Large Scale Computations in Electromagnetics Using Fast Multipole Method

Sanjay Velamparambil Jiming Song* Weng Cho Chew

Center for Computational Electromagnetics
Dept. of Electrical and Computer Engineering
University of Illinois
Urbana, IL-61801
E-mail: song@sunchew.ece.uiuc.edu

Abstract

In recent years, the Multilevel Fast Multipole Algorithm (MLFMA) has been developed into one of the most powerful techniques for accelerating the iterative solution of integral equations of electromagnetics. It has been shown that MLFMA reduces the computational complexity of a matrix-vector multiply of a dense matrix from $O(N^2)$to $O(N \log N)$ or $O(N)$, where N is the number of unknowns. We have recently developed an implementation of MLFMA called the Fast Illinois Solver Code (FISC) for multiprocessor shared memory computers, which has been distributed to more than 400 copies to government and industrial users. In an attempt to extend the range of problems that can be solved using MLFMA we have also developed an application independent, distributed memory MLFMA kernel, called **ScaleME** *using MPI.*

In this paper, we shall summarize the characteristic features which distinguishes MLFMA from its static counterpart, such as work required for each level, the size of multipole expansions and interpolation/filtering operations, and their influence in the parallel algorithm design. We shall next discuss major issues in the parallelization which are unique to the dynamic MLFMA, such as reducing the memory requirements for translation operators and the reduction of replicated geometric data structures. We shall also briefly discuss the load balancing strategies. Finally, we shall present some representative numerical results from a ScaleME accelerated electromagnetic scattering code, including a simulation involving 4 million unknowns and that of the radar cross-section computation of a full scale air-craft on a Beowulf class cluster. We also give a brief overview of the sequential implementation of dynamic MLFMA and discuss its major achievements, including the scattering from a sphere of 120 wavelength diameter, modeled with 9,633,792 unknowns and a VFY218 at 8 GHz with 9,990,918 unknowns. The equivalent dense matrix system would have been impossible to solve using classical methods on existing computers.

1. Introduction

Integral equation methods are widely used for the solution of electromagnetic scattering problems. In this approach, the problem is first formulated in terms of an appropriate integral equation and then reduced to a system of linear equations using the Method of Moments (MoM). The matrix for the linear system is dense and requires $O(N^2)$ memory storage, where N is the number of unknowns. The resulting linear system is then solved by either a direct method such as LU-decomposition or by an iterative method such as the Conjugate Gradient Method (CGM). It is well known that the number of operations necessary for the direct method is $O(N^3)$ and hence, the maximum size that can be solved is limited to a few hundred thousand unknowns even with modern supercomputers. On the other hand, iterative methods require the application of the coefficient matrix to a sequence of vectors in the solution process. Each of these evaluations requires $O(N^2)$ operations, making it prohibitively expensive for large problems.

In recent years, a number of techniques have been proposed for the rapid application of the coefficient matrix to a given vector. Among these, the Fast Multipole Method [1] and its multilevel or recursive variant the Multilevel Fast Multipole Algorithm (MLFMA) [2] have established themselves as the most powerful ones. It has been shown that MLFMA reduces the computational complexity of a matrix-vector multiply to $O(N \log N)$. Using MLFMA accelerated MoM codes, researchers have solved very large scale problems [2, 3, 4].

We have developed two versions of dynamic MLFMA: FISC for multiprocessor shared memory computers and **ScaleME** for distributed memory MLFMA kernel using the Message Passing Interface (MPI) [5]. FISC (Fast Illinois Solver Code) [6, 7], co-developed by the Center for Computational Electromagnetics, University of Illinois and SAIC-DEMACO, is designed to compute RCS of a target described by a triangular facet file. The problem is formulated by the method of moments (MoM),where the RWG (Rao, Wilton, and Glisson) [8] basis functions are used. The resultant matrix equation is solved iteratively by the conjugate gradient (CG) method. The MLFMA is used to speed up the matrix-vector multiply in CG. Both complexities for the CPU time per iteration and memory requirements are of $O(N\log N)$. Since March of 1997, more than 400 copies of FISC have been distributed to government and industrial users.

FISC was parallelized for multiprocessor shared memory computers. Recently, in an attempt to extend applications, the authors have developed a new parallel version of dynamic MLFMA. This new portable implementation of the dynamic MLFMA is called the **ScaleME** for *Scale*able *M*ultipole *E*ngine. **ScaleME** is implemented as a library of user-callable functions and is targeted at distributed memory machines and networked cluster of workstations. In order to achieve portability over various multicomputers, MPI was used for communication. We have reported some preliminary results in [9] and a more extensive set of results, specifically for electromagnetic scattering problems, has been reported in [3, 10]. Further details on this library have been reported in [11].

The objective of the present paper is briefly to discuss some of the unique characteristics of the dynamic MLFMA, compared to its static counterpart, and their influence on the parallel algorithm design. These issues include, but not limited to, the work required for each level, the size of multipole expansions and interpolation/filtering operations, scaling of translation matrices, and domain decomposition. We shall discuss some of the techniques we have developed to handle these issues. Further details on these topics have been reported in [12].

The paper is organized as follows: in the next section we shall briefly discuss the dynamic MLFMA

and its unique characteristics. Then, we shall discuss the construction of the tree, the domain decomposition algorithms, the scaling of translation matrices associated with the MLFMA and techniques for solving this issue, and a technique for load balancing. Finally, in Section 4 we shall report some numerical results demonstrating the performance for some standard electromagnetic problems using both **ScaleME** and FISC.

2. A Brief Review of Electrodynamic MLFMA

MLFMA for electromagnetic scattering problems is essentially an extension of that for scalar Helmholtz equation to vector problems. Although it has been developed for both two and three dimensional electromagnetic problems, in this paper, we shall restrict our discussion to the three dimensional algorithm. Furthermore, since our objective is to solve integral equations describing various scattering problems, our discussion will implicitly assume that we are using MoM based technique for their discretization.

We begin by noting that in MLFMA, we are dealing with the interaction between a set of "sources" and a set of "receivers". These two concepts can be best explained with respect to the MoM solution of an operator equation. Consider the integral equation

$$f(\mathbf{x}) = g(\mathbf{x}) + \int_G K(\mathbf{x}, \mathbf{x}') f(\mathbf{x}')\, d\mathbf{x}' \tag{1}$$

to be solved using the MoM. In the above equation, $D \subset \mathbb{R}^3$, $K(\mathbf{x}, \mathbf{x}')$ is the kernel of the equation, $g(\mathbf{x})$ is a known function and $f(\mathbf{x})$ is the unknown to be determined. In order to do this, we approximate the unknown function

$$f(\mathbf{x}) = \sum_{i=1}^{N} f_i \phi_i(\mathbf{x})$$

where $\phi_i(\mathbf{x}), i = 1, 2, \cdots, N$ are known, linearly independent functions called the *basis functions* and f_i are unknown coefficients to be determined. By defining a suitable innerproduct, $\langle , \rangle$ and by choosing a set of *testing* functions $\{t_i\}, i = 1, 2, \cdots, N$, we reduce the integral equation to the form

$$Ax = b, \qquad A \in \mathbb{C}^{N \times N}, \quad x, b \in \mathbb{C}^N$$

where $x = (f_1, f_2, \cdots, f_N)$, $b_i = \langle t_i, g \rangle$, and

$$a_{ij} = \begin{cases} \langle t_i, f \rangle - \langle t_i, \psi_j \rangle & \text{if } i = j \\ -\langle t_i, \psi_j \rangle & \text{if } i \neq j \end{cases} \tag{2}$$

where

$$\psi_j(\mathbf{x}) = \int_G K(\mathbf{x}, \mathbf{x}') \phi_j(\mathbf{x}')\, d\mathbf{x}'.$$

We stipulate that the kernel $K(\mathbf{x}, \mathbf{x}')$ is such that the function $\psi_i(\mathbf{x})$ satisfies the Helmholtz equation

$$\nabla^2 \psi_i + k^2 \psi_i = 0 \tag{3}$$

outside a sphere containing the support of ϕ_i.

