This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAP.2019.2963619, IEEE Transactions on Antennas and Propagation

Efficient Solution of Electromagnetic Scattering Problems Using Multilevel Adaptive Cross Approximation (MLACA) and LU Factorization

Walton C. Gibson

***Abstract*—The Multilevel Adaptive Cross Approximation (MLACA), previously described in the literature, extends the single-level ACA with a recursive multilevel algorithm that significantly improves compression of off-diagonal matrix blocks resulting from electromagnetic Integral Equations (IE) discretized via the Method of Moments (MoM). In this paper, the MLACA approach is extended and applied to a direct solution of the MoM matrix system via LU factorization. It will be shown through numerical experiments that the off-diagonal LU blocks are also compressible using MLACA, yielding a compression rate superior to the single-level ACA and a memory complexity of $O(N^{4/3}\log N)$. In addition, the MLACA LU block updates are performed in rank-reduced form, yielding a very efficient software implementation via a Level 3 BLAS optimized for the CPU or GPU.**

***Index Terms*—Integral equations, method of moments (MoM), electromagnetic scattering, adaptive cross approximation (ACA), fast integral equation methods, matrix compression**

I. Introduction

THE solution of electromagnetic integral equations in the frequency domain remains one of the most accurate and effective methods of solving realistic radiation and scattering problems. The most popular solution technique remains the method of moments (MoM) [1], where the sources (currents) are discretized using basis functions with unknown coefficients, resulting in a dense matrix system. As a result, the required storage is of $O(N^2)$ and a direct solution via LU factorization of $O(N^3)$. In a straightforward implementation, this approach quickly consumes all available system RAM and CPU resources, limiting application of the MoM to problems of small electrical size. Thus out of necessity, the MoM has been historically eschewed for more tractable (but less accurate) asymptotic methods, such as Geometrical Optics (GO) [2] or Physical Optics (PO) [3].

There have been several approaches presented in the literature that attempt to mitigate both the high storage and computational requirements of the MoM. One is the Adaptive Integral Method (AIM) [4], which replaces the far-zone matrix elements with point-like currents on a rectangular grid, allowing for acceleration of the matrix-vector product in an iterative solver [5]. Another is the very popular Fast Multipole Method (FMM) [6], [7] and the Multilevel Fast Multipole Algorithm (MLFMA) [8], which group basis functions into spatially localized groups. During the iterative solver matrix-vector product, matrix blocks comprising interactions between "far" basis groups are then computed quickly using plane wave expansions. However, iterative solvers are known to have significant drawbacks, particularly when applied to electromagnetic problems. The FMM implementation is specific to the kernel's Green's function, and is thus not method-agnostic. Additionally, the MoM system is often ill-conditioned, leading to long iteration times, or the solver may not converge at all. Thus, a preconditioner may be required, which has its own setup and storage costs. Finally, having taken these precautions, the iteration time may still be extremely long for each right-hand side (RHS), or the method may still fail altogether.

The methods discussed thus far exploit the fact that matrix blocks representing the interactions between "well-separated" basis groups are rank deficient. More recently, there has been research made into methods for compressing and storing such blocks in rank-reduced form. The most naive approach is to apply the singular value decomposition (SVD), however the SVD has complexity of $O(N^3)$ and requires each matrix block to be completely filled. This is clearly undesirable for large problems. The ACA algorithm, originally presented in [9], is an algebraic, kernel-agnostic method for approximating rank-deficient matrix blocks. A key element in this algorithm is that it only requires knowledge of selected rows and columns of the block to be compressed; that is, the entire block does not need to be filled. It was applied successfully to electromagnetic problems [10], where the basis functions were partitioned into spatially localized groups, defining the block structure of the system matrix. However, in this paper an iterative solver was used to solve the matrix system. It was then shown [11] that if the ACA is applied to a block-LU factorization of the ACA-compressed system matrix, the off-diagonal LU matrix blocks are also compressible via the ACA. The author of that paper then showed that for scattering problems with many scattering vectors closely spaced in angle, the RHS blocks, as well as the corresponding solution (current) blocks, are also compressible via ACA. This use of LU factorization eliminates the reliance on iterative solvers, and is thus relatively immune to ill-conditioned matrix problems. This approach, referred to in this paper as the "single-level ACA", is particularly suited to efficient computer implementation, as block updates in the LU factorization (and RHS solutions) are performed in rank-reduced form using a compressed matrix representation. This technique has been used successfully in commercial MoM solver codes, where the block matrix operations are carried out via Level 3 BLAS operations, optimized for CPUs or high-

W. Gibson is with Tripoint Industries, Inc., Huntsville, Alabama, USA. (email: kalla@tripoint.org)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAP.2019.2963619, IEEE Transactions on Antennas and Propagation

performance graphics processing units (GPUs).

These concepts have since been extended via multilevel or hierarchical decomposition of the geometry and recursive butterfly-based compression schemes [12]. One such approach was a multilevel extension of the single-level ACA, known as MLACA, presented in [13]. In this approach, basis functions in each matrix block are further organized into sub-groups, based on certain criteria. Blocks are then split into column sub-blocks, which are compressed via ACA. Each resulting $\mathbf{U}$ matrix, comprising the singular values and left-singular vectors of each compressed sub-block, are stacked together and compressed again via ACA in a recursive butterfly scheme. Overall compression was shown to be greatly improved at the expense of increased calculation time, however, an iterative solver was again used. Whether MLACA could be used to compress the off-diagonal LU blocks in an LU factorization was not addressed. That paper also suggested that a special reorganization of each matrix block, based on the solid angle seen by all other matrix blocks, was needed to obtain the reported level of compression.

A butterfly-based direct solver was recently proposed in [14]–[16], implementing a hierarchical LU factorization compressed via a butterfly scheme. This approach does not use MLACA to compress the MoM or LU matrix blocks, instead being based on randomized butterfly reconstruction schemes [17], [18]. While the claimed compression is quite good, the proposed algorithm for the hierarchical LU factorization is very complicated and it remains unclear whether this algorithm is a candidate for GPU acceleration.

