{r} \in \Gamma_n^- \\ 0 & \text{elsewhere,} \end{cases} \quad (7)$$

where  $l_n$  is the length of the common edge between the triangles  $\Gamma_n^+$  and  $\Gamma_n^-$ ,  $A_n^\pm$  is the area of the triangle  $\Gamma_n^\pm$ , and  $\boldsymbol{\rho}_n^\pm(\mathbf{r})$  is the position vector with respect to the free vertex of the corresponding triangle [28].

Substituting  $\mathbf{J}(\mathbf{r})$  in (5), and using Galerkin testing leads to a matrix equation of the form

$$\mathcal{Z}_q \mathcal{I} = \mathcal{F}_q \quad (8)$$

where  $q = e, h, c$ ;  $\mathcal{I} = [I_1, I_2, \dots, I_N]$ ;  $\mathcal{F}_{c,m} = \langle \mathbf{S}_m(\mathbf{r}), \mathcal{V}_c \{ \mathbf{E}^i(\mathbf{r}), \mathbf{H}^i(\mathbf{r}) \} \rangle$  and  $\mathcal{Z}_{c,mn}^i = \langle \mathbf{S}_m(\mathbf{r}), \mathcal{L}_c \{ \mathbf{S}_n(\mathbf{r}) \} \rangle$ . Here  $\langle \cdot, \cdot \rangle$  denotes a standard inner product. It is well known that the solution to the matrix equation scales as  $O(N^2)$  if standard iterative techniques are used. However, these codes can be augmented with a hierarchical computational scheme like the FMM that will considerably reduce the computational cost as well as memory. In the next subsection, a brief description of FMM is presented.

## 2.1 FMM: A brief description

FMM relies on the fact that the fields radiated by a spatially bound source are quasi-bandlimited, and can be expressed in terms of plane waves to an arbitrary degree of accuracy. Consider two spheres of radius  $R_s$ , centered at  $\mathbf{r}_s^c$  and  $\mathbf{r}_o^c$  and labelled  $\alpha_s$  and  $\alpha_o$ , that contain a set of sources and observers. The field radiated by a source basis function  $\mathbf{S}_n(\mathbf{r})$  for  $n \in \alpha_s$  is tested by  $\mathbf{S}_m(\mathbf{r})$  for  $m \in \alpha_o$ . In other words, one needs to evaluate  $\langle \mathbf{S}_m(\mathbf{r}), \mathcal{L}_c \{ \mathbf{S}_n(\mathbf{r}) \} \rangle$ . It can be shown that

$$\langle \mathbf{S}_m(\mathbf{r}), \mathcal{L}_c \{ \mathbf{S}_n(\mathbf{r}) \} \rangle = \frac{1}{8\pi^2 c^2} \sum_{k=0}^M \sum_{p=-M}^M w_{kp} \left[ -\beta \mathcal{S}_m^- \left( \hat{\mathbf{k}}_{kp}, \hat{\mathbf{k}}_{kp} \right) + \mathcal{S}_m^- \left( \hat{\mathbf{k}}_{kp}, \hat{n}_{kp} \right) \right]^T \mathcal{T} \left( \hat{\mathbf{k}}_{kp}, M \right) \left[ \mathcal{S}_n^+ \left( \hat{\mathbf{k}}_{kp}, \hat{\mathbf{k}}_{kp} \right) \right] I_n \quad (9)$$

In (9),  $M = \lfloor 2k\chi R_s + 1 \rfloor$  is the number of harmonics,  $\chi$  is an oversampling factor,  $w_{pq}$  are quadrature weights, and  $\hat{\mathbf{k}}_{kp}$  are the plane wave directions [29]. The translation operator  $\mathcal{T} \left( \hat{\mathbf{k}}_{kp}, M \right)$  and the slant-stack operator are given as

$$\mathcal{T} \left( \hat{\mathbf{k}}_{kp}, M \right) = -\omega^2 \sum_{\nu=0}^M (-j)^\nu (2\nu + 1) h_\nu^{(2)}(k |\mathbf{R}_{\alpha_o \alpha_s}^c|) P_\nu \left( \hat{\mathbf{k}}_{kp} \cdot \mathbf{R}_{\alpha_o \alpha_s}^c / |\mathbf{R}_{\alpha_o \alpha_s}^c| \right) \quad (10a)$$

$$\mathcal{S}_o^\pm(\hat{\mathbf{k}}, t, \hat{\mathbf{v}}) = \int_{S_o} dS' \hat{\mathbf{v}} \times \mathbf{S}_o(\mathbf{r}') e^{\pm jk \hat{\mathbf{k}}_{pq} \cdot (\mathbf{r}' - \mathbf{r}_o^c)}, \quad (10b)$$

for  $o = \{m, n\}$ ,  $\mathbf{R}_{\alpha_o \alpha_s}^c = \mathbf{r}_o^c - \mathbf{r}_s^c$ ,  $P_\nu(\cdot)$  is a Legendre polynomial of degree  $\nu$ , and  $h_\nu^{(2)}(\cdot)$  is a Hankel function of the second kind. Equation (10b) indicates that the field radiated by all  $n \in \alpha_s$  may be computed by first projecting it on to a set of plane waves that travel out of the source sphere  $\alpha_s$ . These are then translated onto another set of plane waves that enter the observation sphere using a diagonal translation operator. Finally, the incoming rays are projected on to all basis  $m \in \alpha_o$ . This completes the evaluation of a matrix-vector product. This procedure can be nested within itself and one can arrive at a hierarchical procedure that scales as  $\mathcal{O}(N \log N)$  for computing scattering from surface unknown distributions.