For our purposes, we shall regard $\{f_i\}$ as the *sources* and the functions $\{t_i\}$ as the *receivers*. The motivation behind this can be illustrated by considering a specific example.

Let $K(\mathbf{x}, \mathbf{x}') = \frac{e^{ik|\mathbf{x}-\mathbf{b}'|}}{k|\mathbf{x}-\mathbf{b}'|}$ and let $D \subset \mathbb{R}^3$. Then $f(\mathbf{x}), \mathbf{x} \in D$ can be interpreted as a continuous distribution of "charge" radiating a field ψ_i. Similarly, the functions $\{t_i\}$ can be interpreted as taking a "weighted" measurement of the fields radiated by the sources – resembling a receiver.

The MLFMA is based on the following key representation: When the geometrical supports of the source ϕ_j and the receiver t_i are "well separated", their interaction $a_{ij} = \langle t_i, K\phi_j \rangle$ can be expressed as [11]

$$a_{ij} \approx \int_{\operatorname{supp} t_i} d\mathbf{r}\, t_i(\mathbf{r}) \frac{i}{4\pi} \int_{S_0} \nu_n(\hat{s}) F_i(\hat{s}) \cdot e^{ik(\mathbf{r}-\mathbf{c}_3)\cdot\hat{s}}\, d\omega_s$$

with

$$\begin{aligned} F_i(\hat{s}) &= \lim_{r\to\infty} (kr)^{\frac{d-1}{2}} \psi(\mathbf{c}_1 + r\hat{s}) e^{-ikr} \\ \text{and} \quad \nu_n(\hat{s}) &= \sum_{m=0}^{n} i^m (2m+1) P_m(\hat{s}.\hat{s}_{13}) h_m(k|\mathbf{c}_2 - \mathbf{c}_1|) \end{aligned}$$

where $\mathbf{c}_1 \in \mathbb{R}^3$ is a point "close" to the expansion function ϕ_j, called the center of multipole expansion , $\mathbf{c}_2$ is a point close to t_i called the center of the local expansion, and S_0 is the unit sphere in $\mathbb{R}^3$. It is obvious that in any numerical implementation, the integration over the unit sphere will be done using an appropriate quadrature rule.

The important thing to be noted is that the source and receiver points are now separated, so that one can combine the "far fields" $F_i(\hat{s})$ of several basis functions to form a single representation and then evaluated once. Thus, the general strategy of a MLFMA is that of clustering basis functions at various spatial lengths and computing the interactions with the testing functions from sufficiently distant clusters using far field representations. For nearby interactions, direct computation is used. Note that in the static fast multipole method, the interactions are computed using multipole expansions whereas in the dynamic case, far field representations are used, which lead to the diagonalization of the translation operators.

In order to do this, the basis functions are first enclosed in a cube. Then using a recursive subdivision, a hierarchy of meshes, dividing the cube into smaller ones, is created. A tree structure is then imposed on the non-empty cubes in this hierarchy.

The basic algorithm for matrix-vector multiply is broken down into two sweeps: the first sweep consists of constructing the far field patterns of each non-empty box at every level. At the finest level, this is done by combining the far field patterns of all the basis functions belonging to the given box. At every other level (except the two coarsest levels) the far field pattern of a given box is obtained by combining the far field patterns of its children. The second sweep consists of constructing local fields of each box at every level, representing the field due to all the cubes that are considered to be "well separated".

A most important point to be noted is that as the boxes grow in size, as we ascend the tree, the number of multipole coefficients or the number of samples of the far field pattern over the unit sphere, increases. Thus, combining the far field patterns of the child boxes to form that of the parent requires some form of interpolation. Similarly, during the second sweep, while inheriting the local fields of the parent, the

number of samples needs to be reduced by filtering out the higher order fields. We refer the reader to [13, 14, 15] more detailed descriptions. This is the most crucial difference of the dynamic MLFMA from its static counter part. For the latter, the number of multipoles used at every level is the same.

3. Parallelization of MLFMA on Distributed Memory Computers

In this section, we shall summarize major issues in the parallelization which are unique to the dynamic MLFMA, such as reducing the memory requirements for translation operators and the reduction of replicated geometric data structures. We shall also briefly discuss the load balancing strategies. Further details on these topics have been reported in [12].

3.1. Tree Construction and Domain Decomposition Issues

Several approaches have been discussed in the literature for tree construction [16, 17] for particle simulation applications. However, in the case of integral equation solvers, there exists two important geometrical properties which influence the choice of the tree construction algorithm.

The first property of the geometric data arising from the discretisation of integral equations is that they are stationary in time. This property can be used to generate a good load balancing strategy during the initialization time itself.

However, the second property is not so helpful. It refers to the fact that most discretisation procedures use basis functions with non-trivial support. That is, the sources are no longer located at a single point, but spread over a region. An important example of such a discretisation is the Galerkin's method with the RWG functions for solving electromagnetic scattering problems [8]. The RWG functions have support over two triangular facets. Although this problem is not unique to the electromagnetic scattering problems, it is particularly important in this case owing to the recent developments in higher order approximation techniques [18].

Such discretisations make the so called "connection matrix" non-diagonal. The connection matrix relates the basis functions to geometric description of the object. For instance, in the case of RWG basis functions, the connection matrix relates each basis function with the indices of the triangular facets on which it is defined. Therefore, any decomposition of basis functions across multiple processors will have to ensure *completeness* of local data. That is, one needs to make sure that if a particular basis function is assigned to a processor, all the geometric data associated with that basis function is also available at the same processor. To give an example, with RWG basis functions, if a particular basis function is assigned to a processor, we also have to keep the two facets associated with that basis function in the same processor. But this in turn implies that we also need to keep the vertices and the edges of the facets in the same processor. Such a scheme would ensure that the processors need not communicate with each other to get relevant geometric data. We call such an assignment as a *locally complete decomposition*. Note that a locally complete decomposition may still have replication to some extent.

One way to handle this problem is to simply replicate the full geometric structures in every processor. However, memory for geometric data structures is $O(N)$, where N is the number of unknowns and a full replication has the undesirable scaling of $O(Np)$ where p is the total number of processors. Nevertheless, for small to medium scale problems, ranging from a few thousand unknowns to a few hundreds of thousands, this approach is very practical.

Obviously, for very large scale problems such a simplistic approach is unacceptable and any serious implementation of MLFMA on distributed memory computers will have to support an appropriate domain decomposition.

