SIAM J. SCI. COMPUT.
Vol. 31, No. 3, pp. 1968–1984

ACCELERATING THE MULTILEVEL FAST MULTIPOLE ALGORITHM WITH THE SPARSE-APPROXIMATE-INVERSE (SAI) PRECONDITIONING*

TAHİR MALAS† AND LEVENT GÜREL†

Abstract. With the help of the multilevel fast multipole algorithm, integral-equation methods can be used to solve real-life electromagnetics problems both accurately and efficiently. Increasing problem dimensions, on the other hand, necessitate effective parallel preconditioners with low setup costs. In this paper, we consider sparse approximate inverses generated from the sparse near-field part of the dense coefficient matrix. In particular, we analyze pattern selection strategies that can make efficient use of the block structure of the near-field matrix, and we propose a load-balancing method to obtain high scalability during the setup. We also present some implementation details, which reduce the computational cost of the setup phase. In conclusion, for the open-surface problems that are modeled by the electric-field integral equation, we have been able to solve ill-conditioned linear systems involving millions of unknowns with moderate computational requirements. For closed-surface problems that can be modeled by the combined-field integral equation, we reduce the solution times significantly compared to the commonly used block-diagonal preconditioner.

Key words. preconditioning, sparse-approximate-inverse preconditioners, integral-equation methods, computational electromagnetics, parallel computation

AMS subject classifications. 31A10, 65F10, 78A45, 78M05, 65Y05

DOI. 10.1137/070711098

1. Introduction.

Iterative solutions of linear systems using Krylov subspace methods make it possible to solve large-scale scientific problems with modest computing requirements [22]. Krylov subspace methods access the system matrix through matrix-vector multiplications. Effective parallelization of both the matrix-vector multiplication and the iterative solvers are possible, allowing even larger systems to be solved with cost-effective parallel computers [21]. However, iterative solvers usually require preconditioning in order to be effective. Most preconditioners use methods similar to direct solution techniques, rendering their parallelization a difficult task. As a result, preconditioning is currently an important bottleneck for the solution of large scientific problems [1].

Computational electromagnetics (CEM) is an active research area, while it is also sufficiently mature to contribute to various industrial applications. For instance, radar cross section computations of arbitrarily shaped three-dimensional targets and complicated antenna design calculations benefit from the highly accurate solutions obtained with the method of moments (MOM). However, MOM produces large and dense linear systems, whose solutions become viable only with fast methods, such as the multilevel fast multipole algorithm (MLFMA) [6]. MLFMA reduces the computational complexity of the matrix-vector product to $\mathcal{O}(n \log n)$, and recent attempts have yielded efficient parallelizations of the method [10, 9, 23].

*Received by the editors December 15, 2007; accepted for publication (in revised form) November 21, 2008; published electronically March 27, 2009. This work was supported by the Scientific and Technical Research Council of Turkey (TUBITAK) under research grants 105E172 and 107E136, the Turkish Academy of Sciences in the framework of the Young Scientist Award Program (LG/TUBA-GEBIP/2002-1-12), and contracts from ASELSAN and SSM.

http://www.siam.org/journals/sisc/31-3/71109.html

†Department of Electrical and Electronics Engineering, Bilkent University, TR-06800, Bilkent, Ankara, Turkey (tmalas@ee.bilkent.edu.tr, lgurel@bilkent.edu.tr). Computational Electromagnetics Research Center (BiLCEM), Bilkent University, TR-06800, Bilkent, Ankara, Turkey.

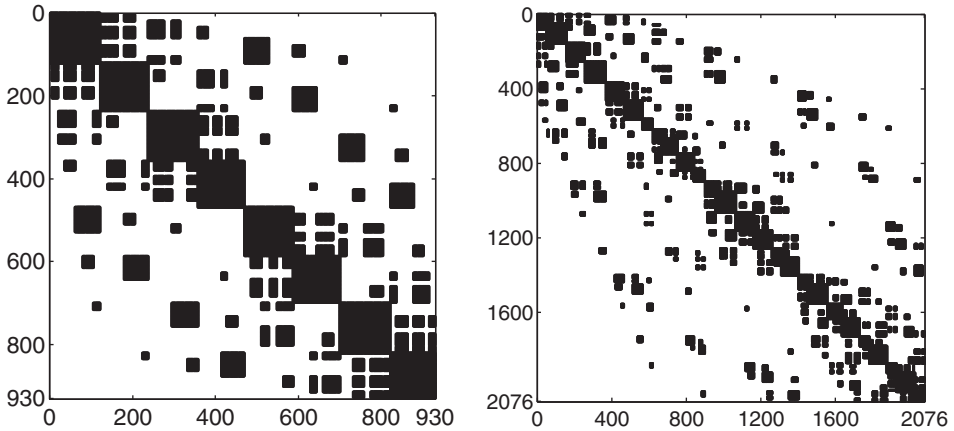

FIG. 1.1. *Sparsity patterns of the near-field matrices for a sphere geometry with two sizes:* (a) 930-*unknown problem with* 208,386 *nonzero matrix elements and* (b) 2,076-*unknown problem with* 405,968 *nonzero entries.*

On the other hand, constructing parallel and efficient preconditioners for CEM applications can be difficult. MLFMA stores only the near-field matrix, which is composed of the interactions of the neighboring (touching) boxes or clusters in the lowest level of the tree structure. When the ordering of the unknowns is in accordance with the cluster membership, the near-field matrix takes a block structure. For example, in Figure 1.1, we show the near-field patterns of two problems involving sphere geometries of different sizes. Both the maximum size of the blocks and the maximum number of the nonzero blocks per row are fixed. Therefore, as the problem size increases, the near-field matrix becomes sparser. Since preconditioners are usually built from near-field matrices, effective preconditioning of CEM problems may become a challenge, particularly for large problem sizes.

Nonetheless, effective utilization of the near-field matrix provides strong preconditioners for problems up to certain large sizes. Each element of the matrix represents the electromagnetic interaction of a basis function and a testing function. Green's function used for the computation of the matrix elements decays with $1/R$, where R is the distance between the pair of basis and testing functions under consideration. Due to this rapid decay of Green's function, magnitudes of the matrix elements display a variety. The general trend of this variety obeys physical proximity; i.e., basis and testing functions that are close to each other are expected to have strong electromagnetic coupling, resulting in matrix elements with relatively larger magnitudes. Therefore, the sparse near-field matrix is likely to retain the most relevant contributions of the dense matrix. The exact inverse of such a sparse matrix is, in general, a dense matrix. Nevertheless, the inverse matrix also displays a similar variation among the magnitudes of its elements. Hence, the inverse matrix can also be approximated by a sparse matrix.