### 3 Parallel Algorithms

#### 3.1 Computational Framework

The scatterer is enclosed in a fictitious cubical domain. The solution to (8) is typically obtained using an iterative solver like TFQMR (Transpose Free Quasi-Minimal Residual). The matrix vector product that is required by such an iterative solver is computed as follows: The matrix is decomposed into a nearfield component and a farfield component. The nearfield portion of the matrix represents the interactions between basis functions that are “close” (to be made precise later) to each other, and is computed once and used in every iteration. The product of the farfield component of the matrix multiplied by the vector is directly computed using the FMM algorithm in each iteration.

The contribution to the matrix vector product due to the farfield component is computed as follows: A compressed octree [30] is built using a recursive decomposition of the domain such that each leaf node in the tree represents a cubical box of side length approximately  $0.4\lambda$ . The algorithm represents the spectrum of electromagnetic field radiated by the sources that reside in a box using a set of sample points, corresponding to plane wave directions [1, 7]. The number of sample points is proportional to the square of the side length of the box. The work done at each level depends on the number of sample points and hence the problem is often referred to as *dynamic* FMM, different from the *static* FMM used to solve the well known N-body problem.

In what follows, we use the terms *node* (in the tree) and the *box* represented by the node interchangeably, provided such usage does not result in confusion. For the leaf boxes, the sample points are computed from the basis functions in the leaf box. For each internal box, the sample points are computed from the sample points of its child boxes, using an operation called *interpolation*. This involves interpolating the sample points of the children to obtain the values along a set of directions associated with the parent box, shifting to the center of the parent box and then merging. Interpolation can be done by using a straightforward algorithm ( $\mathcal{O}(k^2)$  time), an FFT based algorithm ( $\mathcal{O}(k \log k)$  time), or a Lagrange interpolation ( $\mathcal{O}(k)$  time), where  $k$  is the total number of original and interpolated sample points. In our CEM solver, we use an FFT/band-limited algorithm. The sample points for all the boxes are computed using a bottom-up tree traversal.

The next step is to compute the field at all the boxes. This is done by using the following classification: A pair of boxes of the same size are said to be in the *near field* of each other if they are adjacent, and in the *far field* otherwise. For a given box, interactions with boxes in the near field have to be resolved at lower levels in the tree. Interaction with a box in its far field can be computed at a higher level in the tree, if the respective parent boxes are in the near field of each other. Thus, the boxes interacting with a given box are

those in its far field whose parents are in the near field of its parent. We use the phrase *interaction list* to describe such boxes. For each box, the fields from the boxes in its interaction list are *translated*. The far field at a node is the result of aggregating the results of translations in all its ancestral nodes. This is computed using a top-down traversal. For each node, the results of translations from its interaction list are combined with the parent’s far field (by approximately halving the number of sample points and shifting the field to the center of the child box) to compute its far field. This operation is known as *anterpolation*. The sample points at the leaf boxes are then projected back to the basis functions.

Thus, the farfield FMM computation consists of: 1) Building the compressed octree, 2) computing interpolations using a bottom-up traversal, 3) computing translations for each box using its interaction list, 4) computing the anterpolations using a top-down traversal and 5) projecting the field at leaf boxes back to the observer basis functions using diagonal translation operator.

Care must be taken while performing the interpolation and anterpolation operations, especially for vector problems. Typically, one would translate all three Cartesian components; however, note that (10b) can be expressed in terms of  $\hat{\theta}$  and  $\hat{\phi}$  components only. Thus, it is possible to evaluate (9) using only *two* components. It should be noted that if one were to use a scalar spherical filter [31] for interpolation/anterpolation, all three components are required. Alternatively, one needs to use a vector spherical filter if only two components are to be used [32].

## 3.2 Preprocessing

Preprocessing of the geometry is performed on every processor. This includes setting electromagnetic constants, computing  $x, y, z$  components of the incident field, computing the lengths of all edges and initializing arrays for storing the node, patch and edge information. Each basis function is associated with a number and a type (source or observer type) that uniquely identifies the basis function. The basis functions are distributed among the processors based on the numbers. This is the initial data partitioning and is trivial.

## 3.3 Constructing the Tree

In this section, we present our parallel algorithm for constructing the compressed octree. We use compressed octrees to be able to design provably efficient algorithms with optimal worst-case runtime independent of the shape of the scatterer. As will be evident later, tree construction requires an insignificant amount of the total run-time. However, this stage is very important because tree partitioning is the key determinant of the load balancing and communication efficiency of subsequent stages. We store the tree in postorder traversal and this helps in computing the farfield component of the matrix vector product efficiently. No explicit load balancing is required in our implementation.

Let  $D$  be the side length of the domain and  $l$  be the side length of a box. The level of the box is defined to be  $\log_2 \frac{D}{l}$ . The level of a node in the compressed octree is defined to be the level of its box. Let  $L$  denote the number of levels in the tree, which is determined by the leaf box size and the size of the domain enclosing the scatterer. In practice,  $L$  is quite small. For instance, 20 levels allows us to have a tree with potentially  $8^{20} = 2^{60}$  leaf nodes, enough to capture a wide variety of distributions for even the largest simulations.