Fortunately, we can construct a good decomposition of geometric data which requires $O(N)$ memory independent of the number of processors, provided we *already have a distributed tree*. To see this, we note that in order to compute the matrix-vector products, we need two types of information for every box. First of these refers to the radiation and receiving patterns of every basis function in any given box. This implies that we need to store the connectivity information for all the local basis function. However, this information is not sufficient. This is because, to compute the near interactions, we need to have the geometric data and connectivity information for every basis function in the near interactions lists of local boxes. Therefore, once we have the tree, all we need to store is the connectivity and geometric information of every basis function in the near lists of local boxes. Since the number of direct interactions is $O(N)$, it follows that the total storage requirement still remains $O(N)$. However, note that the actual storage will be more than that required for one copy of the geometric data structures because of a few replications.

To exploit this, in **ScaleME** , we have developed a two stage initialization procedure in the first stage of which the distributed tree is constructed. The application program can then use the information provided by **ScaleME** to decompose the geometric data structures.

3.2. Scaling of Translation Matrices

In the dynamic MLFMA, the speedup in computing the matrix-vector products is achieved through the use of *diagonalized translation operators*. Usually, in any practical implementation of MLFMA, the diagonalized translation operators are pre-computed at the setup stage and stored. In the straight forward implementation of MLFMA for distributed memory computers using message passing paradigm, the translation operators are replicated in each processor in order to reduce communication costs and to accelerate the matrix-vector multiply. However, this requires an $O(N)$ storage in each processor. Thus the memory requirements for translation operators scale as $O(Np)$ where p is the number of processors. This is unacceptable for large scale problems.

A naive approach to solve this problem is to compute the translation operators "on the fly". In other words, compute them as they are required. However, it increases the overall computational complexity of MLFMA to at least $O(N^{3/2})$ from $O(N \log N)$.

In order to solve this problem, we have developed a new *compressed* representation for the translation matrices which can be evaluated rapidly as and when they are required [19]. This new representation is based on the observation that for a given $\hat{s}_0$, the product $\hat{s} \cdot \hat{s}_0 \in [-1, 1]$ and that the function $\nu_L(\hat{s})$ is thus a polynomial of degree $L - 1$ in the interval $[-1, 1]$. We use a specially designed one dimensional fast multipole algorithm to construct a fast, polynomial representation which requires only $O(\sqrt{N})$ storage and $O(N)$ evaluation time, thus bringing down the scaling of storage requirements of the translation operator to $O(p\sqrt{N})$ (See [19] for more details of this work).

However, the introduction of the compressed translation operators increases the time for matrix-vector products significantly. To see this, we note that the translation phase is driven by the "receiver" box and its interaction list [11]. This would require the evaluation of a given translation matrix more than once in a single matrix-vector product computation. Therefore, even though the compressed translation operators have a lower time complexity, the overall computation time increases.

In order to avoid this, we need to rearrange the translation phase in such a way that each translation matrix is evaluated once and only once during each matrix-vector computation. For this, we create a list of ordered pairs of boxes for each translation operator. See Figure 1 for a pictorial illustration of the interaction list of a translation matrix. Once this is done, the translation phase can be rearranged as follows:

```
for l = 2 to maxlev do
   for each translation operator ν_n(r) at this level do
      for each ordered pair (b, b') ∈ Interaction_List of ν_n(r) do
         Translate the far field of b' to b';
      end for
   end for
end for
```

On a single processor implementation, this scheme ensures that each translation operator is accessed or computed only once. However, in a distributed memory implementation it may happen that some of the boxes in the interaction list of a translation operator may not be locally available for accessing the radiation pattern $F_{b'}(\hat{s})$. Hence, a direct implementation of the above scheme cannot be used.

However, this problem can be solved by using the concept of *ghost boxes* [20]. Ghost boxes are dummy boxes where incoming radiation patterns are stored. For instance, let a box with Morton key k_1 exist in processor p_1 and its radiation pattern is needed in processor p_2. We establish a dummy box in processor p_2 with a key k_1 and reserve sufficient memory for holding the incoming radiation pattern. Each processor keeps several such ghost boxes for each level. Now, as soon as the radiation patterns arrive, the translations are done using the translator driven algorithm described earlier.

One of the issues that comes up is the amount of memory required for ghost boxes. It turns out that if we let each processor do a partial synchronization at each level during the translation phase, the memory used by boxes at a given level can be reused for the ghost boxes at the next level. Using this observation, we shall demonstrate in the numerical results section that the memory used by the ghost boxes is much smaller than that would have been used by storing all the translation matrices.

3.3. A Costzone Scheme for Load Balancing

In this section, we describe a static costzone scheme for balancing the CPU load across the processors. The approach we employ here is similar to the one discussed in [21]. We note that, in the case of acoustic and electromagnetic scattering problems, the positions of the basis functions remain the same throughout the simulation. As a consequence, the MLFMA tree used for computing matrix-vector products also remains the same. Thus it is feasible to look for a load balancing scheme which can be incorporated as an inexpensive preprocessing scheme for tree construction, since it is a one-time affair.

For simplicity, we ignore the communication overhead associated with the computations. We shall also assume that the cost associated in transmitting data across any two processors is the same. Although this assumption is not always true, it simplifies the situation considerably.

We begin by mapping each particle to a box at the finest level, thereby assigning a Morton key, not necessarily unique, to each of them. After sorting the particles according to their keys, the finest level of

the MLFMA tree is obtained. We shall keep this sorted list of basis functions and their respective keys in each processor an use it throughout the computations.

Corresponding to each box, we shall define a data structure, `node_work`, which can be represented as a 4-tuple: $($`key, nparticle, first_particle, my_work`$)$, where `key` is the Morton key to the box, `nparticle` is the number of particles in the box, `first_particle` is a pointer to the first particle in the box in the sorted list, and `my_work` is the total work required for the box. The total work required for a given node is estimated through a "simulation" of the MLFMA. That is, the tree is traversed exactly as in an actual matrix-vector multiply, but instead of computing the matrix-vector products, we only estimate the work required for every operation, such as translations, interpolations and filtering, required for each box in the tree. This may be termed as a "dry run" of the algorithm. Note that, in order to do this, we do not need all the details of the tree, or even the memory. It turns out that this "dry run" is quite inexpensive.

An important fact to be noted is that computing the work required for each node is not the same as doing the actual computations. Therefore, under the assumption that the total number of non-empty boxes is $O(N)$, it can be shown that this process has a complexity of $O(N \log N)$, and that the constant associated is insignificant compared to that for the matrix-vector multiply.

Once this dummy tree is constructed, we compute the total work, W, required for each matrix-vector product. The i-th processor is then assigned all the nodes in the tree which contribute from $\frac{i-1}{p}W$ to $\frac{i}{p}W$ segment of the total work, where p is the number of processors. This can be readily done by traversing the tree in post-order and summing up the contributions from each node as the traversal progresses.

4. Numerical Results

As we mentioned earlier, **`ScaleME`** is an application independent kernel which can be retrofitted into independently developed integral equation solvers. We have interfaced it with several such codes developed at the Center for Computational Electromagnetics of University of Illinois. In this section, we report some results from one such code for demonstration purposes.