In this work, we consider sparse-approximate-inverse (SAI) preconditioners for large CEM problems. This is partly because an efficient parallelization of the more standard incomplete LU (ILU) preconditioners [19] is difficult for matrices with unstructured sparsity patterns [1]. Application of the SAI preconditioners to CEM problems in the context of MLFMA has been analyzed by the CERFACS group [5] and by Lee, Zhang, and Lu [17]. Here, we present an effective construction scheme

with an effective load-balancing method that produces high parallel efficiency. We also propose to use the near-field pattern for the approximate inverse with filtered matrices and then compare different filtering strategies. Moreover, for conductor problems, the earlier work [5] considered only the electric-field integral equation (EFIE). However, for conducting geometries with closed surfaces, the combined-field integral equation (CFIE) should also be considered. Even though EFIE can also be used in such problems, this has no practical use since CFIE can solve the closed-surface problems much faster. Furthermore, we show that CFIE solutions of large real-life problems with closed surfaces can benefit more from SAI than from simple preconditioners, such as the block-diagonal preconditioner.

This paper is organized as follows. After presenting a brief summary of the SAI preconditioners in the next section, we dwell upon the implementation details in section 3. In particular, we explain pattern selection and filtering strategies. For a parallel implementation, we present a load-balancing algorithm and show how the communication in the construction phase can be efficiently performed. The results section analyzes CFIE and EFIE problems separately. Then, in section 5, we discuss some conclusions.

2. Brief review of SAI. Commonly used ILU preconditioners approximate the original matrix in the form of incomplete factors. In each step of an iterative method, preconditioning is performed by backward and forward solves. In contrast, SAI preconditioners are based on approximating the inverse of the matrix directly. For this purpose, an approximate inverse is explicitly constructed and stored. Then, the preconditioner is applied by a sparse matrix-vector multiplication. In the context of MLFMA, the $n \times n$ dense coefficient matrix is decomposed as $\overline{\mathbf{A}} = \overline{\mathbf{A}}^{near} + \overline{\mathbf{A}}^{far}$, where $\overline{\mathbf{A}}^{near}$ is the available sparse near-field matrix and the application of $\overline{\mathbf{A}}^{far}$ to a vector is computed by MLFMA. Hence, we use $\overline{\mathbf{A}}^{near}$ to generate the preconditioner and our approximation is of the form $\overline{\mathbf{M}} \approx (\overline{\mathbf{A}}^{near})^{-1}$.

In this work, we concentrate on the SAI preconditioners derived from the Frobenius norm minimization. There are two other classes of approximate inverses that have been proposed in the literature [3]. One of the classes involves the factorized sparse approximate inverses, whose important examples are abbreviated as FSAI [16] and AINV [2] by different authors. Both FSAI [16] and AINV [2] have already been tried on CEM problems, and their performances have been discouraging [4]. SAI preconditioners of the third class are the inverse ILU techniques, which consist of approximately inverting an incomplete factorization of the matrix. Because of the initial incomplete factorization phase, the inverse ILU methods have some serious drawbacks for parallel computing [3].

2.1. Methods derived from the Frobenius norm minimization. For this class of preconditioners, the approximate inverse of the near-field matrix is computed by minimizing

$$\left\| \overline{\mathbf{I}} - \overline{\mathbf{M}} \cdot \overline{\mathbf{A}}^{near} \right\|_F . \tag{2.1}$$

The approximation is implemented by forcing $\overline{\mathbf{M}}$ to be sparse. With the Frobenius norm choice, the minimization can be performed independently for each row by using the identity

$$\left\| \overline{\mathbf{I}} - \overline{\mathbf{M}} \cdot \overline{\mathbf{A}}^{near} \right\|_F^2 = \sum_{i=1}^{n} \left\| \mathbf{e}_i - \mathbf{m}_i \cdot \overline{\mathbf{A}}^{near} \right\|_2^2 , \tag{2.2}$$

where $\mathbf{e}_i$ is the ith unit row vector and $\mathbf{m}_i$ is the ith row of the preconditioner.

Various preconditioners have been developed with different pattern selection and minimization techniques. Chow and Saad proposed to solve each equation $(\overline{\mathbf{A}}^{near})^T \cdot \mathbf{m}_i^T = \mathbf{e}_i^T$ iteratively and approximately [8]. One way to do this is to use the first few iterations of the generalized minimal residual (GMRES) solver. However, the cost of this method is of order $\mathcal{O}(n^2)$ for sparse matrices, assuming a fixed number of nonzero elements per row. To avoid this high cost, the authors proposed to keep the iterates and other vectors sparse, as well as the matrix. This is done by filtering iterates as they become denser. Then, matrix-vector multiplications are carried out in sparse-sparse mode. However, such a multiplication scheme is not efficiently implemented with MLFMA.

Considering the difficulty in finding a suitable nonzero pattern for the approximate inverse, Grote and Huckle [13] proposed to find the sparsity pattern adaptively starting with an initial sparsity pattern. Construction time of this preconditioner can be very high [3], and, hence, it should be used only if simpler methods fail.

On the other hand, the nonzero structure of the near-field matrix itself is a natural candidate for the nonzero pattern of the SAI preconditioner. The storage scheme used for the block-sparse matrices consumes less memory than regular sparse matrices. Moreover, as noted in [5], when using the block structure of the near-field matrix, QR factorization involved in the least-squares solutions of (2.2) can be done once for each diagonal block, which corresponds to self-interactions of the last-level clusters in MLFMA. In this way, construction time of the preconditioner can be reduced substantially. If filtering is required, however, the block structure is distorted and both the setup time and memory consumption of the preconditioner can be even higher than the no-filtering case. Moreover, in a parallel implementation, load balancing should be ensured and communications in the construction phase should be carefully performed, since this phase involves all-to-all exchanges of rows. In the next section, we analyze these issues in more detail.