In this paper, we propose a semi-hierarchical technique that combines elements from the single-level ACA [11] and MLACA [13]. In this scheme, we partition the geometry using K-means clustering, and the MoM system matrix is computed and compressed using butterflies via MLACA. We then apply an algorithmically simple right-looking LU factorization, where the off-diagonal LU matrix blocks are also butterfly compressed using MLACA. We will show that the K-means block subdivision scheme is sufficient for MLACA to deliver additional compression of the MoM and LU matrix, yielding a memory requirement of $O(N^{4/3}\log N)$. Finally, we show that the proposed approach, just as in the single-level ACA, is a excellent candidate for CPU and (particularly) GPU acceleration using an optimized Level 3 BLAS, making an excellent addition to existing full matrix or ACA-based MoM solver codes.

This paper is organized as follows: In Section II, we will review the single-level ACA and block-LU factorization in compressed form, and in Section III we review MLACA. Section IV comprises the core of this paper, where we present our modifications of MLACA and apply it to the block-LU factorization. Finally, in Section V, we present several numerical examples that compare and contrast compute times and compression rates of a block-LU factorization using MLACA to that obtained using the single-level ACA.

II. Single-Level ACA

It is well known that if the basis functions are partitioned into spatially localized groups, the MoM matrix is now a block structure where off-diagonal blocks are rank-deficient. Consider off diagonal block $\mathbf{Z}_{m\times n}$ which has m rows and n columns. $\mathbf{Z}$ can be decomposed via the SVD as

$$\mathbf{Z}_{m\times n} = \mathbf{U}_{m\times m}\mathbf{\Sigma}_{m\times n}\mathbf{V}^*_{n\times n} \tag{1}$$

where the diagonals of $\mathbf{\Sigma}$ comprise the singular values, and $\mathbf{U}$ and $\mathbf{V}^*$ contain the left and right-singular vectors, respectively. As $\mathbf{Z}$ is rank-deficient, the singular values in $\mathbf{\Sigma}$ decrease quickly and $\mathbf{Z}$ can be approximated as

$$\mathbf{Z}_{m\times n} \approx \mathbf{Z}_u\mathbf{Z}_v = \mathbf{U}_{m\times k}\mathbf{V}^*_{k\times n} \tag{2}$$

where the effective rank $k \ll m, n$ is chosen based on a threshold τ of largest to smallest singular values. The use of SVD, however, is undesirable given its computational expense, and the fact that $\mathbf{Z}$ must be completely filled.

The ACA described in [9], [10] mitigates these difficulties. It is a simple, kernel-agnostic, fast rank-revealing algorithm that successively builds up approximations of $\mathbf{Z}_u$ and $\mathbf{Z}_v$ by selecting only k rows and columns from $\mathbf{Z}$, until the residual norm falls below a chosen threshold τ_{ACA}. Thus for small k, only a small fraction of $\mathbf{Z}$ needs to be computed, and the storage requirement for $\mathbf{Z}_u$ and $\mathbf{Z}_v$ is now $k(m+n)$, versus mn for $\mathbf{Z}$.

A. QR/SVD Recompression

The rows and columns of $\mathbf{U}$ and $\mathbf{V}$ as determined by ACA are not strictly orthonormal. Therefore, significant additional compression can be obtained by applying the QR/SVD recompression prescribed in [19], which requires QR decompositions of $\mathbf{U}$ and $\mathbf{V}$ and an SVD of size $k\times k$. In this step, it is usually sufficient to choose a $\tau_{SVD} \approx 10\ \tau_{ACA}$. For the single-level ACA, compression levels above 95% are possible, which improves further with increasing problem size. The QR/SVD recompression step is required in MLACA.

B. Clustering

Various algorithms for clustering the basis functions have been proposed. In [10] a recursive axis-aligned bounding-box strategy similar to that used in the MLFMA was used, and a "cobblestone" method was proposed in [11]. However, we have found that simple K-means clustering based on a target group size yields the best overall compression for the single-level ACA.

C. Matrix Fill

In matrix filling, diagonal blocks are not compressible and are computed and stored the usual way. Off-diagonal blocks are approximated by ACA+QR/SVD and only the compressed portions of those blocks are stored. To facilitate the ACA matrix filling, the programmer provides a routine that computes a single row or column of a matrix block.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAP.2019.2963619, IEEE Transactions on Antennas and Propagation

D. LU Factorization

The LU factorization comprises a right-looking block algorithm, described in Algorithm 1. Whereas standard right-looking algorithms found in libraries such as LAPACK [20] factor an entire panel and update the trailing sub-matrix, our algorithm updates each diagonal and its trailing row and column sequentially. In this way, the off-diagonal LU blocks are recompressed only once during the factorization.

Algorithm 1 Right-Looking Block-LU Factorization

1: **for** $b = 1$ to N **do**
2: $\quad \mathbf{A} = \mathbf{Z}_{bb} - \sum_{s=1}^{b-1} [\mathbf{L}_u \mathbf{L}_v]_{bs} [\mathbf{U}_u \mathbf{U}_v]_{sb}$
3: $\quad [\mathbf{LU}]_{bb} = \mathrm{LU}(\mathbf{A})$ $\quad$ (scalar LU factorization)
4: $\quad$ **for** $s = b + 1$ to N **do**
5: $\quad\quad \mathbf{L}_{sb} = \Big[[\mathbf{Z}_u \mathbf{Z}_v]_{sb} - \sum_{p=1}^{b-1} [\mathbf{L}_u \mathbf{L}_v]_{sp} [\mathbf{U}_u \mathbf{U}_v]_{pb} \Big] \mathbf{U}_{bb}^{-1}$
6: $\quad\quad$ Compress $\mathbf{L}_{sb}$ via ACA and store $[\mathbf{L}_u \mathbf{L}_v]_{sb}$
7: $\quad\quad \mathbf{U}_{bs} = \mathbf{L}_{bb}^{-1} \Big[[\mathbf{Z}_u \mathbf{Z}_v]_{bs} - \sum_{p=1}^{b-1} [\mathbf{L}_u \mathbf{L}_v]_{bp} [\mathbf{U}_u \mathbf{U}_v]_{ps} \Big]$
8: $\quad\quad$ Compress $\mathbf{U}_{bs}$ via ACA and store $[\mathbf{U}_u \mathbf{U}_v]_{bs}$
9: $\quad$ **end for**
10: **end for**