Following [16], we represent boxes using integer keys. For a given box, the domain can be visualized as partitioned into boxes of the same size. A box can be represented by an integer triple  $(x, y, z)$ , denoted by the  $x, y$  and  $z$  coordinates of the box in this box space. The key is formed by interleaving the bits of  $x,$

$y$  and  $z$  in that order, and prepending it with a 1 bit. This unique and unambiguous representation of all the boxes in the tree corresponds to ordering them according to Morton ordering, also known as  $Z$ -space filling curve ordering [33]. For any pair of boxes, either the two boxes are disjoint or one of them is completely contained in the other box. We consider two boxes to be disjoint if they merely touch at the boundaries. Define an ordering of a pair of boxes as follows: If a box is completely contained in another, then the box with smaller size appears first in the ordering. If they are disjoint, find the smallest box that contains both these boxes. Each of the two boxes will be contained in a distinct immediate subbox of this smallest box. Order the two boxes according to the value of the keys of the corresponding immediate subboxes. These operations can be done using bit arithmetic, and are taken as  $O(1)$  time operations. We order the nodes in the tree according to their boxes. In drawing the tree, we make use of this ordering. Thus, ordering of all the nodes corresponds to the postorder traversal of the nodes in the tree. Furthermore, given two nodes  $v_1$  and  $v_2$  in the tree, the smallest box containing their boxes corresponds to the lowest common ancestor of  $v_1$  and  $v_2$ .

The compressed octree is constructed as follows: Each processor is initially given  $\frac{N}{P}$  source basis functions. Every processor is also given an equal number of observer basis functions. The location of the basis functions is used to generate the corresponding leaf box. For every source basis function, there is an observer basis function which maps to the same leaf box as the source basis function. The basis functions are sorted in parallel, collecting the basis functions that fall within the same leaf box. Each processor borrows the first leaf box from the next processor and runs a sequential algorithm to construct the compressed octree for its leaf boxes together with the borrowed box in  $O\left(\frac{N}{P} \log \frac{N}{P}\right)$  run-time [30]. This local tree is stored in postorder traversal order in an array. Each node stores the indices of its parent and children in the array.

**Lemma 1** *Consider the compressed octree for the  $N$  basis functions, which we term the global compressed octree. Each node in this tree can be found in the local tree of at least one processor.*

**Proof:** Every internal node is the lowest common ancestor of at least one consecutive pair of leaves. More specifically, let  $u$  be a node in the global compressed octree and let  $v_1, v_2, \dots, v_k$  be its children ( $2 \leq k \leq 8$ ). Consider any two consecutive children  $v_i$  and  $v_{i+1}$  ( $1 \leq i \leq k-1$ ). Then,  $u$  is the lowest common ancestor of the rightmost leaf in  $v_i$ 's subtree and the leftmost leaf in  $v_{i+1}$ 's subtree. If we generate the lowest common ancestors of every consecutive pair of leaf nodes, we are guaranteed to generate every internal node in the compressed octree. However, there may be duplicates – each internal node is generated at least once but at most seven times. ■

Each node in a local compressed octree is also a node in the global compressed octree. In order to generate the postorder traversal order of the global compressed octree, nodes which should actually appear later in the postorder traversal of the global tree, should be sent to the appropriate processors. Such nodes, termed *out of order nodes*, appear consecutively after the borrowed leaf in the postorder traversal of the local tree and hence can be easily identified. Also, the borrowed leaf makes sure that the lowest common ancestor of every consecutive pair of leaves is generated; it is not considered a part of the local tree. An *MPI Allgather* operation is used to collect the first leaf box in each processor into an array of size  $P$ . Using a binary search in this array, the destination processor for each out of order node can be found. Nodes that should be routed to the same processor are collected together and sent in a single message, using *MPI Alltoall*.

**Lemma 2** *The number of out of order nodes in a processor is at most  $L$ .*

**Proof:** We show that there can be at most one node per level in the local tree that is out of order. Suppose this is not true. Consider any two out of order nodes in a level, say  $v_1$  and  $v_2$ . Suppose  $v_2$  occurs to the right of  $v_1$  ( $v_2$  comes after  $v_1$  in box order). Since  $v_1$  and  $v_2$  are generated, some leaves in their subtrees belong to the processor. Because the leaves allocated to a processor are consecutive, the rightmost leaf in the subtree of  $v_1$  belongs to the processor. This means  $v_1$  is not an out of order node, a contradiction. ■

**Lemma 3** *The total number of nodes received by a processor is at most  $7L$ .*

**Proof:** We first show that a processor cannot receive more than one distinct node per level. Suppose this is not true. Let  $v_1$  and  $v_2$  be two distinct nodes received at the same level. Without loss of generality, let  $v_2$  be to the right of  $v_1$ . A node is received by a processor only if it contains the rightmost leaf in the subtree of the node. Therefore, all the leaf nodes between the rightmost leaf in the subtree of  $v_1$  and the rightmost leaf in the subtree of  $v_2$  must be contained in the same processor. In that case, the entire subtree under  $v_2$  is generated on the processor itself and the processor could not have received  $v_2$  from another processor, a contradiction. Hence, a processor can receive at most  $l$  distinct nodes from other processors. As there can be at most 7 duplicate copies of each node, a processor may receive no more than  $7L$  nodes. ■

The received nodes are merged with the local postorder traversal array and their positions are communicated back to the sending processors. The net result is the postorder traversal of the global octree distributed across processors. Each node contains the position of its parent and each of its children in this array. From here onwards, we use the term *local tree* or *local array* to refer to the local portion of the global compressed octree stored in postorder traversal order. The size of the local tree is  $O\left(\frac{N}{P} + L\right)$ .

### 3.4 Near Field Computation

A box  $a$  is said to be in the nearfield of  $b$ , if  $a$  and  $b$  are of the same size and  $a$  is adjacent to  $b$ , i.e, if  $a$  lies within a shell around the box  $b$ . Otherwise, they are said to be in the farfield of each other. The nearfield portion of the matrix is a consequence of the interactions between basis functions from the same or adjacent leaf boxes. Direct computation is used for nearby interactions. Since the geometry is fixed throughout the number of iterations required for solving the matrix vector equation, the nearfield matrix is built only once and used in every iteration.