3. Parallel implementation details. In this work, we adopt K-way rowwise conformable partitionings of the near-field matrix $\overline{\mathbf{A}}^{near}$, the approximate inverse $\overline{\mathbf{M}}$, and the right-hand side (RHS) vector $\mathbf{b}$ as

$$\overline{\mathbf{A}}^{near} = \begin{bmatrix} \overline{\mathbf{A}}_1^{near} \\ \vdots \\ \overline{\mathbf{A}}_k^{near} \\ \vdots \\ \overline{\mathbf{A}}_K^{near} \end{bmatrix}, \quad \overline{\mathbf{M}} = \begin{bmatrix} \overline{\mathbf{M}}_1 \\ \vdots \\ \overline{\mathbf{M}}_k \\ \vdots \\ \overline{\mathbf{M}}_K \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \vdots \\ \mathbf{b}_k \\ \vdots \\ \mathbf{b}_K \end{bmatrix}, \tag{3.1}$$

where $\overline{\mathbf{A}}_k^{near}$ and $\overline{\mathbf{M}}_k$ are $n_k \times n$ submatrices, $\mathbf{b}_k$ is an $n_k \times 1$ subvector, and

$$\sum_{k=1}^{K} n_k = n. \tag{3.2}$$

Process P_k holds $\overline{\mathbf{A}}_k^{near}$, $\mathbf{b}_k$, and $\overline{\mathbf{M}}_k$. However, we use a different partitioning for $\overline{\mathbf{M}}$ during the generation of SAI as explained in section 3.3.

The construction of the SAI preconditioner is accomplished by solving

$$\mathbf{m}_i \cdot \overline{\mathbf{A}}^{near} = \mathbf{e}_i \qquad \text{for } i = 1, 2, \ldots, n \tag{3.3}$$

subject to sparsity conditions. Left preconditioning is consistent with rowwise decomposition, because in this scheme the computations involve the rows of the original

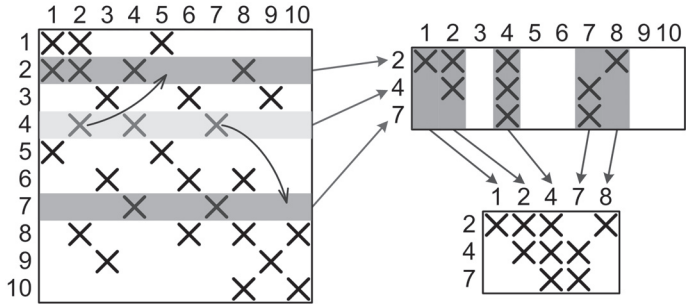

FIG. 3.1. *Reduction of a* 10×10 *matrix for the generation of the* 4*th row of SAI.*

matrix, and each row block $\overline{\mathbf{M}}_k$ of the approximate inverse is generated by a different process. However, for the EFIE formulation, which produces symmetric complex matrices, right preconditioning is also viable. This can be accomplished by a transpose matrix-vector multiplication operation in the application phase as will be detailed in section 3.5.

In a rowwise decomposition of the matrix, each process P_k solves part of (3.3). For a given row $i \in P_k$, let $I \subset \{1, 2, \ldots, n\}$ denote the set of column indices j for which $\mathbf{m}_i(j)$ is nonzero. Then, only the rows of the near-field matrix included in this set affect the solution. Therefore, the ith minimization problem is reduced to

$$\mathbf{m}_i(I) \cdot \overline{\mathbf{A}}^{near}(I,:) = \mathbf{e}_i. \tag{3.4}$$

This step incurs a communication among the processes, because not all of the rows in I belong to P_k. Hence, P_k requires some sparse rows (i.e., nonzero values and column indices) from other processes. Once $\overline{\mathbf{A}}^{near}(I,:)$ is formed, because of the sparsity of the near-field matrix, some of the columns of $\overline{\mathbf{A}}^{near}(I,:)$ will be zero. Denoting the indices of the nonzero columns by J, the $n \times n$ problems in (3.3) are reduced to $n_1 \times n_2$ problems

$$\mathbf{m}_i(I) \cdot \overline{\mathbf{A}}^{near}(I,J) = \mathbf{e}_i(J) \qquad \text{for } i = 1, 2, \ldots, n, \tag{3.5}$$

where n_1 and n_2 are the number of elements in the sets I and J, respectively. An example of reduction of a 10×10 sparse matrix for the 4th row is illustrated in Figure 3.1. In this example, the sparsity pattern of SAI is the same as that of the original matrix.

The $n_2 \times n_1$ problems

$$\overline{\mathbf{A}}^{near}(I,J)^T \cdot \mathbf{m}_i(I)^T = \mathbf{e}_i(J)^T \tag{3.6}$$

can be solved by first computing the reduced QR factorization

$$\overline{\mathbf{A}}^{near}(I,J)^T = \overline{\mathbf{Q}} \cdot \overline{\mathbf{R}} \tag{3.7}$$

and then obtaining the solution as

$$\mathbf{m}_i(I)^T = \overline{\mathbf{R}}^{-1} \cdot \overline{\mathbf{Q}}^H \cdot \mathbf{e}_i(J)^T. \tag{3.8}$$

For the near-field matrix, the maximum number of nonzero elements in a row or a column is fixed for a given problem irrespective of the problem size. Therefore, a

```
find the largest entry max_k of A_k^near
global_max = reduce_all_maximum(max_k)
for each a_ij ∈ A_k^near do
   if  a_ij/global_max < threshold then
      drop a_ij
   endif
endfor
```

FIG. 3.2. *Filtering algorithm.*

constant number of rows are involved in (3.4), i.e., $n_1 = \mathcal{O}(1)$. In $\overline{\mathbf{A}}^{near}(I,:)$, each row again contains a fixed number of nonzero entries. The worst case occurs when the locations of the nonzero elements do not coincide for all rows $i \in I$. Even in that case, $n_2 = \mathcal{O}(1)\mathcal{O}(1) = \mathcal{O}(1)$. This makes the complexity of the SAI preconditioner $\mathcal{O}(n)$. On the other hand, the QR factorization used in the solution of the $n_2 \times n_1$ least-squares problem requires asymptotically $n_2 n_1^2$ flops, causing the setup time of the SAI preconditioner to be high, even though it has a low complexity.

Because of this possible high construction cost, the implementation of the SAI preconditioner deserves close attention. The following subsections will detail the main steps for the generation and application of the preconditioner.

3.1. Pattern selection and filtering. In a Frobenius-norm minimization technique that depends on a fixed inverse pattern, the main issue for an efficient and effective preconditioner is the selection of an appropriate sparsity pattern. Because of the possible high cost of the SAI preconditioner, filtering is used in general. Filtering refers to dropping small elements from the original matrix. Then, the preconditioner is constructed from this sparser matrix. In our work, we have used the algorithm detailed in Figure 3.2 for filtering.

Filtering decreases only n_2 if a different pattern from the filtered matrix is used for the approximate inverse. In that case, n_1 is determined by the pattern of the approximate inverse. However, filtering causes smaller n_1 and n_2 values if the pattern of the filtered matrix is used for the approximate inverse.

Considering MLFMA and the special structure of the near-field matrix, we think that the following pattern selection and filtering strategies are appropriate for low-cost SAI generation.