The application considered is that of electromagnetic scattering from perfectly conducting obstacles. This is a classical and very important problem having a wide variety of applications. The formulation and implementation issues are well documented in the literature and we omit the details here [13, 2, 3]. We present some representative results pertaining to the issues discussed in this paper.

4.1. Single Processor Performance

The first set of results we report is that of comparison of single processor performance of the new code to the FISC. In order to do this, we choose the classical example of scattering from perfectly conducting spheres of various sizes. This comparison was carried out on an SGI Power Challenge series multiprocessor with 2GB RAM, running at 90MHz. During this test, the parameters of **`ScaleME`** , such as number of terms in the multipole expansions and the order of the interpolation, are chosen to match exactly with that of FISC. The results are plotted in Figure 2. It is seen that the **`ScaleME`** is about 23% slower than that of FISC. This is a very good performance since the MLFMA as implemented in FISC is tightly coupled to the MoM code and hence uses several specialized optimizations for efficiency. On the other hand, we see that the memory requirements for **`ScaleME`** is almost the same as that for FISC. This test demonstrates that the single processor performance of **`ScaleME`** is comparable to that of the

best known sequential code.

4.2. Scaling of Translation Matrices

In this section, we present some representative results demonstrating the savings in memory obtained by using the compressed translation operators. The example chosen is a set of spheres of radii varying from 12λ to 80λ where λ is the wavelength. In Table 1, we report the memory required for the full matrices and that required for the compressed translators. Given also are the number of unknowns used in each case. The savings are clearly impressive. In order to give a more concrete picture of the performance of this algorithm, in Table 2, we present the total savings on 16 processors while solving the 80λ problem involving nearly 4 million unknowns.

4.3. Load Balancing

As discussed earlier, we use a static costzone scheme for load balancing. In this section we report two results on the load balance achieved during practical simulations. In the first case, we consider the scattering from a metallic sphere of radius 80λ. This problem involved $3,981,312$ unknowns. We report the *average* time spent in the kernel of ScaleME for each matrix-vector product as a measure of the balance achieved. This problem was solved on 16 processors of NCSA's Orgin 2000. This is shown in Figure 3. In the second example, we computed the bistatic radar cross-section of a full scale aircraft, namely a VFY218, at 500MHz. The aircraft was modeled using 54,990 unknowns. This problem was solved on a Beowulf [22] class cluster, called *Orion*, of PCs running Linux. Each node of this cluster has an 350MHz AMD K6-2 processor and 128MB of RAM. The nodes are connected through a 100Mbps ethernet and a 24 port, Xyplex Network's Megaswitch which gives a maximum throughput of 3.2Gbps. More details on the cluster can be found in [10]. The load distribution is reported in Figure 4. We note that the load distribution across the processors is not as good as that obtained for the spheres. This is due the geometrical asymmetry, with respect to the hierarchical decomposition, of the problem.

4.4. Fast Illinois Solver Code (FISC)

FISC is fine tuned for sequential implementation of dynamic MLFMA and parallelized for shared memory machine like SGI Power Challenge and Origin2000. In Figure 5, we plot the RMS error between Mie series and FISC results for a sphere from 0.75 to 100 λ. The number of unknowns is 588 to 9,633,792 and number of levels in MLFMA is from 2 to 9. It is observed that the error is almost a constant except for the first two points, where the geometry error is dominant. The 9,633,792 unknown model is also used to calculate the RCS for a 120 λ sphere as shown in Figure 6. It uses 32 processors of Origin2000, 26.7 GB of memory, 1.5 hrs. for filling matrix, 13.0 hrs. for 43 iterations in GMRES (restart after 15 iterations) to reach 0.001 residual error, and 3 minutes for 1,800 points of RCS. The RMS error is 0.20 dB for all $1,801$ points for elevation angle ranging from $90°$ to $-90°$ with the incident angle at $90°$, and 0.11 dB for elevation angle ranging from $90°$ to $-30°$. Because of the memory limit, the accuracy setting is lower than the run for 100 λ.

In Figure 7, we plot bistatic RCS of the VFY218 at 8 GHz. The incident direction is from broadside with H-polarization (EL=0°, AZ=90°). This problem used 64 processors of Origin2000, 46 GB of memory, and 19 hours. Numerical results calculated by Xpatch, a high frequency approximation code, are also plotted for comparison.

5. Conclusions

In this paper, we have discussed some of the important issues in the parallelization of the electrodynamic MLFMA using the message passing paradigm. We have discussed the unique characteritics of MLFMA that influence the parallel algorithm design. Techniques for resolving issues such as domain decomposition, scaling of translation matrices, and load balancing were discussed briefly. We have presented some representative performance results demonstrating the success of these algorithms. Using FISC on Origin2000 machines in HPC ASC MSRC and NCSA at the University of Illinois, we have solved solved the scattering problem of a 120λ sphere with 9,633,792 unknowns and VFY218 at 8 GHz with 9,990,918 unknowns. Our results show that there is no significantly error propagation from level to level in MLFMA.

Acknowledgements

This work was supported by the AFOSR under a MURI grant F49620-96-1-0025 and National Science Foundation under grant NSF ECS 99-06651. The computer time was provided by the ASC MSRC and the NCSA at the University of Illinois.

References

[1] V. Rokhlin. Diagonal forms of translation operators for the Helmholtz equation in three dimensions. *Applied and Comp. Harmonic Analysis*, 1:82–93, 1993.

[2] J. M. Song, C. C. Lu, and W. C. Chew. MLFMA for electromagnetic scattering from large complex objects. *IEEE Trans. Ant. Propag.*, 45(10):1488–1493, Oct. 1997.

[3] S. V. Velamparambil, J. M. Song, and W. C. Chew. A Portable Parallel Multilevel Fast Multipole Solver for Scattering from Perfectly Conducting Bodies. *IEEE Antennas Propagat. Symp.*, 1:648–651, July 1999.

[4] J. J. Ottusch, M. A. Stalzer, J. L. Visher, and S. M. Wandzura, Scalable electromagnetic scattering calculations for the SGI Origin 2000. In *IEEE Proceedings SC99*, (Portland, OR), Nov. 1999.

[5] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI: The Complete Reference*. Scientific and Engineering Computation Series. The MIT Press, Cambridge,MA, 1996.

[6] J. M. Song, C. C. Lu, W. C. Chew, and S. W. Lee. Fast Illinois Solver Code. *IEEE Antennas and Propagation Magazine*, 40(3):27–34, June 1998.

[7] J. M. Song and W. C. Chew, The Fast Illinois Solver Code: requirements and scaling properties. *IEEE Computational Science and Engineering*, 5(3):19–23, July-Sept. 1998.

[8] S. M. Rao, D. R. Wilton, and A. W. Glisson. Electromagnetic scattering by surfaces of arbitrary shape. *IEEE Trans. Antennas Propagat.*, 30(3):409–418, May 1982.

[9] S. Velamparambil, J. Song, W. Chew, and K. Gallivan. ScaleME: A portable, scalable multipole engine for electromagnetic and acoustic integral equation solvers. *IEEE Antennas Propagat. Symp.*, 3:1774–1777, 1998.

[10] S. V. Velamparambil, J. E. Schutt-Aine, J. G. Nickel, J. M. Song, and W. C. Chew. Solving large scale electromagnetic problems using a linux cluster and parallel MLFMA. *IEEE Antennas Propagat. Symp.*, 1:636–639, July 1999.