Let  $A$  be an array size  $O\left(\frac{N}{P}\right)$  that stores the basis functions. The basis functions carry the information about the leaf node that they belong to. The array  $A$  is also sorted along with the leaf node array during the parallel tree construction phase. As a result, basis functions whose centers are contained in the same leaf node occupy consecutive positions in  $A$ . Hence the mapping of the leaf nodes to the basis functions that belong to it, can be easily maintained.

Each processor computes part of the nearfield portion of the matrix, corresponding to the basis functions in its leaf boxes. For computing this, accessing information from basis functions in adjacent leaf boxes is necessary. Because of the postorder traversal order, it is easy to determine the processor responsible for a leaf box based on the coordinates of the leaf box. For this purpose, another array  $B$  of size  $2P$  is built by gathering the integer keys of the first and last leaf nodes of every processor. This is done using the *MPIAllgather* primitive so every processor has the boundary information. Binary search in  $B$  is used to determine the processor responsible for a leaf box.

The basis functions required at a processor can be separated into two categories: those that are available locally within the same processor, and those that reside in other processors. Each processor sends information of its leaf boxes to processors that should contain leaf boxes adjacent to it. This requires only one *MPI\_Alltoall* communication, as opposed to requesting information for adjacent boxes, followed by receiving it.

Note that a leaf box information sent to a processor may not be needed, because the adjacent leaf box prompting the communication may be non-existent. Similarly, an expected leaf box may not arrive because it is non-existent. These can be easily detected by ordering the received leaf boxes in postorder traversal order. The leaf boxes received from each processor are already in this order. Hence, merging the received lists of leaf boxes is enough to create the required postorder traversal.

It is easy to see that the number of leaf boxes sent by or received on a processor is bounded by  $O(\frac{N}{P})$ . Although a better bound can be established, and the number of out of processor leaf boxes necessary will be even smaller because of the locality properties of the Morton order, this is sufficient to prove that run-time complexity of this phase is  $O(\frac{N}{P} \log \frac{N}{P})$ . This is because each leaf box has at most 26 neighboring leaf boxes and the algorithm performs 26 binary searches for each of the leaf boxes. The work per leaf box is proportional to the number of basis functions that belong to it and the number of basis functions that belong to the leaf boxes adjacent to it. If each leaf box has at most  $c$  basis functions, then the number of interactions per leaf box in this phase is at most  $27*c^2$ .

### 3.5 Building Interaction Lists

For each box, the interaction list specifies the list of boxes which are in its farfield but whose parent boxes are in the nearfield of its parent. As with the nearfield matrix computation, the interaction lists are also built just once and used in every iteration. But the interaction lists are built for all the nodes in the tree and not just the leaf nodes.

For a box in a compressed octree, its parent box, corresponding to the box of the parent node in the tree, need not necessarily be a box of double the side length of that of the child box and may reside on some other processor. Therefore, it is convenient to compute the parent box using the following lemma.

**Lemma 4** *The parent of a box is the smallest box containing it and its adjacent box in the postorder traversal order.*

**Proof:** If the box is the rightmost child of its parent, then the parent box will be adjacent to it in the postorder traversal order. In this case, the smallest box containing the child box and its adjacent box will be the adjacent box itself, which is the parent. If the box is not the rightmost child of its parent, then the adjacent box will be the leftmost leaf box in the subtree of its next sibling. The lowest common ancestor of these two boxes will be the parent box, which proves the lemma. ■

Following are the steps to compute the interaction list of box,  $b$ :

1. Using the above lemma, compute the parent box of  $b$ .
2. Obtain the boxes in the nearfield of the parent box using bit arithmetic operations.

3. Compute subboxes of the boxes obtained in Step 2 such that size of each subbox is equal to the size of  $b$ . Discard those subboxes that are in the near field of  $b$ .
4. Partition these subboxes into two arrays for convenience : an array *Local* for subboxes that are local to the processor, and an array *Remote* for subboxes that are remote to the processor. The *Remote* array will thus have all the nodes whose information needs to be fetched at the *translation* phase of each iteration.

Following the building of interaction lists, multiple iterations of the remaining stages are run until convergence. At the end of each iteration, the difference between the matrix vector product and the right hand side is used to improve the solution vector.

### 3.6 Computing Interpolations

In this subsection, we describe our algorithm to compute the field radiated by each box using interpolations. First, each processor scans its local array from left to right. During the scan, each node is interpolated to its parent, provided the parent is local to the processor. As the tree is stored in postorder traversal order, if all the children of a node are present in the same processor, it is encountered only after all its children are. This ensures that the field due to a box is known when the scan reaches it. This computation takes  $O\left(\frac{N}{P} + L\right)$  time. During the scan, some nodes are labelled *residual nodes* based on the following rules:

- If the field due to a box is known but its parent lies in a different processor, it is labelled a *residual leaf node*.
- If the field at a node is not yet computed when it is visited, it is labelled a *residual internal node*.

Each processor copies its residual nodes into an array. It is easy to see that the residual nodes form a tree (termed the *residual tree*) and the tree is present in its postorder traversal order, distributed across processors.