3.1.1. No filtering. In this strategy, no filtering is applied to the near-field matrix and the same pattern is used for the approximate inverse. Because of the block structure of $\overline{\mathbf{A}}^{near}$, all rows of the SAI preconditioner that reside in the same diagonal block require the same rows of $\overline{\mathbf{A}}^{near}$ for their computation; hence, the reduced matrices $\overline{\mathbf{A}}^{near}(I,:)$ and $\overline{\mathbf{A}}^{near}(I,J)$ become the same. Therefore, QR factorization is done only once for each diagonal block and the least-squares solution is obtained for multiple RHSs. Since the least-squares solution is dominated by the QR factorization, substantial savings can be achieved.

3.1.2. Preserving block structure in filtering. Even though the near-field matrices become sparser as the number of unknowns increases, filtering may be required. This may be due to the possible high cost of SAI construction or because of some special problems, such as densely packed metamaterial structures [12], which produce denser near-field matrices. To be able to use the advantage of the block structure, we suggest using the near-field pattern for the approximate inverse and

```
for each m_ij ∈ M̄_k do
   if  j is not marked then
      p = findProcId(j)
      append j into rowRecvList[p]
      mark j
   endif
endfor
send rowRecvList; receive into rowSendList          ! All-to-all communication
```

FIG. 3.3. *The pseudocode that finds the rows to be sent by the process* P_k.

```
for each row i ∈ rowSendList do
   append column indices of row i to sendColIndices
endfor
send sendColIndices; receive into recvColIndices    ! All-to-all communication
```

FIG. 3.4. *The pseudocode that finds the column indices of the sparse rows to be received by the process* P_k.

filtering only the near-field matrix, from which the approximate inverse is generated. In this way, the row size n_1 does not change, but n_2 can be much smaller. Hence, we expect cheaper construction time compared to the no-filtering case.

3.1.3. Using a filtered pattern for the approximate inverse. If we use the block structure for the approximate inverse, the memory requirement of the preconditioner will be the same as the near-field matrix, which is the largest data in MLFMA. One way to reduce the memory cost is to use a filtered pattern for the preconditioner. On the other hand, with this strategy, we will not be able to use the advantage of the block structure. Therefore, we have to perform n factorizations instead of n/m, where m is the average size of the diagonal blocks. Hence, substantial filtering should be employed in order to decrease the memory and construction costs with this scheme.

3.1.4. Block filtering. Another strategy to take advantage of the block structure can be to drop an entire block, instead of only the nonzero entries, with the hope that dropped blocks do not carry significant information. To determine which blocks to drop, the Frobenius norm of each one is computed; those having a relative norm smaller than a prescribed tolerance are dropped.

3.2. Communication phase and enlarging the local submatrix. After a suitable pattern is selected for SAI, each process P_k exchanges some rows of $\overline{\mathbf{A}}_k^{near}$ with others. In this way, they enlarge their local submatrix $\overline{\mathbf{A}}_k^{near}$ so that no communication is required during the generation of $\overline{\mathbf{M}}_k$. For this purpose, P_k scans the nonzero pattern of $\overline{\mathbf{M}}_k$ and decides which rows it needs for the generation of the kth block. Then, after an all-to-all communication, each process learns the row identities it has to send. This communication pattern is detailed in Figure 3.3.

However, the information obtained is not sufficient for the exchange of rows because the processes do not yet know the column indices of the nonzero entries of the rows to be received. Hence, another scan and exchange of data regarding the column indices is performed. Finally, the values are exchanged. These two steps are illustrated in Figures 3.4 and 3.5.

The near-field matrix and SAI are held in compressed sparse row (CSR) format. This has two advantages. First, access to memory is minimized for the sparse matrix-

```
for each row i ∈ rowSendList do
   append a_ij to sendColValues
endfor
send sendColValues; receive into recvColValues          ! All-to-all communication
```

FIG. 3.5. *The pseudocode that exchanges the sparse rows.*

vector multiplications. More importantly, with CSR storage, the access of the matrix is done by rows, and, hence, the communications in the last two steps are done in place.

There could be another way to exchange the rows, in which the communication is done during the construction of the SAI preconditioner. This approach allows communications and computations to overlap by exchanging data for the next row while computing the current row. However, in this method, a row can be exchanged many times. Moreover, as shown in section 4.1, our implementation produces superior parallel performance; hence, we did not need to try this alternative strategy.

3.3. Load balancing of SAI.

3.3.1. Load balancing for the generation phase. The computation of the nonzero elements of the near-field matrix constitutes an expensive part of the setup phase in MLFMA. In this part, the cost of a row is proportional to the number of nonzero elements in that row. To ensure load balancing, the rows of the near-field matrix are distributed among the processes so that each process acquires approximately an equal number of nonzero elements. Since the application of the near-field matrix in the iterative phase is also proportional to the number of nonzero elements in a submatrix, this approach serves the load balancing of the near-field matrix-vector multiplication as well.

On the other hand, the cost of the generation of the ith row $\mathbf{m}_i$ of SAI is proportional to $n_2 n_1^2$, where n_1 and n_2 are the dimensions of the reduced matrix. Note that n_1 is the number of the nonzero elements in that row if filtering is not applied. If the near-field partitioning is also used for SAI, this high cost can cause the SAI generation to be unbalanced. For this reason, we repartition the near-field matrix in accordance with the load-balancing scheme of the SAI setup.

After the pattern of SAI is decided, we can quickly determine the cost of each row by finding n_1 and n_2 values. Then, the workload of SAI is distributed among the processes so that each process has approximately an equal amount of work. Alternatively, it is also possible to apply an incremental partitioning to existing near-field partitioning to decrease the overhead of repartitioning as detailed in [7]. We follow the former approach, where we use a separate partitioning for the SAI generation that is different from the partitioning of the near-field matrix. This way, we obtain a better load balance, and we can still limit the overhead of repartitioning by overlapping the communications with computations as explained in the next section.