[11] S. Velamparambil, J. Song, and W. C. Chew. ScaleME: A Portable, Distributed Memory Multilevel Fast Mulipole Kernel for Electromagnetic and Acoustic Integral Equation Solvers. Technical Report CCEM-23-99, Center for Computational Electromagnetics, Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1999. (Under the review of SIAM J. Scientific Computing).

[12] S. V. Velamparambil, J. M. Song, and W. C. Chew. On the Parallelization of Dynamic Multilevel Fast Multipole Method on Distributed Memory Computers. *Proc. International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, Maui, Hawaii* , 1999. To be published by the IEEE Computer Society.

[13] J. M. Song and W. C. Chew. Multilevel fast multipole algorithm for solving combined field integral equations of electromagnetic scattering. *Micro. Opt. Tech. Lett.*, 10(1):14–19, Sept. 1995.

[14] B. Dembart and E. Yip. A 3D Fast Multipole Method for Electromagnetics with Multiple Levels. *Applied Computational Electromagnetics Society*, pages 621–628, 1995.

[15] L. Greengard and S. Wandzura. Fast mutlipole methods. *IEEE Computational Science and Engineering*, 5(3), July 1998. Special Issue on Fast Multipole Methods.

[16] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree N-body algorithm. In *Supercomputing '93 Proceedings*, pages 12–21, Washington, DC, 1993. IEEE Comp. Soc. Press.

[17] A. Y. Grama, V. Kumar, and A. Sameh. Scalable parallel formulations of the barnes-hut method for n-body simulations. In *Supercomputing '94 Proceedings*, pages 439–448, Washington, DC, 1994. IEEE Comp. Soc. Press.

[18] K. C. Donepudi, J. M. Jin, S. Velamparambil, J. M. Song, and W. C. Chew. A higher order parallelized multilevel fast multipole algorithm for 3D scattering. *Submitted to IEEE Trans. Antennas and Propagation.*

[19] S. Velamparambil and W. C. Chew. Fast Polynomial Representations for the Diagonal Translation Operators of the Three Dimensional Helmholtz Equation. Technical Report CCEM-17-99, Center for Computational Electromagnetics, Dept. of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 1999. (Under the review of SIAM J. Numerical Analysis).

[20] E. J. Lu and D. I. Okunbor. A massively parallel fast multipole algorithm in three dimensions. In *The Proceedings of Fifth IEEE International Symposium on Parallel and Distributed Processing*, pages 40–48, 1996.

[21] A. Grama, V. Kumar, and A. Sameh. Scalable parallel formulations of the Barnes-Hut method for n-body simulations. *Parallel Computing*, 24(5–6):797–822, June 1998.

[22] <http://www.beowulf.org/>.

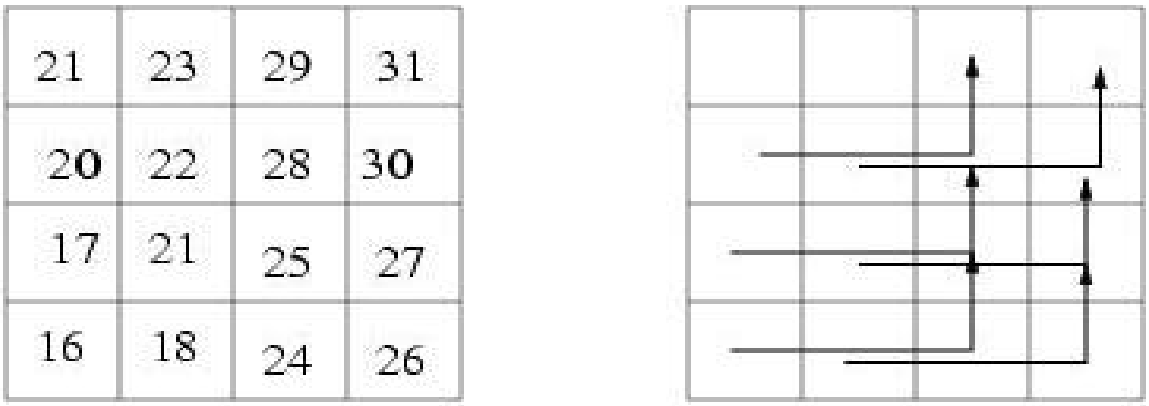
Table 1. Savings in memory obtained by the use of compressed translation operators.

Elect. Size	Num. Unknowns	Memory (MB)	
		Full	Compressed
12	150,528	10.83	0.8
24	602,112	36.54	2.3
48	2,408,448	132.4	5.3
60	3,145,728	201.1	6.6
80	3,981,312	557.5	12.34

Table 2. Memory required at each level for the translation operators for the 80λ sphere problem and the savings obtained using compressed translation operators and ghost boxes. N_s is the number of samples over the unit sphere corresponding to a given level

Level	N_s	Memory (MB)	
		One Matrix	Full
2	170532	1.3	411.13
3	45004	0.343	108.5
4	12172	0.093	29.35
5	3532	27.6(KB)	8.52
Memory required for one processor			557.5
Total Memory for 16 processors(MB)			8,912
Memory for Compressed Operators (MB)			12.34
Total Memory for ghost boxes (MB)			2,048
Savings in Memory (MB)			6,674.6

Figure 1. Illustration of the list of interactions associated with a given translation operator.



Interaction list of a given translator: {(16,25),(17,28),(18,27),(21,30),(20,29),(22,31)}

Figure 2. Comparison of speed and the memory requirements for `TRIMOM+ScaleME` with FISC on a single processor.