Note that in steps 2, 5 and 6, the block updates operate on full-size matrices. In step 2, the diagonal $\mathbf{Z}_{bb}$ is already full, and in steps 5 and 6, $\mathbf{Z}_{sb}$ and $\mathbf{Z}_{bs}$ are first expanded to full size. The update operations, however, are very fast, as the matrix products are performed in rank-reduced form. To illustrate, consider the product

$$\mathbf{L}^{m_1 \times n_1} \mathbf{U}^{m_2 \times n_2} = \mathbf{L}_u^{m_1 \times k_1} \mathbf{L}_v^{k_1 \times n_1} \mathbf{U}_u^{m_2 \times k_2} \mathbf{U}_v^{k_2 \times n_2} \tag{3}$$

where $n_1 = m_2$. The innermost product is performed first. This results in

$$[\mathbf{LU}]^{m_1 \times n_2} = \mathbf{L}_u^{m_1 \times k_1} \mathbf{A}^{k_1 \times k_2} \mathbf{U}_v^{k_2 \times n_2}. \tag{4}$$

Which product in (4) is performed next depends on $\min(k_1, k_2)$. If $k_1 < k_2$, we compute $\mathbf{B} = \mathbf{A}\mathbf{U}_v$ yielding

$$[\mathbf{LU}]^{m_1 \times n_2} = \mathbf{L}_u^{m_1 \times k_1} \mathbf{B}^{k_1 \times n_2} \tag{5}$$

otherwise we compute $\mathbf{C} = \mathbf{L}_u \mathbf{A}$, resulting in

$$[\mathbf{LU}]^{m_1 \times n_2} = \mathbf{C}^{m_1 \times k_2} \mathbf{U}_v^{k_2 \times n_2}. \tag{6}$$

These products are performed via the Level 3 BLAS function `cgemm`. The only operations in Algorithm 1 not done in rank-reduced form are the triangular forward and back-substitutions with $\mathbf{U}_{bb}^{-1}$ and $\mathbf{L}_{bb}^{-1}$, performed via `ctrsm`. Highly optimized versions of `cgemm` and `ctrsm` are available for the CPU as well as GPU. Note that in practice, the full-size $\mathbf{L}_{sb}$ and $\mathbf{U}_{bs}$ blocks each comprise a temporary full matrix which is compressed via ACA and then discarded.

E. Block-RHS Solution

Monostatic scattering problems often have thousands of incident angles, and solving for each RHS vector individually can be very time consuming. Let us instead group all the RHS vectors into a matrix $\mathbf{B}^{N \times N_{RHS}}$ where N is the number of unknowns and N_{RHS} the number of incident angles. It was shown in [11] that for problems with many closely spaced incident angles, if the ACA block grouping is applied to $\mathbf{B}$, the blocks of $\mathbf{B}$, as well as the corresponding solution blocks, are highly compressible using ACA. Thus, all right-hand sides can be solved for simultaneously via Algorithm 2. As in the LU factorization, all matrix products are in rank-reduced form and carried out efficiently via the Level 3 BLAS.

Algorithm 2 Block-RHS Solution

1: Given $\mathbf{LU}\ \mathbf{X} = \mathbf{B}$, first solve $\mathbf{UX} = \mathbf{L}^{-1}\mathbf{B}$:
2: **for** $b = 1$ to N **do**
3: $\quad \mathbf{B}_b = \mathbf{L}_{bb}^{-1} \big[[\mathbf{B}_u \mathbf{B}_v]_b - \sum_{s=1}^{b-1} [\mathbf{L}_u \mathbf{L}_v]_{bs} [\mathbf{B}_u \mathbf{B}_v]_s \big]$
4: $\quad$ Compress $\mathbf{B}_b$ via ACA and store updated $[\mathbf{B}_u \mathbf{B}_v]_b$
5: **end for**
6: Then solve $\mathbf{X} = \mathbf{U}^{-1} \big[\mathbf{L}^{-1} \mathbf{B} \big]$
7: **for** $b = N$ to 1 **do**
8: $\quad \mathbf{X}_b = \mathbf{U}_{bb}^{-1} \big[[\mathbf{B}_u \mathbf{B}_v]_b - \sum_{s=b+1}^{N} [\mathbf{U}_u \mathbf{U}_v]_{bs} [\mathbf{B}_u \mathbf{B}_v]_s \big]$
9: $\quad$ Compress $\mathbf{X}_b$ via ACA and store $[\mathbf{X}_u \mathbf{X}_v]_b$
10: **end for**

III. Multi-Level ACA (MLACA)

MLACA is based on the concept that the degrees of freedom (DoF) remains asymptotically constant when the source and testing group are inversely changed in size [12]. An L-level MLACA [13] further divides basis functions in a single-level ACA group into 2^L sub-groups, and applies a butterfly algorithm wherein source and testing groups are recursively joined or split, with the ACA applied at each level. To illustrate, let us consider an $L = 2$ level MLACA, where the off-diagonal block $\mathbf{Z}^{m \times n}$ now comprises row and column blocks having approximately $\frac{m}{2^L}$ and $\frac{n}{2^L}$ rows and columns, respectively. Operations on the first level $l = 0$ are depicted in Figure 1(a). Each block column of $\mathbf{Z}$ is compressed separately using ACA+QR/SVD, yielding four $\mathbf{U}$ and $\mathbf{V}$ matrix pairs. The singular values from the recompression step are retained in each $\mathbf{U}$. The $\mathbf{U}$ matrices are then stacked together to create matrix $\mathbf{A}_0^{(1)}$, whereas the smaller $\mathbf{V}$ matrices, comprising the diagonal blocks of the notional block-diagonal matrix $\mathbf{B}_0^{(1)}$, are fully compressed and are stored. We then proceed as in Figure 1(b), where we join pairwise the block columns of matrix $\mathbf{A}_0^{(1)}$, and split the result in half row-wise. We then recurse to level $l = 1$, where the block columns of each half are compressed in the same way as on the previous level. For each half, the $\mathbf{V}$ matrices are stored, and the process repeated again until the finest level is reached, which has only a single block row and column. After applying the ACA, the $\mathbf{U}$ and $\mathbf{V}$ matrix are stored and the MLACA compression of $\mathbf{Z}$ is complete. As the algorithm stacks together and recursively compresses the $\mathbf{U}$ blocks, we refer to this as a U-type MLACA. An error analysis of the algorithm was previously presented by other authors in [21].