3.3.2. Load balancing for the SAI application. To ensure load balancing for the application phase, we have to redistribute the rows of SAI according to the near-field partitioning. The overhead of this data transfer can be eliminated by overlapping communications with computations as detailed in Figure 3.6. In the first loop, all processes initiate the receptions of the rows that they should have with respect to the near-field partitioning, but they do not generate. Then, all processes generate those rows in their SAI partitioning that do not belong to themselves and initiate their transfers to appropriate processes. While the communications take place, local

```
for each row i ∈ R_k^near do
   if  row i ∉ R_k^SAI then
      p = findProcId(i)
      start the reception of m_i from p            ! Nonblocking communication
   endif
endfor
for each row i ∈ R_k^SAI do
   if  row i ∉ R_k^near then
      p = findProcId(i)
      generate m_i and start the transfer to p     ! Nonblocking communication
   endif
endfor
for each row i ∈ R_k^SAI do
   if  row i ∈ R_k^near then
      generate m_i
   endif
endfor
finish all nonblocking communications
```

FIG. 3.6. *Redistribution of the SAI rows according to the near-field partitioning.* R_k^{near} *and* R_k^{SAI} *denote the row indices of process* k *with respect to the near-field and SAI partitionings, respectively.*

computations, i.e., the generation of the rows that belong to process k with respect to both near-field and SAI partitionings, are performed. Finally, all processes wait for the nonblocking communications to finish.

3.4. Construction of the preconditioner. For the generation of the ith row $\mathbf{m}_i$, first a map of length n is prepared to map the sets I and J to $\bar{I} = \{1, 2, \ldots, n_1\}$ and $\bar{J} = \{1, 2, \ldots, n_2\}$, respectively. Then, we form the $n_2 \times n_1$ dense matrix

$$\overline{\mathbf{A}}^{near}\left(\bar{I}, \bar{J}\right)^T = \overline{\mathbf{A}}^{near}\left(\bar{J}, \bar{I}\right). \tag{3.9}$$

Finally, we solve the least-squares problem

$$\overline{\mathbf{A}}^{near}\left(\bar{J}, \bar{I}\right) \cdot \mathbf{m}_i(I)^T = \mathbf{e}_i(J)^T \tag{3.10}$$

via QR factorization and generate the ith row of $\overline{\mathbf{M}}_k$.

3.5. Application of the preconditioner. The application of the preconditioner is performed with the sparse matrix-vector multiplication $\mathbf{y}_k = \overline{\mathbf{M}}_k \cdot \mathbf{x}$. Since P_k computes $\mathbf{x}_k$, an expand operation, i.e., $\mathbf{x} = \text{expand}(\mathbf{x}_k)$ (also known as "gather all"), is required before the multiplication so that all processes possess the entire $\mathbf{x}$ vector.

For EFIE, using the symmetry of the near-field matrix, we have

$$\left\|\overline{\mathbf{I}} - \overline{\mathbf{M}} \cdot \overline{\mathbf{A}}^{near}\right\|_F = \left\|\overline{\mathbf{I}} - \left(\overline{\mathbf{M}} \cdot \overline{\mathbf{A}}^{near}\right)^T\right\|_F = \left\|\overline{\mathbf{I}} - \overline{\mathbf{A}}^{near} \cdot \overline{\mathbf{M}}^T\right\|_F. \tag{3.11}$$

Therefore, right preconditioning can be achieved with the operation $\mathbf{y}^k = (\overline{\mathbf{M}}_k)^T \cdot \mathbf{x}_k$ or equivalently $(\mathbf{y}^k)^T = (\mathbf{x}_k)^T \cdot \overline{\mathbf{M}}_k$, where $\mathbf{y} = \sum_{k=1}^{K} \mathbf{y}^k$. This multiplication can be done in CSR format using the outer product form of matrix-vector multiplication,

```
y^k = 0
for i = 1 to n_k do
    xval = x_k(i)
    kStart = IA(i); kEnd = IA(i + 1) − 1
    for k = kStart to kEnd do
        j = JA(k)
        y^k(j) = y^k(j) + xval ∗ VA(k)
    endfor
endfor
```

FIG. 3.7. *The pseudocode for the sparse matrix-vector multiplication used for right preconditioning.* **IA**, **JA**, *and* **VA** *are, respectively, row-index, column-index, and value arrays of* $\overline{\mathbf{M}}_k$, *which is stored in CSR format.*

i.e.,

$$\mathbf{y}^k = \sum_{i=1}^{n_k} \mathbf{x}_k(i)\overline{\mathbf{M}}_k(i,:). \tag{3.12}$$

We outline this operation in Figure 3.7. Finally, a fold operation, i.e., $\mathbf{y}_k = \text{fold}(\mathbf{y}^k)$ (also known as "reduce scatter"), is required so that partial sums $\mathbf{y}^k$ are summed across the processes, and each process P_k ends up with the kth subvector $\mathbf{y}_k$ of $\mathbf{y}$.

4. Results. In this section, we present the parallel performance of the generation phase of the SAI preconditioner. Then, for EFIE and CFIE formulations, we compare different versions of SAI with other preconditioners.

The solutions presented in this section are obtained on a 16-node cluster connected with an Infiniband network. Each node includes two quad-core Intel Xeon processes and 16 GB of RAM. All of the results are obtained on 32 cores (4 processes on each node). For robustness, we use the GMRES method with no restart as the solver. Contrary to results presented in [5], the orthogonalization cost of GMRES is negligible, compared to the time spent on the matrix-vector multiplications. For example, the largest problem shown in this study involves 3,838,496 unknowns. For the solution of this problem, the time spent on GMRES orthogonalization is only 2.3% of the time spent on matrix-vector multiplications by MLFMA. We use zero as the initial guess and set the stopping criteria as a six order of magnitude relative decay in the initial residual or a maximum of 1,000 iterations. In our MLFMA implementation, we use the Rao–Wilton–Glisson functions [20] for both basis and testing functions. We set the size of the smallest clusters to 0.25λ and the number of accurate digits to three. Three digits of accuracy have proven to yield accurate results, as shown in [15] by comparing the numerical results with the analytical ones for the sphere problem formulated with CFIE. For the patch problem formulated with EFIE, accuracy is demonstrated by comparing the numerical solution with a physical optics solution that gives accurate results at some specific observation angles for high frequencies [18].

4.1. Parallel performance of the construction phase. In Figure 4.1, we show the speedup curves for the construction of SAI with no filtering for a patch geometry with 344,000 unknowns, a half-sphere with 408,064 unknowns, and the stealth target Flamme [14], which has 312,120 unknowns. To show the worst-case performance, the processes are distributed so that the internode communications are maximized. Thanks to our efficient parallelization scheme and the load-balancing method, we obtain superior speedups for all problems.