**Lemma 5** *The number of residual internal nodes in a processor is at most  $L - 1$ .*

**Proof:** We show that there can be at most one residual internal node per level in a processor. Suppose this is not true. Consider any two residual internal nodes in a level, say  $v_1$  and  $v_2$ . Suppose  $v_2$  occurs to the right of  $v_1$  ( $v_2$  comes after  $v_1$  in box order). Because of postorder property, the right most leaf in  $v_1$ 's subtree and the rightmost leaf in  $v_2$ 's subtree should belong to this processor. Because the leaves allocated to a processor are consecutive, all the leaves in the subtree of  $v_2$  belong to the processor. This means  $v_2$  is not a residual internal node, a contradiction. Therefore, there can be only one residual internal node at each level in a processor. Furthermore, no leaf can be a residual internal node. Hence, the claim. ■

**Corollary 6** *The size of the residual tree is at most  $8PL$ .*

**Proof:** From the previous lemma, the total number of residual internal nodes is at most  $P(L-1)$ . It follows that the maximum number of residual leaf nodes can at most be  $7P(L-1) + 1$ , for a total tree size of at most  $8P(L-1) + 1 < 8PL$ . ■

The interpolation operation has the associative property. Because of this property, interpolations on the residual tree can be computed using an efficient parallel upward tree accumulation algorithm. We use the algorithm by Sevilgen *et al.* [26]. The number of communication rounds required by the algorithm is the logarithm of the size of the tree. During each round of communication, the algorithm uses an *MPI\_Scan* (parallel prefix) and *MPI\_Alltoall*. The algorithm could have been directly applied to the global compressed octree but this would require  $O(\log N)$  rounds. The residual tree can be accumulated in  $O(\log P + \log L)$  rounds. Thus, we reduce the worst-case number of communications from the logarithm of the size of the tree to the logarithm of the height of the tree, which is much smaller!

It is important to understand the implication of the reduction in the number of communication rounds. Computing interpolations and antepolations are the most communication-demanding stages in the application. For a uniform distribution, the height of the tree is  $O(\log N)$ , reducing the number of communications to  $O(\log \log N)$ . For an exponential distribution, the height of the tree is  $O(N)$ . Though the general proof of the algorithm suggests that the worst-case number of communications is  $O(\log N)$  in this case, the number of actual communication rounds required by the algorithm is just one! Based on a number of problem instances we have seen, we conjecture that the worst-case number of communications is  $O(\log \log N)$ .

### 3.7 Computing Translations

To compute translations at a node, the field at each of the nodes in its interaction list is needed. The interaction lists built in the preprocessing stage are used for this stage. First, an *MPI\_Alltoall* is used to request fields of nodes from the interaction lists that reside on remote processors. Another *MPI\_Alltoall* is used to receive the fields at these nodes. Because the interaction lists remain the same through all the iterations, requests for nodes in interaction lists need to be done only in the first iteration, thereby reducing the communication to one *MPI\_Alltoall* in subsequent iterations.

A query box generated at the time of building interaction lists could be empty or could contain one subbox with the remaining region being empty (see Figure 1). It is easy to modify the binary search algorithm on postorder traversal to ensure that a search for such a box retrieves the corresponding subbox. In Figure 1, interaction list of  $v$  contains box  $c$ . However  $c$  is not present in the tree because it is empty except for the region given by subbox  $e$ . Instead of  $c$ , subbox  $e$  should be returned in response to the query. Once all the information is available locally, the translations are conducted within each processor. Each processor performs  $O(\frac{N}{P})$  translations. Note that translations may be needed between boxes of different sizes. Oversampling is used when a smaller box is translated to a bigger box.

### 3.8 Computing Antepolations

Similar to the interpolation operation, it can be shown that the antepolation operation is also associative. Taking advantage of this property, antepolations can be computed using a reverse of the algorithm for computing interpolations. First, the antepolations for the residual tree are calculated. This requires  $O(\log P + \log L)$  communication rounds. Then, the antepolations for the local tree are computed using a right-to-left scan of the postorder traversal of the local tree. The exact number of communication rounds required is the same as in computing interpolations. The sample points at the leaf boxes are then projected back to the observer basis functions.

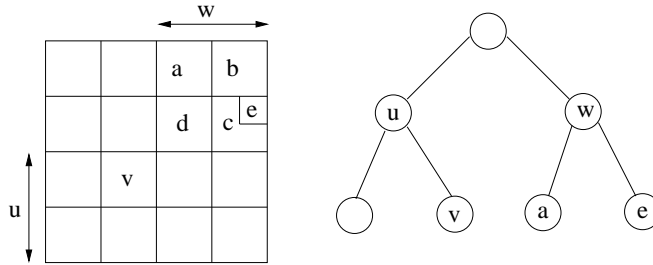


Figure 1: Illustration of translations. Node  $u$ , parent of node  $v$ , has node  $w$  in its near field.  $v$  generates subboxes  $a$ ,  $b$ ,  $c$  and  $d$  of box  $w$ . Subbox  $a$  is translated to  $v$ ; Subboxes  $b$  and  $d$  are empty; Subbox  $c$  has a childbox  $e$  and rest of the region is empty. Therefore,  $e$  should be translated to  $v$ .

## 4 Experimental Results

We developed parallel code that takes the geometry of the surface, details of the impinging wave etc. into account and computes the electromagnetic scattering from the surface. It is a combined C and Fortran code using MPI for parallel communication. The numerical kernels, such as those for translation, interpolation and anterpolation, are coded in Fortran. The remaining part of the program is coded in C. It is a complete end-to-end application software with controllable accuracy, incorporating all the parallel algorithms described earlier and other necessary tasks such as the creation of basis functions themselves from the geometry etc. We had earlier developed a serial code, which is used both to validate the parallel software and to compute the runtime speedups achieved by it.

We experimentally tested the software using an IBM xSeries cluster. The cluster consists of 32 dual processor nodes, each running at 1.26 GHZ and containing 1GB memory per node. The nodes are connected using Myrinet to support parallel communication at the rate of 2Gbits/sec, full duplex. We tested the software using two different types of surfaces: 1) A sphere and a 2) a 3-dimensional  $L$ -shaped surface. The spherical surface is used as a representative uniform distribution and the  $L$ -shaped surfaces serves as a non-uniform distribution.