A. Matrix Fill and Storage

When filling off-diagonal blocks, the procedure is similar to the single-level ACA, except the MoM solver now provides an additional routine that returns partial rows of each column block. During the MLACA recursion, upper level nodes of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAP.2019.2963619, IEEE Transactions on Antennas and Propagation

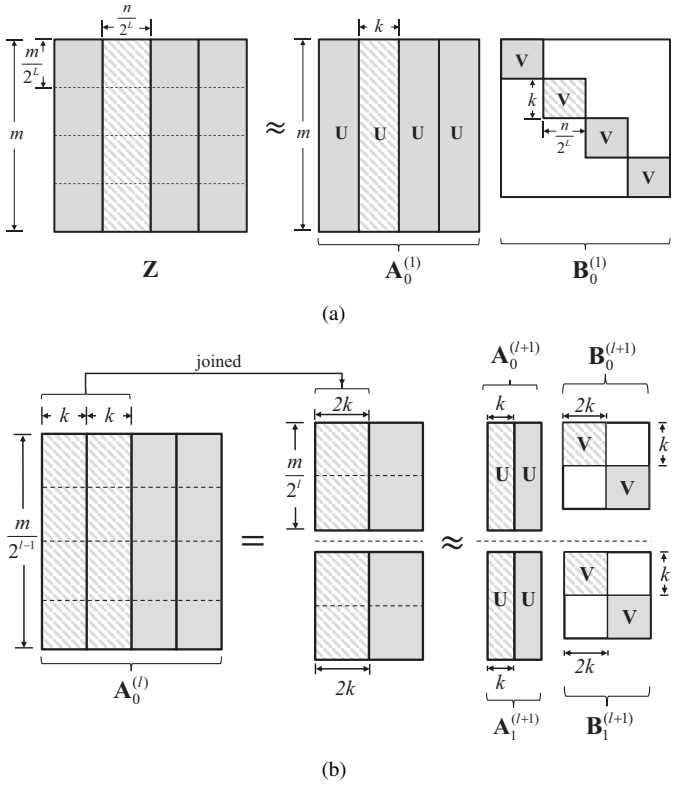


Fig. 1. 2-Level U-Type MLACA. (a) Operations on Level $l = 0$, (b) Operations on Level $l + 1$.

MLACA tree store only **V** matrices, whereas the leaf nodes store both **U** and **V**. These are stored in a single array in memory, with each node storing offsets into that array.

IV. Direct Solution of MLACA-Compressed Matrix System

We are now ready to investigate the direct solution of the MLACA-compressed matrix system via LU factorization. We again apply Algorithm 1 from Section II-D, where now all off-diagonal blocks of the MoM and LU matrix compressed via MLACA. To do this efficiently, we make several modifications to MLACA as originally presented.

A. Clustering of Sub-Groups

In [13] a tree-based subdivision of basis functions in angle space, based on the solid angle of a source group seen by a testing group, and vice versa, was used when compressing the off-diagonal blocks with MLACA. Additionally, it was said that this solid-angle approach is critical for the MLACA to obtain its gains in computation complexity and memory savings. However, this approach results in a local reordering of the rows and columns in each block that are necessarily different for other blocks. If using an iterative solver, this strategy is viable, as only elements from the input and output vectors have to be shuffled during the matrix-vector multiply with a block. However, this causes difficulties in performing steps 5 and 6 in Algorithm 1, as **Z**, **L**, and **U** would need to be unshuffled for each matrix product so that the global ordering was followed. This step would be very inefficient.

Our approach herein is to apply the same K-means algorithm used to create the single-level ACA groups. Each group is divided in half via K-means, which are recursively subdivided in half via K-means, and so on, until the required depth is reached. This comprises a per-group reordering, thus the global ordering is adjusted to match and no shuffling is required later. During development, this strategy was compared to the tree-based solid angle method when compressing off-diagonal blocks for a variety of test geometries. It was observed that the size of the compressed blocks differed by less than 2 percent in each case, validating our approach. The numerical examples in Section V will show that MLACA using the proposed K-means sub-group clustering delivers excellent compression.

B. Block Reconstruction

As before, the block updates in steps 5 and 6 of Algorithm 1 first expand $\mathbf{Z}_{sb}$ and $\mathbf{Z}_{bs}$ to full size. For MLACA-compressed blocks this is done via recursion, where nodes on level l reconstruct their portion of the matrix $\mathbf{A}^{l-1}$ on the previous level. On level $l = 0$ (the coarsest level), once $\mathbf{A}_0^{(1)}$ is available, the full matrix is then obtained via the product

$$\mathbf{Z} = \mathbf{A}_0^{(1)}\mathbf{B}_0^{(1)} \tag{7}$$

Operations on each level are very fast, as each reconstruction step comprises matrix products in rank-reduced form. When performing the updates, matrix products use the MLACA-compressed $\mathbf{A}_0^{(1)}$ and $\mathbf{B}_0^{(1)}$ factors of the **L** and **U** blocks. Thus, only the $\mathbf{A}_0^{(1)}$ factor (or $\mathbf{B}_0^{(1)}$ factor, see Section IV-C) is rebuilt for **L** and **U**. We will discuss how to perform these matrix products efficiently in the following section.

C. Matrix Product and V-Type MLACA

Matrix products in the LU block updates, discussed in Section II-D, are fast as they operate in rank-reduced form and require only a few calls to the Level 3 BLAS. However, consider the product **LU** of two U-type MLACA matrices at level $l = 0$ as shown in Figure 2. Prior to performing the product, the compressed **U**-column blocks of each matrix have been reconstructed. It is clear that the innermost product $\mathbf{L}_v\mathbf{U}_u$ is not efficient, as each small diagonal block of $\mathbf{L}_v$ multiplies only a small portion of each $\mathbf{U}_u$ block column. For the 2-level MLACA, this requires $2^L = 4$ `cgemm` calls.

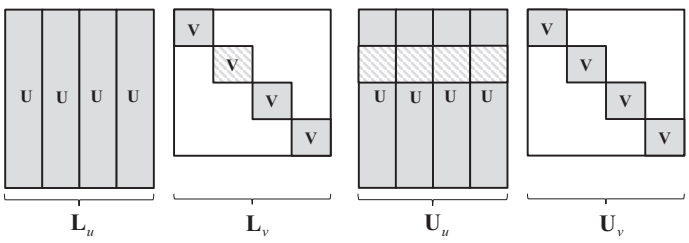


Fig. 2. Product of U-Type MLACA Matrices.