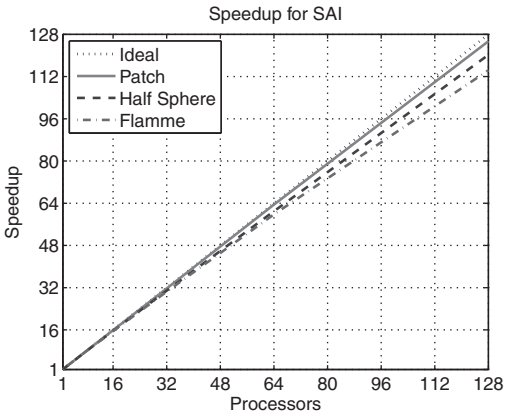

FIG. 4.1. *Speedup curves for the patch, half-sphere, and Flamme problems.*

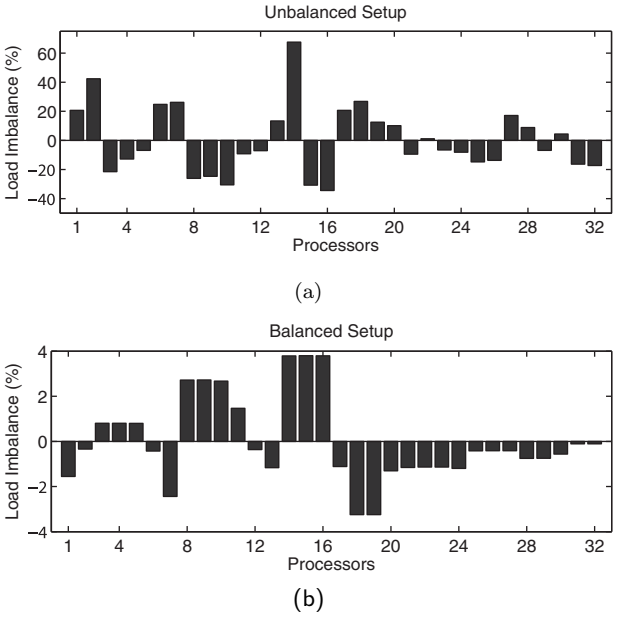

FIG. 4.2. *Load imbalance of the Flamme problem for* (a) *unbalanced and* (b) *balanced cases.*

The effect of the load-balancing algorithm is demonstrated in Figure 4.2 on the Flamme problem. The load imbalance ε_k of process k is defined as

$$\varepsilon_k = \frac{time_k - time_{avg}}{time_{avg}}. \tag{4.1}$$

In (4.1), $time_k$ is the setup time of SAI for process k and $time_{avg}$ is the average setup time. Particularly for complex geometries, such as Flamme, adopting the same partitioning of the near-field matrix for SAI can cause significant imbalance and inefficiency. Using the proposed load-balancing method, we reduce the average imbalance of 18.5% to 1.5% and achieve high efficiency.

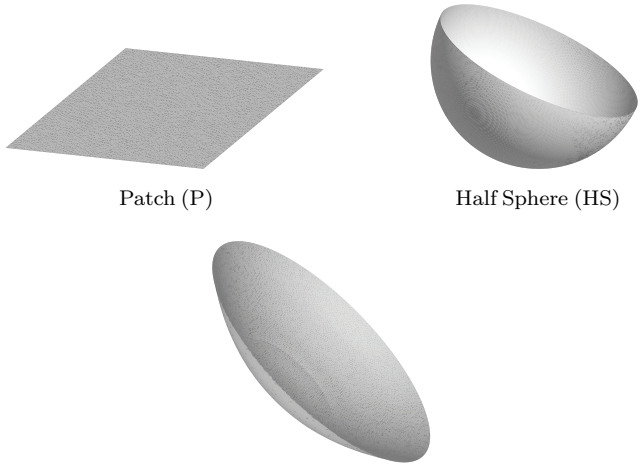

FIG. 4.3. *Open geometries used in EFIE problems.*

TABLE 4.1
Quantitative features of the open geometries.

Problem	Frequency (GHz)	Size (λ)	MLFMA levels	n
P1	16	16	7	85,840
P2	32	32	8	344,000
P3	64	64	9	1,377,280
P4	96	96	10	3,062,400
HS1	16	32	8	101,888
HS2	32	64	9	408,064
HS3	64	96	10	1,633,280
HS4	96	192	10	3,838,496
RA1	4	13	7	47,870
RA2	8	27	8	187,144
RA3	16	53	9	748,024
RA4	32	107	10	2,991,067

4.2. EFIE results. The sample geometries that are solved with EFIE in this paper are illustrated in Figure 4.3, and their quantitative features are listed in Table 4.1. Only open geometries are solved with EFIE since closed geometries can be solved more easily with CFIE. In Table 4.1, patch is abbreviated with P, half-sphere with HS, and reflector antenna with RA. The "size" value stands for the diameter of the half-sphere and the reflector antenna, and it is the length of one side for the square patch. Mesh lengths are chosen as one-tenth of the corresponding wavelength.

The experimental results indicate that there is no significant difference between left and right preconditioning of SAI. For consistency with the CFIE results, we prefer left preconditioning with EFIE. The results for the no-preconditioning case and comparisons of the three types of SAI preconditioners are shown in Tables 4.2 and 4.3, respectively. We omit the results with the block-filtering version of SAI, because it performs worse than other SAI preconditioners. We also omit the results with the block-diagonal preconditioner (BDP), because it deteriorates the convergence rate, compared to no preconditioning. In Table 4.3, "ratio" stands for the ratio of

TABLE 4.2
The solutions with no preconditioning for open geometries formulated by EFIE.

Problem	Iterations	Time (s)
P1	814	346
P2	$> 1,000$	-
P3	$> 1,000$	-
P4	$> 1,000$	-
HS1	913	1,363
HS2	$> 1,000$	-
HS3	$> 1,000$	-
HS4	$> 1,000$	-
RA1	795	446
RA2	$> 1,000$	-
RA3	$> 1,000$	-
RA4	$> 1,000$	-

TABLE 4.3
Comparison of SAI preconditioners for open geometries formulated by EFIE.

	Filtered pattern				Near-field pattern			No filtering		
Problem	Ratio	Setup	Iter	Time	Setup	Iter	Time	Setup	Iter	Time
P1	51%	14	94	41	2	91	38	2	74	31
P2	50%	62	139	224	10	132	209	10	109	174
P3	49%	370	194	1,431	45	190	1,384	48	157	1,147
P4	50%	1,495	243	7,849	129	231	7,368	132	194	6,225
HS1	73%	32	159	246	4	146	217	5	132	196
HS2	73%	133	266	1,762	19	246	1,583	20	221	1,424
HS3	71%	703	426	12,382	87	392	11,235	92	351	10,046
HS4	59%	2,512	599	30,828	343	570	28,295	350	480	23,458
RA1	6%	0	599	336	1	228	127	2	63	36
RA2	7%	1	859	1,890	9	557	1,230	9	93	204
RA3	36%	80	171	1,539	33	173	1,552	37	139	1,266
RA4	13%	1,142	598	22,269	148	303	11,144	201	200	7,276

Notes: "Ratio" is the ratio of the sparsity of the SAI to that of the near-field matrix.
"Setup" and "time" denote the setup and solution times, respectively, given in seconds.
"Iter" denotes the number of iterations.

the sparsity of the filtered near-field matrix to the original near-field matrix. "Setup" stands for the generation time of SAI and "time" for the solution time, both in seconds. For EFIE, we set the threshold of filtering at 0.5%.