The run-times for solving the complete application for 20,000 and 40,000 unknowns for both uniform and non-uniform distributions as a function of the number of processors are shown in Figure 2. From the graphs, it is easy to see that the algorithm works well for both types of distributions, as predicted by the theoretical analysis. It also scales well with number of processors.

The breakdown of the run-times into times spent in various stages of the application for solving the problem with 20,000 unknowns are shown in Table 1 and Table 2 for uniform and non-uniform distributions, respectively. We show the breakdown of the run-times for 20,000 unknowns to illustrate that our algorithm scales very well even for a small problem size. The nearfield and farfield components (including interpolations, translations and anterpolations) take approximately half the running time each, with a negligible amount of time spent in constructing the tree and building the interaction lists. The last column in each of these tables lists the speedup when compared to running the serial code for the same problem on one processor of the cluster. Table 3 shows the run-times of the various stages for a problem size of 1,000,000 unknowns for a uniform distribution. As expected, with increase in problem size, most of the run-time is spent in the farfield component.

The scaling of the run-time with problem size for a fixed number of processors is shown in Figure 3.

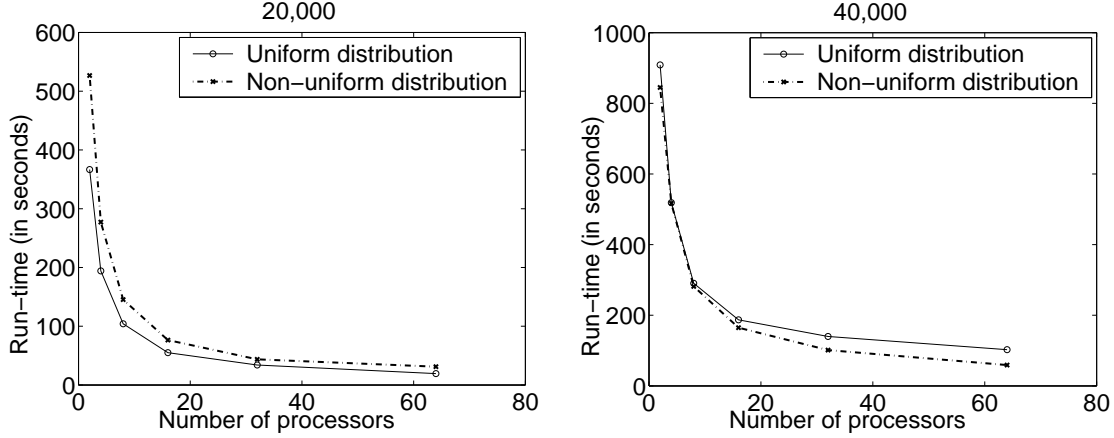


Figure 2: Total run-times for solving the problem as a function of the number of processors for 20,000 and 40,000 unknowns.

Number of procs	Tree const.	Nearfield computation	Building inter-action lists	Interpolation	Translation	Anterpolation	Total	Speed-up
1	0.25	381.06	0.10	109.97	116.20	119.39	726.97	1
2	0.65	174.46	0.76	55.54	73.93	60.82	366.16	1.99
4	0.33	87.72	0.39	28.44	46.50	30.66	194.04	3.75
8	0.16	44.95	0.20	14.54	28.11	16.02	103.98	6.99
16	0.07	23.13	0.12	7.66	15.68	8.30	54.96	13.23
32	0.05	12.38	0.06	4.71	11.78	4.95	33.93	21.42
64	0.05	6.41	0.04	2.77	7.33	2.77	19.37	37.53

Table 1: Run-times in seconds for various stages of the algorithm for 20,000 unknowns for uniform distribution.

The graph to the left shows the run-time for one iteration of farfield computation. The graph to the right shows the total application run-time. The number of iterations for convergence increases with increase in the number of unknowns and this behavior is reflected in the growth of the application run-time.

Finally, it is necessary to demonstrate that the results generated by this code are indeed valid. To establish this fact, the results generated by our parallel software are compared against analytical solutions for canonical problems. Here we choose to compute the field scattered by a sphere of radius  $1m$  that is illuminated by an incident plane wave that is propagating in the  $-\hat{z}$  direction, and is  $\hat{x}$  polarized. The sphere is discretized using 518,310 basis functions and the frequency of the incident wave is 4.2GHz. As is evident from Figure 4, the agreement between the analytical and numerical RCS data is excellent.

## 5 Conclusions and Future work

In this paper, we reported the development of a scalable, parallel computational electromagnetics solver for computing the scattering from three dimensional perfect electrically conducting surfaces. The software is based on fundamental algorithmic advances that guarantee efficient run-time irrespective of the shape of

Number of procs	Tree const.	Nearfield computation	Building inter-action lists	Interpolation	Translation	Anterpolation	Total	Speed-up
1	0.21	817.30	0.03	130.13	30.36	140.68	1118.71	1
2	0.64	368.35	0.12	65.96	20.77	70.04	525.88	2.13
4	0.33	190.51	0.07	33.78	16.06	36.04	276.79	4.04
8	0.17	98.22	0.04	17.65	10.11	18.93	145.12	7.71
16	0.09	50.44	0.03	9.56	6.33	9.85	76.30	14.66
32	0.06	28.04	0.02	5.76	4.34	5.44	43.66	25.62
64	0.06	21.03	0.01	3.82	2.72	3.48	31.12	35.94

Table 2: Run-times in seconds for various stages of the algorithm for 20,000 unknowns for non-uniform distribution.