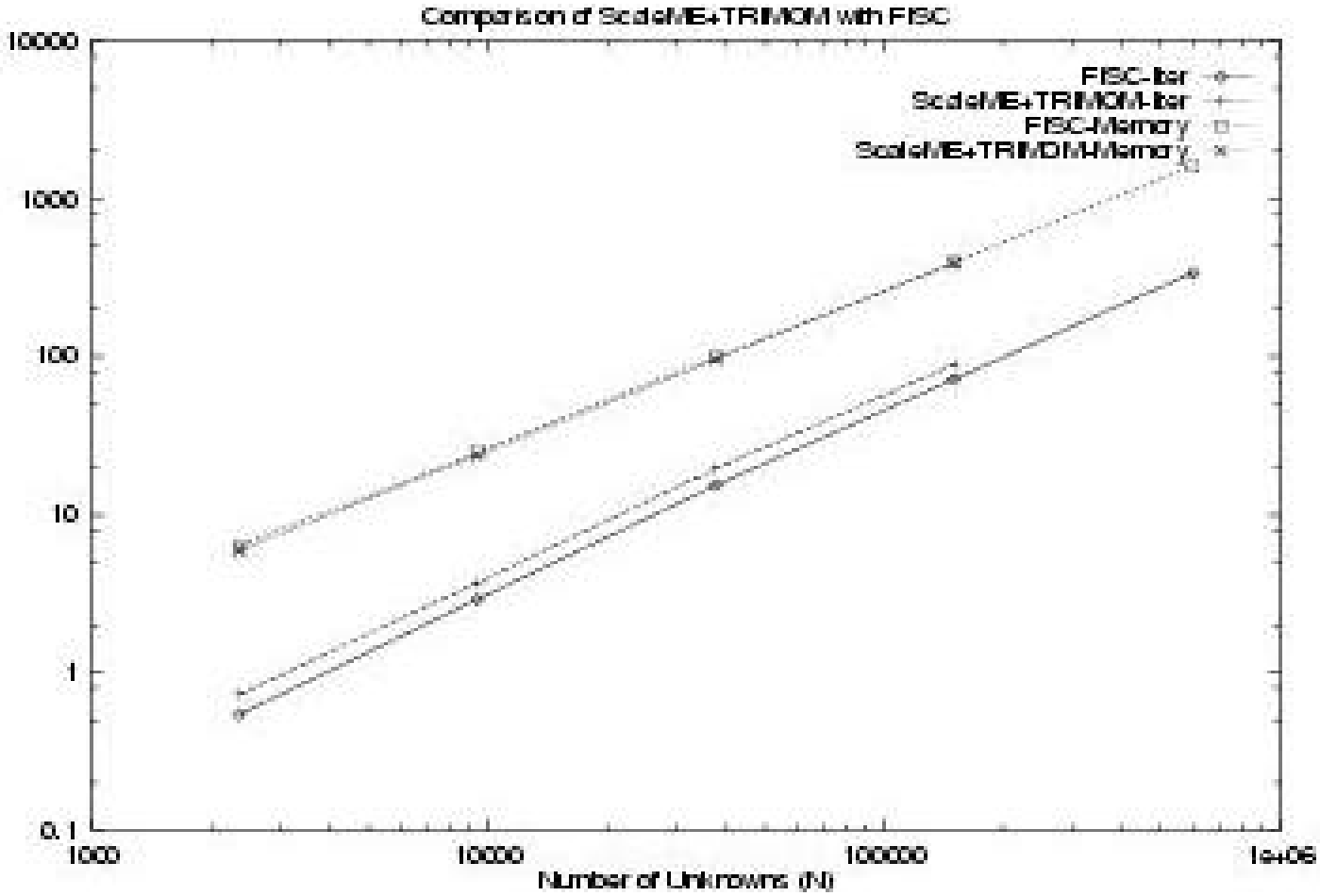


Figure 3. The average time spent in the kernel for each matrix-vector product plotted against the processor Id. We use this as a measure of the load balance achieved.

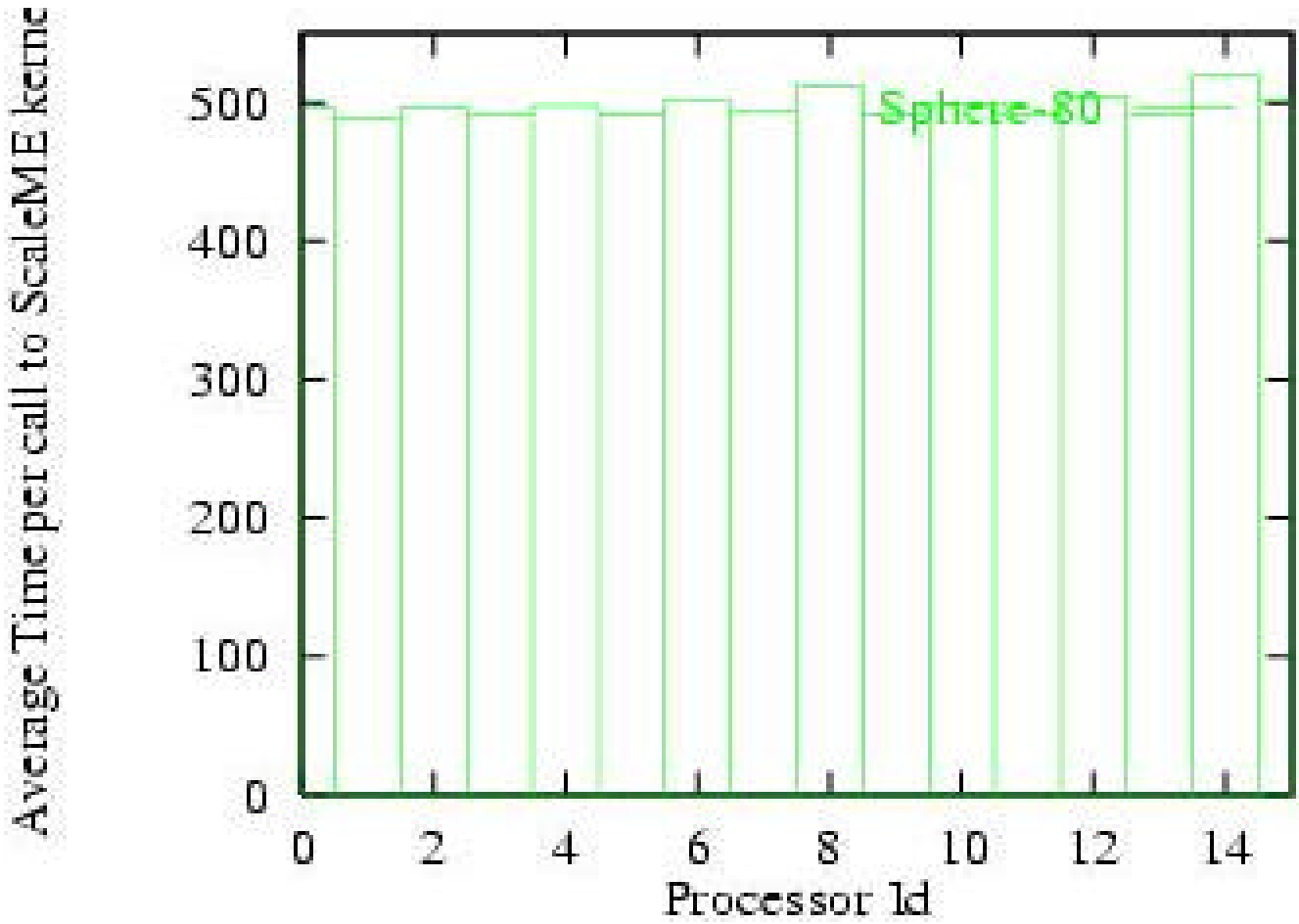


Figure 4. The average time spent in the kernel for each matrix-vector product plotted against the processor Id for the aircraft.