We outline our observations as follows:

- Without an effective preconditioner, EFIE solutions converge only for small problems. On the other hand, SAI preconditioners solve all problems within reasonable iteration counts. Even for those that converge without preconditioning, SAI with no filtering decreases the iteration counts by an order of magnitude for the patch and the reflector antenna and by six times for the half-sphere.
- When we apply filtering, using the block structure of the near-field matrix for the approximate inverse decreases setup times significantly. Even for the reflector antenna, for which 90% of the near-field entries are dropped with filtering, setup times of SAI preconditioners that use the near-field pattern are much smaller. However, for large simulations, memory savings can be an important motivation to use the filtered pattern. For example, for RA4,

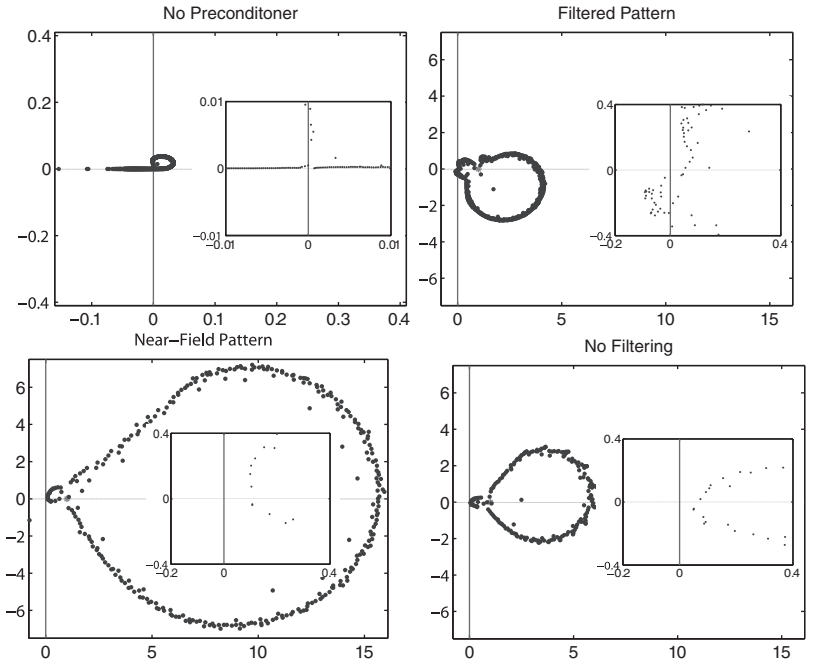

FIG. 4.4. *Approximate eigenvalues of the RA4 problem on the complex plane.*

SAI with a filtered pattern requires 840 MB of RAM, whereas SAI with the near-field pattern and SAI with no filtering require 4.2 GB of RAM. However, we note that there should be considerable filtering to provide memory gain, because the format used in a block-structured sparse matrix is more economical than regular sparse matrices.

- In terms of the solution times, SAI with no filtering produces the best results for all geometries. The setup times of the filtered SAI that uses the near-field pattern are the lowest, except for RA1 and RA2; however, SAI with no filtering is more successful in reducing the iteration counts and solution times.
- We observe superior algebraic scalability for the unfiltered SAI preconditioner. For all targets, the largest problem is approximately 64 times larger than the smallest, whereas the iteration count of the P4 is only 2.6 times that of P1, HS4 is 3.6 times that of HS1, and RA4 is only 3.2 times that of RA1.

Finally, in Figure 4.4, we demonstrate the Arnoldi estimates for the eigenvalues of the RA4 problem. These estimates are found as a byproduct of the GMRES solver, and they are known to approximate the bounding eigenvalues of the spectrum [22]. SAI with the filtered pattern leaves some of the eigenvalues in the left half-plane, and there are many small eigenvalues around the origin, accounting for its slow convergence. If the filtered pattern is used for the approximate inverse or filtering is not applied at all, then all of the eigenvalues are clustered in the right half-plane. However, SAI with no filtering produces also a smaller radius for the spectrum and, hence, converges faster.

4.3. CFIE results. Many real-life problems confronted in CEM involve complicated structures enclosing a volume. Due to its favorable properties, CFIE is the

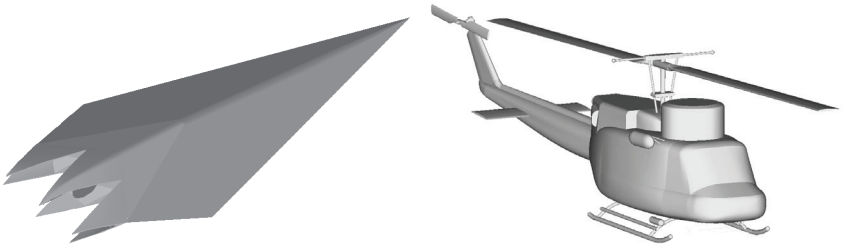

FIG. 4.5. *Closed-surface geometries used in CFIE problems.*

TABLE 4.4
Quantitative features of the closed geometries.

Problem	Frequency (GHz)	Size (λ)	MLFMA levels	n
F1	10	20	8	78,030
F2	20	40	9	312,120
F3	40	80	10	1,248,480
F4	60	120	10	3,166,272
H1	0.3	14	8	46,383
H2	0.6	28	9	185,532
H3	1.2	56	10	742,128
H4	2.4	112	11	2,968,512

TABLE 4.5
The solutions with BDP for closed-surface problems formulated by CFIE.

Problem	Iterations	Time (s)
F1	116	122
F2	122	563
F3	211	4,451
F4	347	11,734
H1	99	73
H2	109	391
H3	121	1,939
H4	138	10,192

preferred integral-equation formulation for those targets with closed surfaces. In Figure 4.5, we illustrate two such geometries: a helicopter (H) and the Flamme (F). We solve these problems at increasing frequencies as detailed in Table 4.4.