Number of procs	Tree const.	Nearfield computation	Building inter-action lists	Interpolation	Translation	Anterpolation	Total
32	6.27	636.32	14.80	993.66	5543.02	1051.46	8245.53
64	2.47	329.60	8.79	584.69	3276.59	600.88	4803.02

Table 3: Run-times in seconds for various stages of the algorithm for 1,000,000 unknowns for uniform distribution.

the scatterer. The practical efficiency of our parallel FMM algorithms is experimentally demonstrated. The software can produce results with controllable accuracy and has been validated against analytical solutions and serial software. The software can be extended to analyze scattering from dielectric random rough surfaces. This would involve computing the electric and magnetic currents in each of the two regions. Further, scattering from multiregion surfaces could also be analyzed.

## References

- [1] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: a pedestrian prescription," *IEEE Antennas and Propagation Magazine*, vol. 35, pp. 7–12, 1993.
- [2] C. C. Lu and W. C. Chew, "Fast algorithm for solving hybrid integral equations," *IEE Proceedings. H, Microwaves, Antennas and Propagation*, vol. 140, pp. 455–460, 1992.
- [3] C. C. Lu and W. C. Chew, "A multilevel algorithm for solving boundary integral wave scattering," *Microwave and Optical Technology Letters*, vol. 7, pp. 466–470, 1994.
- [4] C. C. Lu and W. C. Chew, "Fast far-field approximation for calculating the RCS of large objects," *Microwave and Optical Technology Letters*, vol. 8, pp. 238–241, 1995.
- [5] B. Dembart and E. Yip, "The accuracy of fast multipole methods for maxwell's equations," *IEEE Computational Science and Engineering*, vol. 5, pp. 48–56, 1998.
- [6] V. Jandhyala, B. Shanker, E. Michielssen, and W. Chew, "Fast algorithm for the analysis of scattering by dielectric rough surfaces," *Journal of Optical Society of America*, vol. 15, pp. 1877–1885, 1998.

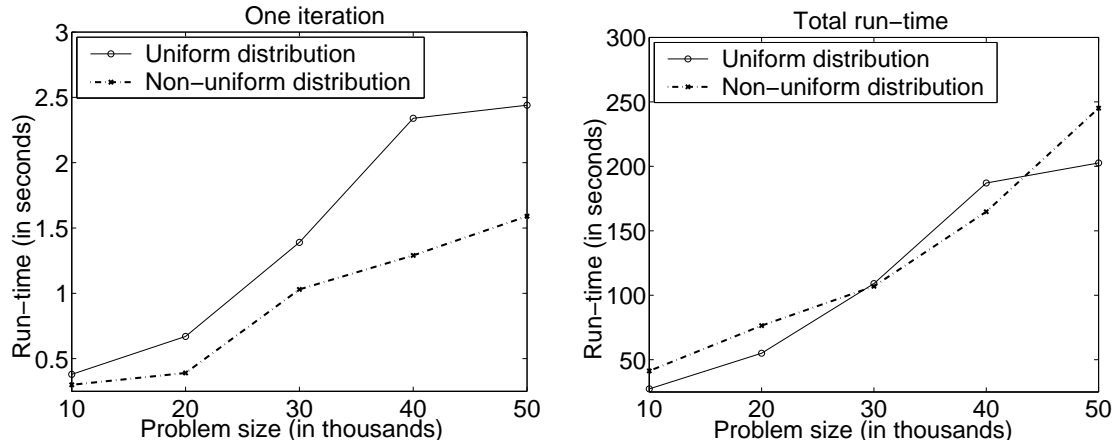


Figure 3: Run-time as a function of the problem size on 16 processors for (i) one iteration of farfield computation and (ii) total application.

- [7] V. Jandhyala, E. Michielssen, B. Shanker, and W. Chew, "A combined steepest descent fast multipole algorithm for the fast analysis of three-dimensional scattering by rough surfaces," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 36, pp. 738–748, 1998.
- [8] V. Jandhyala, D. Sengupta, E. M. E, B. Shanker, and G. Stillman, "Two-dimensional rough surface couplers for broadband quantum-well infrared photodetectors," *Applied Physics Letters*, vol. 73, pp. 3495–3497, 1998.
- [9] K. Aygün, B. Shanker, A. A. Ergin, and E. Michielssen, "A two-level plane wave time domain algorithm for fast analysis of EMC/EMI problems," *IEEE Transactions on EMC*, vol. 44, pp. 152–164, 2002.
- [10] W. C. Chew, J. M. Jin, E. Michielssen, and J. Song, eds., *Fast and Efficient Algorithms for Computational Electromagnetics*. New York, NY: Artech House, 2001.
- [11] N. Geng, A. Sullivan, and L. Carin, "Multilevel fast-multipole algorithm for scattering from conducting targets above or embedded in a lossy half space," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 38, pp. 1561–1573, 2000.
- [12] N. Geng, A. Sullivan, and L. Carin, "Fast multipole method for scattering from an arbitrary PEC target above or buried in a lossy half space," *IEEE Transactions on Antennas and Propagation*, vol. 49, pp. 740–748, 2001.
- [13] V. Rokhlin, "Rapid solution of integral equations of classical potential theory," *Journal of Computational Physics*, vol. 60, pp. 187–207, 1985.
- [14] L. Greengard, *The rapid evaluation of potential fields in particle systems*. Cambridge, MA: MIT Press, 1988.
- [15] M. S. Warren and J. K. Salmon, "Astrophysical N-body simulations using hierarchical tree data structures," in *Proceedings of Supercomputing*, pp. 570–576, 1992.