This innermost product can be reduced to a single operation if we modify the MLACA algorithm used to compress **L**.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAP.2019.2963619, IEEE Transactions on Antennas and Propagation

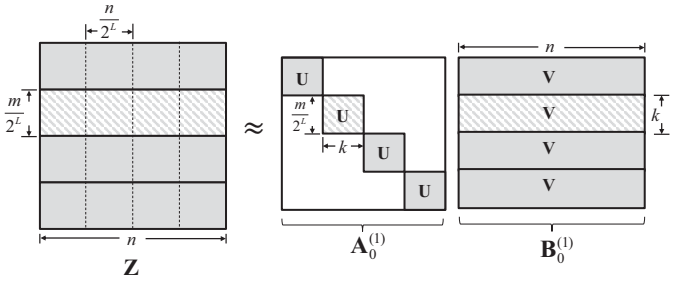


Fig. 3. V-Type MLACA.

Consider again the level $l = 0$ operations depicted in Figure 1(a). If we instead compress the block rows of $\mathbf{Z}$, we are presented with the configuration in Figure 3. The smaller diagonal $\mathbf{U}$ blocks of $\mathbf{A}_0^{(1)}$ are stored, and the larger $\mathbf{V}$ blocks comprising $\mathbf{B}_0^{(1)}$ are paired row-wise and split in half column-wise. These left and right halves are then recursively compressed in a manner analogous to the U-type MLACA. We refer to this variant as the V-type MLACA.

We now replace $\mathbf{L}$ in Figure 2 with a V-type compressed version. After first reconstructing the compressed row blocks of in $\mathbf{L}$ and compressed column blocks in $\mathbf{U}$, we are left with the matrix product of Figure 4. The innermost product now comprises a single operation, yielding

$$\mathbf{A}^{k_\mathbf{L} \times k_\mathbf{U}} = \mathbf{L}_v^{k_\mathbf{L} \times n_1} \mathbf{U}_u^{m_2 \times k_\mathbf{U}} \tag{8}$$

where $\mathbf{A}^{k_\mathbf{L} \times k_\mathbf{U}}$ is a $2^L \times 2^L$ block matrix, and $k_\mathbf{L}$ and $k_\mathbf{U}$ are the sum of the ranks of all the compressed sub-blocks in $\mathbf{L}$ and $\mathbf{U}$, respectively. As in (4), whether the left or right product is performed first depends on $\min(k_\mathbf{L}, k_\mathbf{U})$. If $k_\mathbf{L} < k_\mathbf{U}$, we compute $\mathbf{B} = \mathbf{A}\mathbf{U}_v$ yielding

$$[\mathbf{LU}]^{m_1 \times n_2} = \mathbf{L}_u^{m_1 \times k_\mathbf{L}} \mathbf{B}^{k_\mathbf{L} \times n_2} \tag{9}$$

otherwise we compute $\mathbf{C} = \mathbf{L}_u\mathbf{A}$, resulting in

$$[\mathbf{LU}]^{m_1 \times n_2} = \mathbf{C}^{m_1 \times k_\mathbf{U}} \mathbf{U}_v^{k_\mathbf{U} \times n_2}. \tag{10}$$

As $\mathbf{L}_u$ and $\mathbf{V}_v$ each comprise block-diagonal matrices of 2^L blocks, the product $\mathbf{LU}$ is thus reduced to $2^{L+1} + 1$ matrix products. As in the single-level ACA, each product is carried out in rank-reduced form. To ensure that all MLACA matrix products in the LU factorization have this form, we will compress all blocks below the diagonal using V-type MLACA, and all blocks above the diagonal using U-type MLACA.

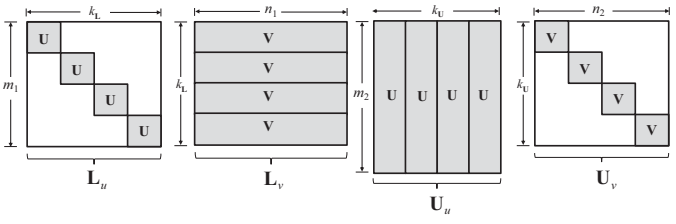


Fig. 4. Product of V and U-Type MLACA Matrices.

D. MLACA Block-RHS Solution

For problems with many right-hand sides, we again apply Algorithm 2. All $\mathbf{L}$ and $\mathbf{U}$ blocks are MLACA-compressed, however in this case we will still use the single-level ACA to compress the RHS blocks. Thus, some matrix products in Algorithm 2 comprise products between a U or V-type MLACA compressed matrix and single-level ACA compressed matrix, as illustrated in Figures 5(a) and 5(b), respectively.

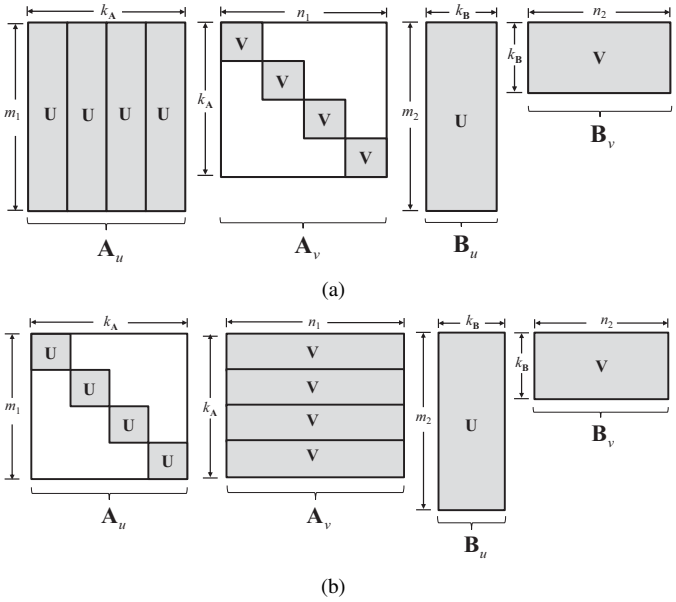


Fig. 5. Product of MLACA Matrix and ACA Matrix. (a) U-type product, (b) V-type product.