Contrary to EFIE, BDP is the commonly used preconditioner for CFIE problems. BDP has negligible setup time and is easily parallelized. In addition, BDP enables fast convergence for a variety of problems due to the diagonal-dominance behavior of CFIE matrices to some extent [11]. Hence, we first provide the solutions of the closed-surface problems with BDP in Table 4.5 and then compare different versions of SAI preconditioners in Table 4.6. With CFIE, we use a smaller threshold value for filtering, i.e., 0.05%, because such a small threshold causes significant filtering due to the diagonal-dominance feature of CFIE matrices.

We summarize our observations and comments about the CFIE results as follows:

- We observe that both the filtered SAI that uses the near-field pattern and the unfiltered SAI decrease the number of iterations and total solution times

TABLE 4.6
Comparison of SAI preconditioners for closed-surface problems formulated by CFIE.

Problem	Filtered pattern				Near-field pattern			No filtering		
	Ratio	Setup	Iter	Time	Setup	Iter	Time	Setup	Iter	Time
F1	25%	31	99	133	14	81	95	17	76	90
F2	21%	55	113	583	43	96	490	53	97	493
F3	19%	196	198	4,253	124	181	3,916	163	174	3,836
F4	33%	1,889	318	10,767	328	297	9,112	374	316	9,530
H1	11%	7	93	76	8	51	45	12	51	48
H2	6%	3	110	391	20	84	326	41	59	241
H3	6%	43	123	2,019	78	97	1,619	152	80	1,403
H4	6%	1,176	152	11,701	366	114	8,649	644	97	7,515

Notes: "Ratio" is the ratio of the sparsity of the SAI to that of the near-field matrix. "Setup" and "time" denote the setup and solution times, respectively, given in seconds. "Iter" denotes the number of iterations.

with respect to BDP for both problems. For instance, if we compare the largest targets, the solution times of F4 and H4 are shortened by 20% and 26%, respectively.

- In terms of the solution time, the unfiltered SAI is the most successful preconditioner, except for F2 and F4. Surprisingly, for these problems, obtaining the preconditioner from a sparser matrix instead of the original near-field matrix improves performance.

5. Conclusion. In this paper, we analyze SAI preconditioning for dense linear systems arising from the discretization of integral equations. We describe in detail practical issues, such as pattern selection, filtering, and load balancing, to obtain a highly parallel and efficient preconditioner.

For large open-surface problems that are modeled by EFIE, linear systems can be challenging to solve. We conclude that, for such problems, it is better to avoid filtering and to construct an SAI preconditioner that has the same block structure as the near-field matrix. The use of the block structure has advantages in reducing the setup cost and memory requirement of the preconditioner. However, if filtering is required for further memory saving, we show that our filtering strategy is robust.

For complex closed-surface problems that can make use of the well-conditioned CFIE, we show that SAI is more beneficial than the commonly used BDP. The benefit will be even more dominant for the computation of backscattering with different incident angles, which requires the solution of linear systems involving many right-hand sides.

REFERENCES

[1] M. BENZI, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[2] M. BENZI AND M. TUMA, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.

[3] M. BENZI AND M. TUMA, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.

[4] B. CARPENTIERI, I. S. DUFF, AND L. GIRAUD, *Experiments with Sparse Preconditioning of Dense Problems from Electromagnetic Applications*, Technical report TR/PA/00/04, CERFACS, Toulouse, France, 1999.

[5] B. CARPENTIERI, I. S. DUFF, L. GIRAUD, AND G. SYLVAND, *Combining fast multipole techniques and an approximate inverse preconditioner for large electromagnetism calculations*, SIAM J. Sci. Comput., 27 (2005), pp. 774–792.

[6] W. C. Chew, J.-M. Jin, E. Michielssen, and J. Song, eds., *Fast and Efficient Algorithms in Computational Electromagnetics*, Artech House, Norwood, MA, 2001.
[7] E. Chow, *Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns*, Int. J. High Perform. Comput. Appl., 15 (2001), pp. 56–74.
[8] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.
[9] Ö. Ergül and L. Gürel, *Fast and accurate solutions of large-scale scattering problems with parallel multilevel fast multipole algorithm*, in Proceedings of the IEEE Antennas and Propagation Society International Symposium, Honolulu, HI, 2007, pp. 3436–3439.
[10] Ö. Ergül and L. Gürel, *Efficient parallelization of the multilevel fast multipole algorithm for the solution of large-scale scattering problems*, IEEE Trans. Antennas and Propagation, 56 (2008), pp. 2335–2345.
[11] Ö. Ergül and L. Gürel, *Comparisons of FMM implementations employing different formulations and iterative solvers*, in Proceedings of the 2002 IEEE AP-S International Symposium and USNC/CNC/URSI National Radio Science Meeting, Columbus, OH, 2003, pp. 19–22.
[12] Ö. Ergül, A. Ünal, and L. Gürel, *MLFMA solutions of transmission problems involving realistic metamaterial walls*, in Proceedings of the 2007 Computational Electromagnetics Workshop, İzmir, Turkey, IEEE, 2007, pp. 79–82.
[13] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[14] L. Gürel, H. Bağci, J.-C. Castelli, A. Cheraly, and F. Tardivel, *Validation through comparison: Measurement and calculation of the bistatic radar cross section of a stealth target*, Radio Sci., 38 (2003), pp. 1046–1058.
[15] L. Gürel and Ö. Ergül, *Fast and accurate solutions of integral-equation formulations discretised with tens of millions of unknowns*, Electron. Lett., 43 (2007), pp. 499–500.
[16] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings* I. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.
[17] J. Lee, J. Zhang, and C.-C. Lu, *Sparse inverse preconditioning of multilevel fast multipole algorithm for hybrid integral equations in electromagnetics*, IEEE Trans. Antennas and Propagation, 52 (2004), pp. 158–175.
[18] T. Malas, Ö. Ergül, and L. Gürel, *Sequential and parallel preconditioners for large-scale integral-equation problems*, in Proceedings of the 2007 Computational Electromagnetics Workshop, İzmir, Turkey, IEEE, 2007, pp. 35–43.
[19] T. Malas and L. Gürel, *Incomplete LU preconditioning with the multilevel fast multipole algorithm for electromagnetic scattering*, SIAM J. Sci. Comput., 29 (2007), pp. 1476–1494.
[20] S. Rao, D. R. Wilton, and A. W. Glisson, *Electromagnetic scattering by surfaces of arbitrary shape*, IEEE Trans. Antennas and Propagation, AP-30 (1982), pp. 409–418.
[21] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[22] L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
[23] S. Velamparambil and W. C. Chew, *Analysis and performance of a distributed memory multilevel fast multipole algorithm*, IEEE Trans. Antennas and Propagation, 53 (2005), pp. 2719–2727.