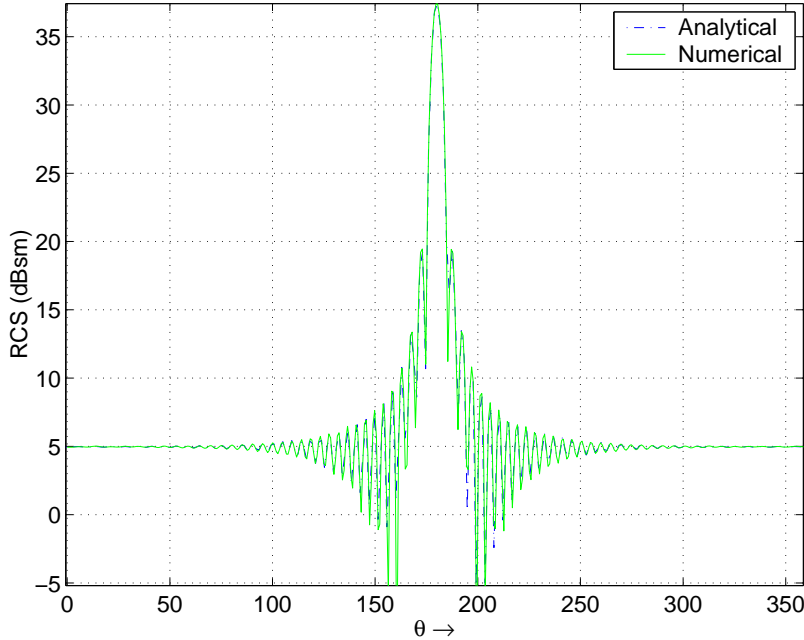


Figure 4: Comparison between analytically and numerically obtained RCS data for a sphere of radius  $1.0m$  discretized using 518,310 unknowns.

- [16] M. Warren and J. Salmon, “A parallel hashed oct-tree N-body algorithm,” in *Proceedings of Supercomputing*, pp. 1–12, 1993.
- [17] J. Singh, C. Holt, T. Tosuka, A. Gupta, and J. Hennessy, “Load balancing and data locality in adaptive hierarchical N-body methods: Barnes-hut, fast multipole and radiosity,” *Journal of Parallel and Distributed Computing*, vol. 27, pp. 118–141, 1995.
- [18] A. Grama, V. Kumar, and A. Sameh, “Parallel hierarchical solvers and preconditioners for boundary element method,” *SIAM Journal on Scientific Computing*, vol. 20, pp. 337–358, 1998.
- [19] P. Liu and S. Bhatt, “Experiences with parallel N-body simulation,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, pp. 1306–1323, 2000.
- [20] S. H. Teng, “Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation,” *SIAM Journal on Scientific Computing*, vol. 19, pp. 635–656, 1998.
- [21] H. Schwichtenberg, G. Winter, and H. Wallmeier, “Acceleration of molecular mechanic simulation by parallelization and fast multipole techniques,” *Parallel Computing*, vol. 25, pp. 535–546, 1999.
- [22] Y. Yuan and P. Banerjee, “A parallel implementation of a fast multipole based 3-D capacitance extraction program on distributed memory multicomputers,” *Journal of Parallel and Distributed Computing*, vol. 61, pp. 1751–1774, 2001.
- [23] J. J. Ottusch, M. A. Stalzer, J. L. Visher, and S. M. Wandzura, “Scalable electromagnetic scattering calculations on the SGI Origin 2000,” in *Proceedings of Supercomputing Conference*, 1999.

- [24] J. W. Manke, B. Dembart, M. A. Epton, and E. L. Yip, "A parallel fast multipole solver for the method of moments in computational electromagnetics," in *Proceedings of the SIAM Conference on Parallel Processing for Scientific Computing*, 1999.
- [25] K. Donepudi, J. Jin, S. Velamparambil, J. Song, and W. Chew, "A higher-order parallelized multilevel fast multipole algorithm for 3-d scattering," *IEEE Transactions on Antennas and Propagation*, vol. 49, pp. 1069–1078, 2001.
- [26] F. Sevilgen, S. Aluru, and N. Futamura, "A provably optimal, distribution-independent parallel fast multipole method," in *Proceedings of International Parallel and Distributed Processing Symposium*, pp. 77–84, IEEE Computer Society Press, 2000.
- [27] P. B. Callahan and S. R. Kosaraju, "A decomposition of multidimensional point sets with applications to  $k$ -nearest neighbors and  $n$ -body potential fields," *Journal of the ACM*, vol. 42, pp. 67–90, 1995.
- [28] S. M. Rao and D. R. Wilton, "Transient scattering by conducting surfaces of arbitrary shape," *IEEE Transactions on Antennas and Propagation*, vol. 39, pp. 56–61, 1991.
- [29] B. Shanker, K. Aygün, N. Gres, and E. Michielssen, "Fast integral equation based analysis of transient electromagnetic scattering from three-dimensional inhomogeneous lossy dielectric objects," in *Proceedings of the IEEE Antennas and Propagation Society*, vol. 2, pp. 532–535, 2001.
- [30] S. Aluru and F. Sevilgen, "Dynamic compressed hyperoctrees with application to the N-body problem," in *Proceedings of the Foundations of Software Technology and Theoretical Computer Science*, vol. 1738, pp. 21–33, Springer-Verlag Lecture Notes in Computer Science, 1999.
- [31] R. Jakob-Chien and B. K. Alpert, "A fast spherical filter with uniform resolution," *Journal of Computational Physics*, vol. 136, pp. 580–584, 1997.
- [32] B. Shanker, A. A. Ergin, M. Lu, and E. Michielssen, "Multilevel plane wave time domain algorithm for the analysis of transient electromagnetic scattering," tech. rep., University of Illinois at Urbana, 2001. To appear in *IEEE Transactions on Antennas and Propagation*.
- [33] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," tech. rep., IBM.