In both cases, the middle product is carried out first, yielding

$$\mathbf{C}^{k_\mathbf{A} \times k_\mathbf{B}} = \mathbf{A}_v^{k_\mathbf{A} \times n_1} \mathbf{B}_u^{m_2 \times k_\mathbf{B}}. \tag{11}$$

If $\mathbf{A}$ is U-type, whether $\mathbf{A}_u\mathbf{C}$ or $\mathbf{C}\mathbf{B}_v$ is performed next depends on which path results in the fewest total operations. If $\mathbf{A}$ is instead V-type, we perform the left-most product $\mathbf{A}_u\mathbf{C}$ first as $\mathbf{A}_u$ is a block-diagonal matrix. This yields

$$\mathbf{D}^{m_1 \times k_\mathbf{B}} = \mathbf{A}_u^{m_1 \times k_\mathbf{A}} \mathbf{C}^{k_\mathbf{A} \times k_\mathbf{B}} \tag{12}$$

and then finally

$$[\mathbf{AB}]^{m_1 \times n_2} = \mathbf{D}^{m_1 \times k_\mathbf{B}} \mathbf{B}_v^{k_\mathbf{B} \times n_2}. \tag{13}$$

As before, all products are performed in rank-reduced form.

V. Numerical Results

In this section we will compare the performance and accuracy of MLACA to the single-level ACA for several scattering problems. We will first consider a series of scattering problems of increasing electrical size, which we use to verify that MLACA will compress the off-diagonal LU matrix blocks while maintaining its compression gain over the single-level ACA. We will then apply MLACA to several scattering problems of large electrical size and confirm that its results remain accurate.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAP.2019.2963619, IEEE Transactions on Antennas and Propagation

A. MoM Solver

Numerical results in this section are computed using our *Serenity* radar cross section solver, a high-performance code written in C++. In *Serenity* we implement the direct single-level ACA solution from Section II, as well as the direct MLACA solution newly introduced in Section IV. *Serenity* treats arbitrarily shaped, three-dimensional objects using triangle surface meshes and RWG basis functions [22]. Composite conducting/dielectric objects with junctions are fully supported.

In the examples that follow, we use two versions of *Serenity*: a parallel GPU-accelerated version for shared-memory systems, and a CPU-only MPI version for distributed memory systems. In GPU-accelerated *Serenity*, near matrix elements are computed on the CPU, and far elements on the GPU. The LU block updates are performed on the GPU using NVIDIA cuBLAS. All other matrix operations are performed on the CPU using the Intel Math Kernel Library (MKL).

1) GPU Implementation: On NVIDIA GPUs, matrix products are most efficient if executed in large, single-kernel launches. However, reconstruction of the coarsest-level $\mathbf{V}$ and $\mathbf{U}$ matrix, required for MLACA matrix products, is a recursive operation involving many small matrix products. Rebuilding these on the GPU was found to be inefficient due to excessive kernel launches and low CUDA core occupancy. Thus, we instead reconstruct $\mathbf{V}$ and $\mathbf{U}$ on the CPU and upload the results to the GPU before performing the matrix products. This was found to much more evenly share the load between the CPU and GPU and yielded much faster run times.

2) Use of SVD: During MLACA compression, the $\mathbf{U}$ and $\mathbf{V}$ matrix are typically quite small in the k dimension. On higher levels in the recursion, we found it to be faster to simply apply the SVD directly than to use ACA+QR/SVD.

3) MPI Implementation: In our MPI implementation, a diagonal block and all $\mathbf{U}$ blocks in the column above it are assigned to a node, and the $\mathbf{L}$ blocks in the row to the left of the diagonal to a different node, in a round-robin fashion. This approach has two main goals: to evenly distribute the memory load, and to minimize MPI communication (via non-blocking broadcast) of non-local blocks for the LU block updates. In this way, only the blocks above and to the left of the diagonal are broadcasted during the block row and column update.

B. Compute Platform

Computations in Section V-C were carried out on a Dell Precision T7900 workstation, with dual twelve-core Intel Xeon CPUs (E5-2690 v3) at 2.6 GHz, 256 GB of RAM, and dual NVIDIA GTX 1080 Ti GPUs, running Ubuntu Linux.

Computations in Sections V-D and V-E utilized a distributed memory system comprising 25 nodes, each having twelve-core Intel Xeon CPUs (E5-2680 v3) at 2.50GHz (600 total CPUs) and 128 GB RAM, with a 10 gigabit Ethernet network fabric, running Red Hat Enterprise Linux.

C. Conducting Spheres

We first compare the memory and CPU requirements of the single-level ACA and MLACA, in both matrix filling and LU factorization, for perfectly electrically conducting (PEC) spheres of radius a ranging from 1 to 16 λ. Triangle surface meshes were constructed with approximately 12 edges per wavelength, yielding the number of triangles N_T and unknowns N listed in Table I. The target size of top-level basis function groups was $N_g = 2500$ unknowns. For a MLACA of level L, this yields finest-level group sizes of approximately $N_g/2^L$. For the MoM problem, CFIE is used ($\alpha = 0.5$), and $\tau_{ACA} = 5e^{-3}$. This τ_{ACA} was chosen so that the examples fit in available RAM and illustrate the algorithm's efficacy. A much smaller value ($\tau_{ACA} \leq 10^{-4}$) would be required for an accurate solution. In this study, we only present results from MLACA levels up to $L = 3$, as the compression achieved for levels $L \geq 4$ was not significantly greater than for $L = 3$.

Table II summarizes the matrix fill time ($T_{\mathbf{Z}}$, in seconds) and the memory requirements of the MoM matrix $\mathbf{Z}$ ($M_{\mathbf{Z}}$, in GB). Similarly, Table III summarizes the LU factorization time and the memory requirements of the LU matrix ($M_{\mathbf{LU}}$, in GB). To see better the differences, we have summarized in Table IV the percent difference (decrease) in the size of $\mathbf{Z}$ and the LU matrix using MLACA versus the single-level ACA. The

TABLE I
PEC SPHERES: DIMENSIONS, NUMBER OF TRIANGLES AND UNKNOWNS

$a(\lambda)$	1	2	4	8	16
N_T	5120	20480	81920	327680	1310720
N	7680	30720	122880	491520	1966080

TABLE II
PEC SPHERES: MEMORY ($M_{\mathbf{Z}}$, GB), FILL TIME ($T_{\mathbf{Z}}$, SECONDS)

$a(\lambda)$	ACA		$L=1$		$L=2$		$L=3$	
	$M_{\mathbf{Z}}$	$T_{\mathbf{Z}}$	$M_{\mathbf{Z}}$	$T_{\mathbf{Z}}$	$M_{\mathbf{Z}}$	$T_{\mathbf{Z}}$	$M_{\mathbf{Z}}$	$T_{\mathbf{Z}}$
1	.064	2.4	.058	2.6	.055	2.7	.054	2.9
2	.69	6.4	.65	6.7	.62	6.9	.61	7.2
4	3.25	16.7	3.01	19.5	2.88	21.8	2.78	26.4
8	18.6	69.8	17.2	90.5	15.7	120	14.5	178
16	139	569	118	752	99	1109	97	1982