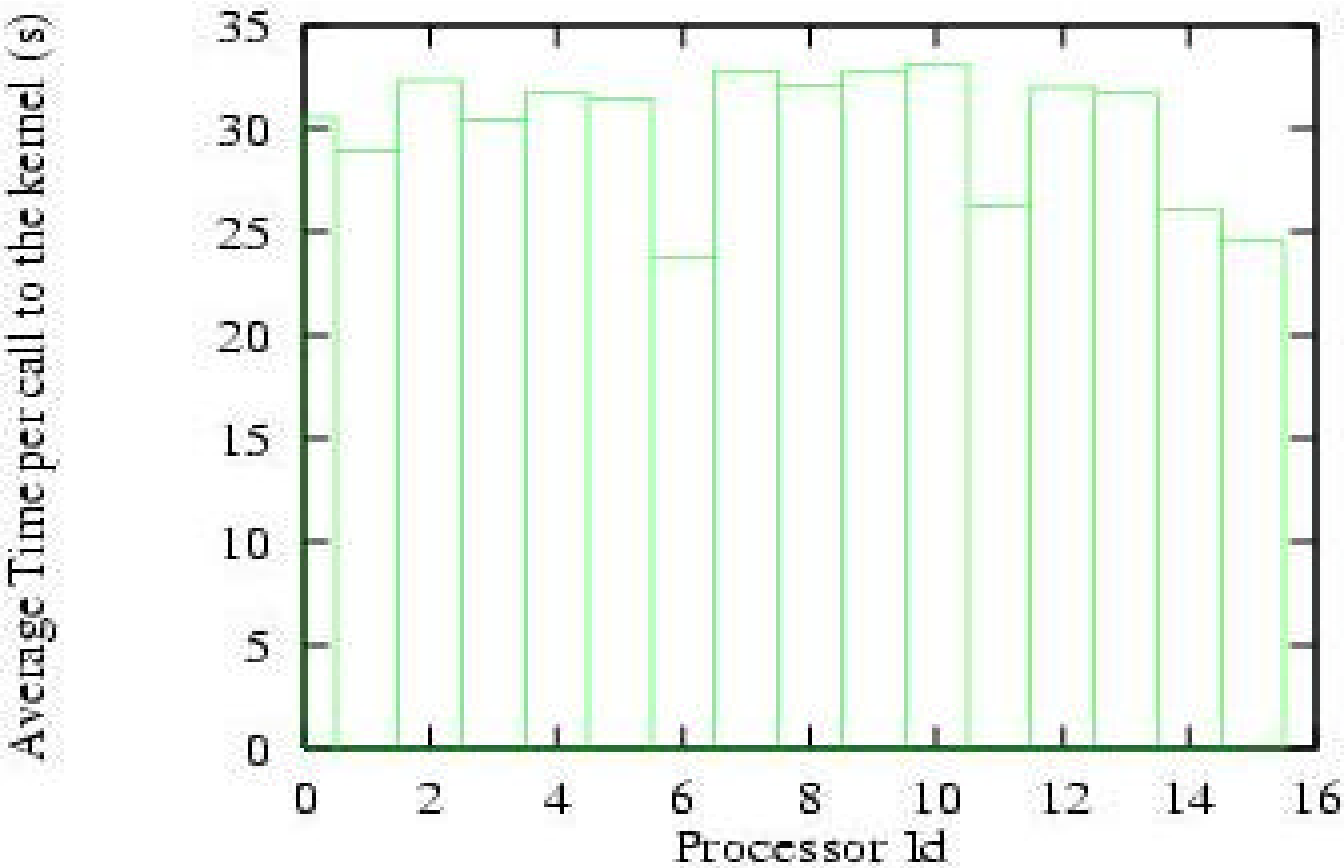


Figure 5. RMS error for a sphere from 0.75 to 100 in diameter. Number of levels in MLFMA is from 2 to 9.

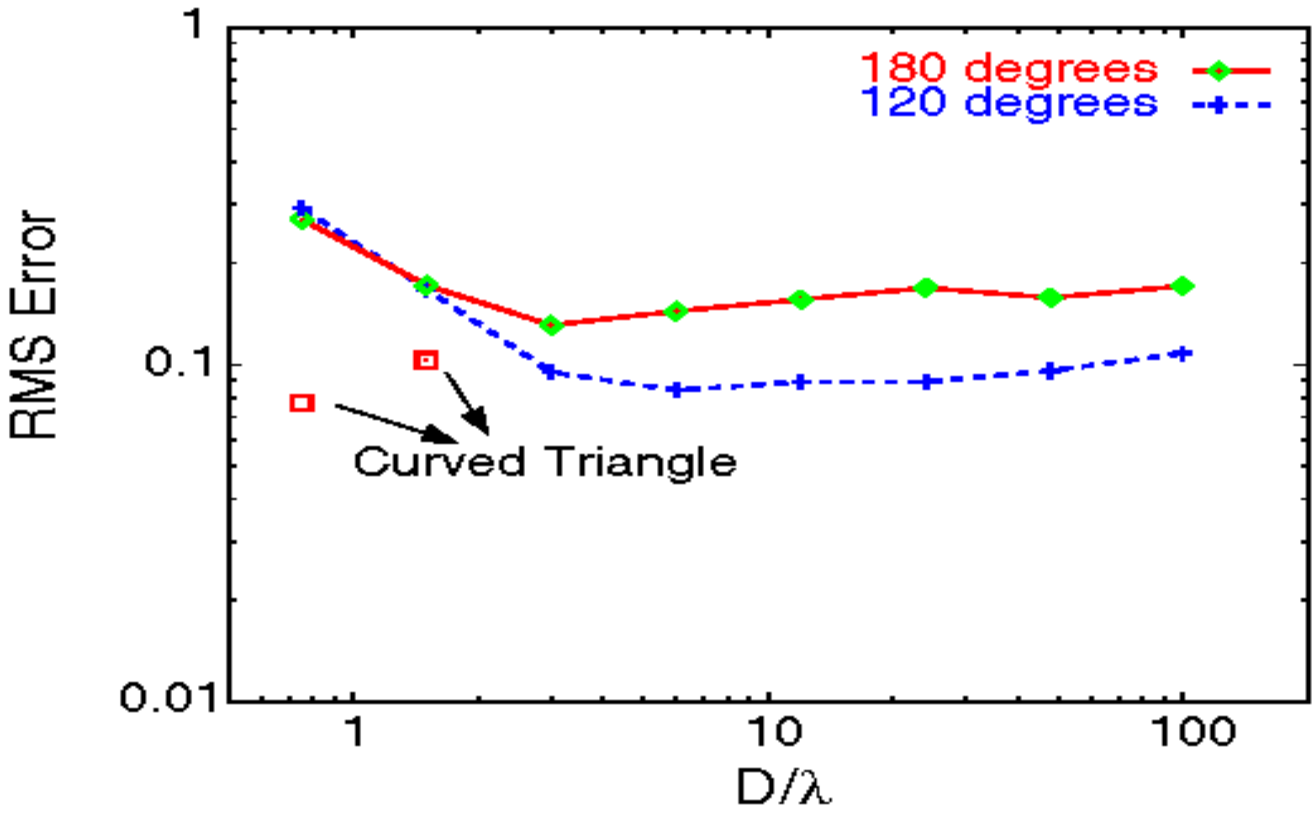


Figure 6. The Mie series and FISC result for a sphere with 120. A total of 9,633,792 unknowns and 9-level MLFMA are used.