TABLE III
PEC SPHERES: MEMORY ($M_{\mathbf{LU}}$, GB), LU FACTORIZATION TIME ($T_{\mathbf{LU}}$, SECONDS)

$a(\lambda)$	ACA		$L=1$		$L=2$		$L=3$	
	$M_{\mathbf{LU}}$	$T_{\mathbf{LU}}$	$M_{\mathbf{LU}}$	$T_{\mathbf{LU}}$	$M_{\mathbf{LU}}$	$T_{\mathbf{LU}}$	$M_{\mathbf{LU}}$	$T_{\mathbf{LU}}$
1	.062	.4	.057	0.5	.055	0.5	.053	0.5
2	.68	3.8	.65	4.0	.62	4.2	.61	4.4
4	3.34	27.4	3.09	31.8	2.92	30.8	2.81	35.7
8	20.5	569	18.3	589	16.5	615	15.3	1021
16	186	26091	145	26926	127	27740	120	29111

TABLE IV
PERCENT DIFFERENCE (DECREASE) IN $M_{\mathbf{Z}}$ AND $M_{\mathbf{LU}}$: MLACA VERSUS SINGLE-LEVEL ACA

	$M_{\mathbf{Z}}$					$M_{\mathbf{LU}}$				
$a(\lambda)$	1	2	4	8	16	1	2	4	8	16
$L=1$	9.8	5.9	7.6	8.0	16.5	8.4	5.7	7.6	11.5	25.5
$L=2$	15.1	10.2	12.0	17.1	34.0	12.0	9.5	13.4	21.7	38.0
$L=3$	17.0	12.7	15.6	24.7	35.6	15.7	11.8	17.2	28.8	43.4

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAP.2019.2963619, IEEE Transactions on Antennas and Propagation

additional compression obtained using MLACA is substantial, with its effectiveness increasing with both level L and problem size. Of particular note is that for $a \geq 4$, MLACA compresses the LU matrix blocks to a greater degree than it does the $\mathbf{Z}$ blocks.

TABLE V
PERCENT DIFFERENCE (INCREASE) IN FILL TIME $T_{\mathbf{Z}}$ AND LU FACTORIZATION TIME $T_{\mathbf{LU}}$: MLACA VERSUS SINGLE-LEVEL ACA

	$T_{\mathbf{Z}}$			$T_{\mathbf{LU}}$		
$a(\lambda)$	4	8	16	4	8	16
$L=1$	15.5	25.8	27.6	2.5	3.4	3.1
$L=2$	26.5	53.0	63.9	3.9	4.6	6.1
$L=3$	45.0	87.4	107	13.9	8.7	10.9

Table V summarizes the percent difference (increase) in matrix fill time $T_{\mathbf{Z}}$ and LU factorization time $T_{\mathbf{LU}}$. We have omitted results for $a < 4$ as these runs were so short that system overhead and other effects caused large variances in total run time. The increases in $T_{\mathbf{Z}}$ is as expected, with the additional compression steps adding significantly to the run time. However the increases in $T_{\mathbf{LU}}$ are far more moderate, particularly for $a = 16\lambda$ where the increase in run time is less than 11 percent across all L. This suggests that while the MLACA compression of $\mathbf{Z}$ blocks adds significantly to the matrix fill time, the block updates (prior to recompression) consume the most CPU time in the LU factorization.

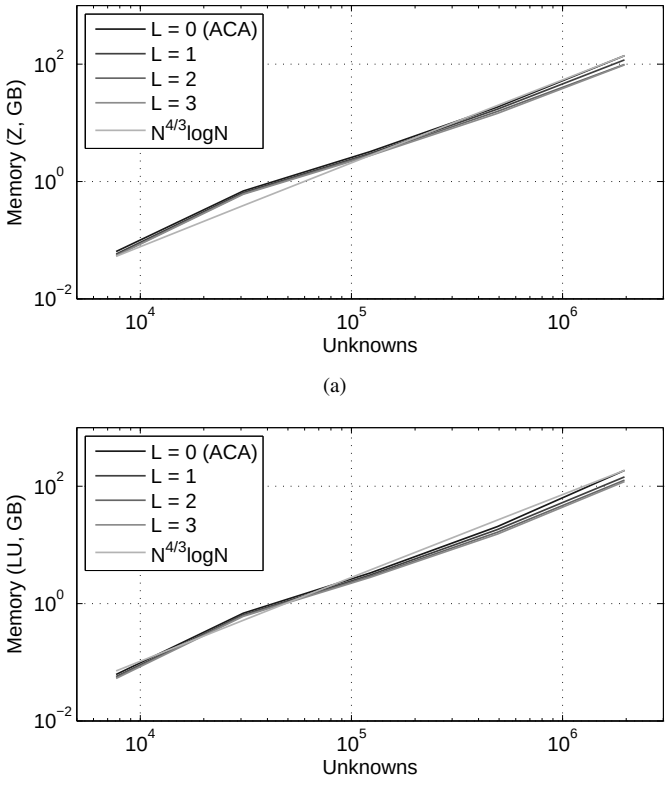


Fig. 6. Complexity of the ACA and MLACA Algorithm for PEC Spheres. (a) $M_{\mathbf{Z}}$, (b) $M_{\mathbf{LU}}$.

In Figure 6 are plotted the memory requirements of $M_{\mathbf{Z}}$ and $M_{\mathbf{LU}}$ versus the number of unknowns. We observe that for the problems considered, the ACA and MLACA have a complexity of approximately $O(N^{4/3}\log N)$. This is consistent with the complexity for a fixed discretization and increasing frequency observed by the authors in [10].

D. Dielectric Sphere

We next consider an 8λ radius dielectric sphere with a permittivity $\epsilon = 2.56$. We use the mesh for the 8λ model from the previous section, however there are now an equal number of electric and magnetic basis functions, for a total of 983040 unknowns. In this case, the PMCHWT formulation [1] is used to solve the MoM problem, and $\tau_{ACA} = 10^{-4}$. In Figure 7 we compare the bistatic RCS obtained from the single-level ACA and MLACA ($L = 3$) to the analytical Mie solution [23]. The results are nearly indistinguishable. Table VI summarizes the memory requirements $M_{\mathbf{Z}}$ and $M_{\mathbf{LU}}$ for the single-level ACA and MLACA ($L \leq 3$). As we have consistently observed in practice for dielectric problems, the size of the LU matrix is approximately double that of $\mathbf{Z}$, however the additional compression afforded by MLACA is maintained across all levels. Table VII summarizes the corresponding matrix fill times $T_{\mathbf{Z}}$ and LU factorization times $T_{\mathbf{LU}}$.