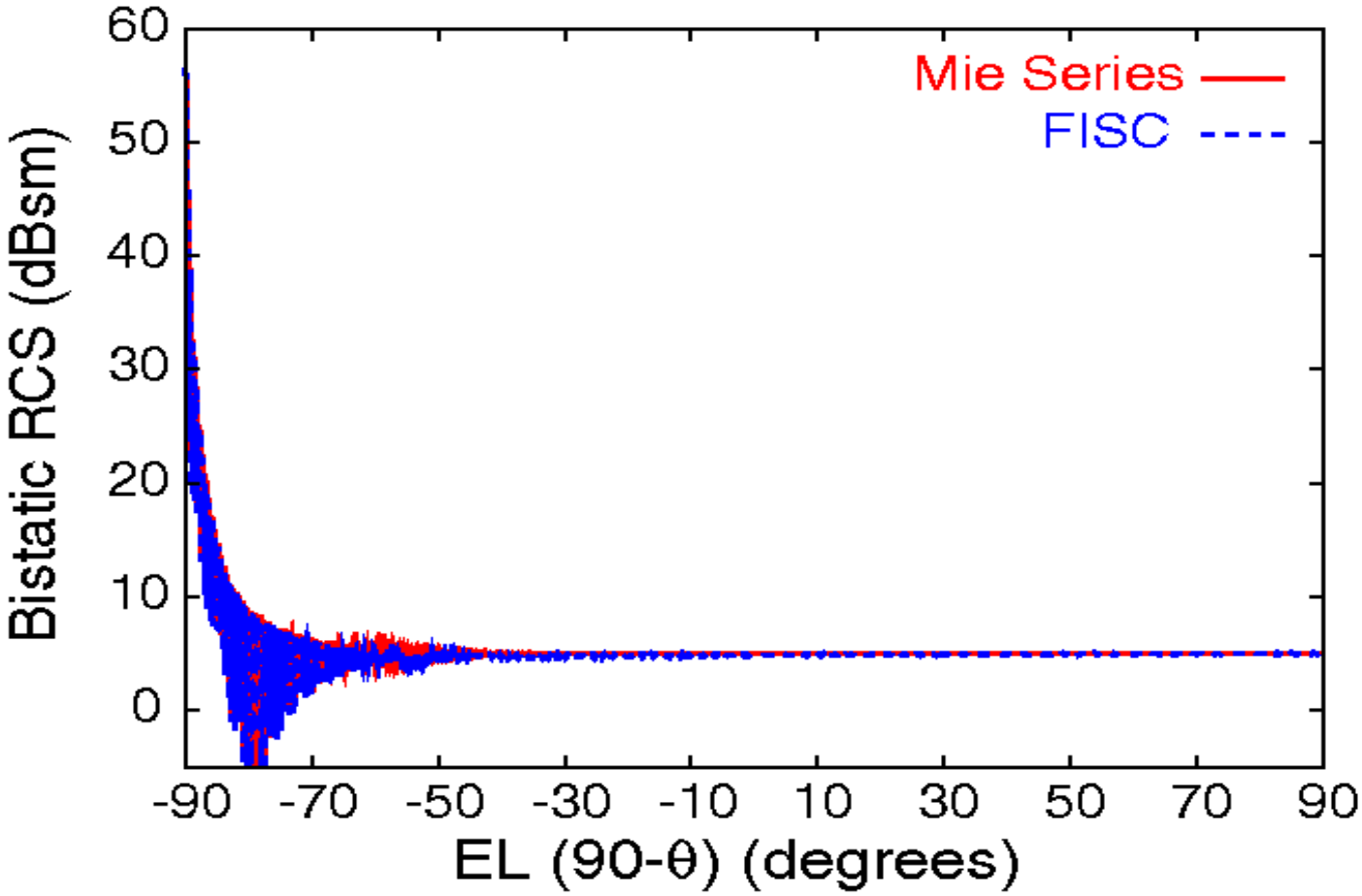
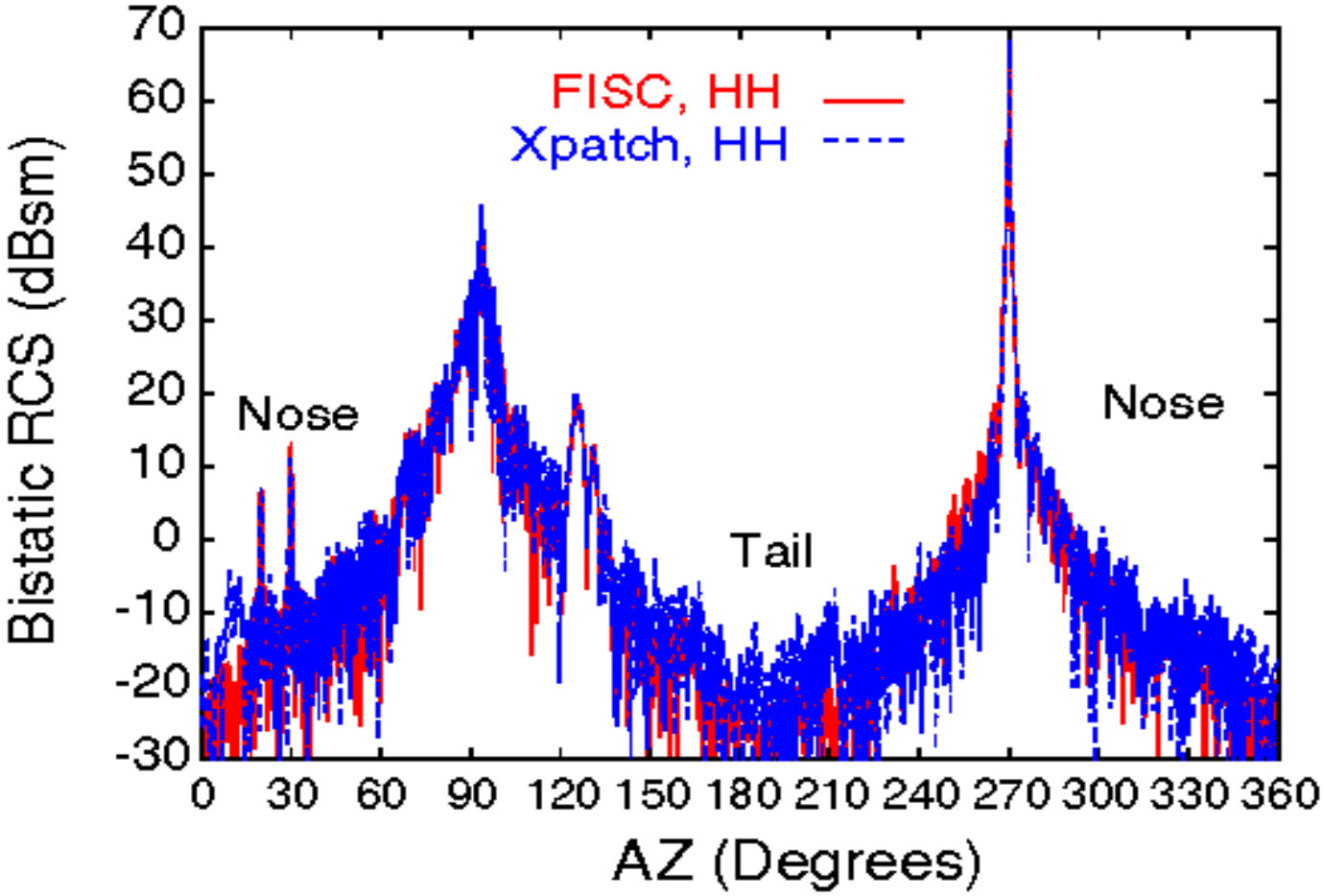



Figure 7. The bistatic RCS of the VFY218 at 8 GHz (H-pol.). A total of 9,990,918 unknowns and 10-level MLFMA are used.