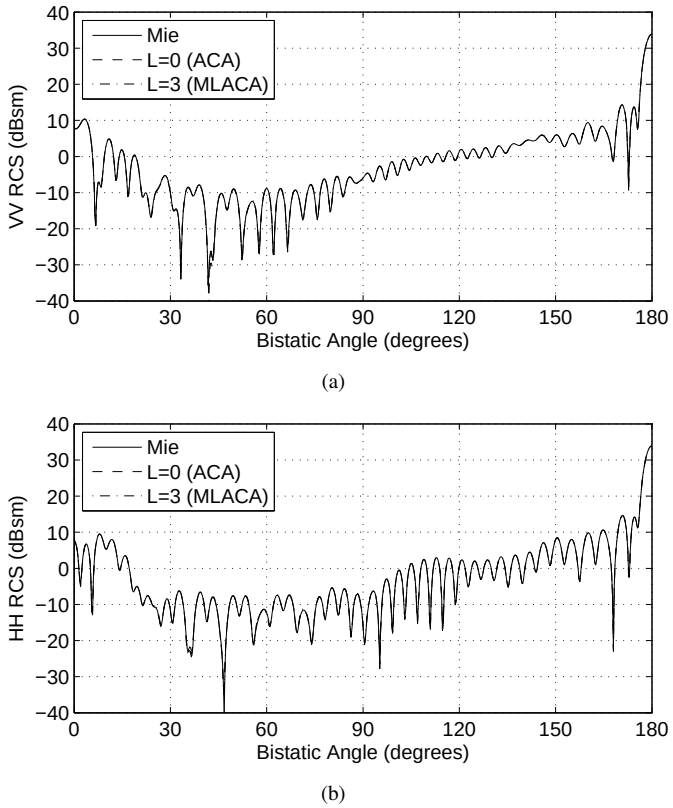


Fig. 7. Bistatic RCS of an 8λ Dielectric Sphere ($\epsilon = 2.56$). (a) VV-Pol RCS, (b) HH-Pol RCS.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TAP.2019.2963619, IEEE Transactions on Antennas and Propagation

TABLE VI
8λ DIELECTRIC SPHERE: MEMORY REQUIREMENTS

$M_{\mathbf{Z}}$ (GB)				$M_{\mathbf{LU}}$ (GB)			
ACA	$L=1$	$L=2$	$L=3$	ACA	$L=1$	$L=2$	$L=3$
141	124	114	109	283	246	221	214

TABLE VII
8λ DIELECTRIC SPHERE: FILL TIME ($T_{\mathbf{Z}}$) AND LU FACTORIZATION TIME ($T_{\mathbf{LU}}$)

$T_{\mathbf{Z}}$ (s)				$T_{\mathbf{LU}}$ (s)			
ACA	$L=1$	$L=2$	$L=3$	ACA	$L=1$	$L=2$	$L=3$
152	402	446	563	4426	4602	4636	5577

E. Cone-Sphere

The last object to be considered is a conducting cone-sphere, whose shape and dimensions are illustrated in Figure 8. It has several grooves cut into it along its length, which will resonate at different frequencies. We compute the monostatic RCS at 9.5, 12.5 and 15.0 GHz for incident angles $0 \leq \theta \leq 180$ degrees, where $\theta = 0$ is nose-on. The angular step size is 0.1 degrees, for a total of 1801 right-hand sides. Three surface meshes were constructed, with a density of approximately 10 edges per wavelength at each frequency. This resulted in models having 505020, 960084, and 1371720 triangles, and 757530, 1440126, and 2057580 unknowns, respectively. For the MoM problem, CFIE is used ($\alpha = 0.5$) and $\tau_{ACA} = 10^{-4}$.

Table VIII summarizes the memory requirements $M_{\mathbf{Z}}$ and $M_{\mathbf{LU}}$ for the single-level ACA and MLACA ($L \leq 3$). MLACA again produces significant gains in compression rate versus ACA. Table IX summarizes the corresponding matrix fill times $T_{\mathbf{Z}}$ and LU factorization times $T_{\mathbf{LU}}$. We note that at 12.5 and 15.0 GHz, the $T_{\mathbf{LU}}$ for each MLACA level is actually less than ACA. This is likely due to reduced communication overhead needed to transmit the smaller MLACA-compressed blocks. In Figure 9 we compare the HH-polarized bistatic RCS obtained from the single-level ACA and MLACA ($L = 3$) to results from our code *Galaxy*, which implements the Body of Revolution Moment Method (MoM-BoR) approach [24]. The comparison is excellent across all angles.

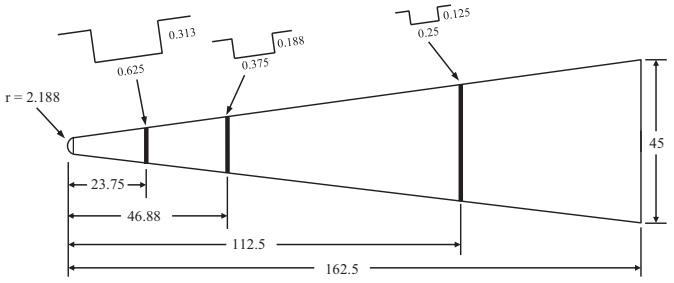


Fig. 8. Cone-Sphere Radar Target (All dimensions in centimeters).

VI. CONCLUSION

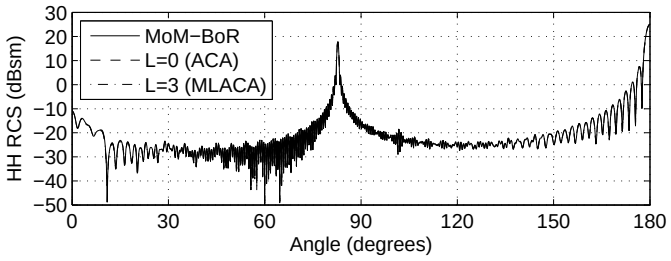
In this paper, the MLACA algorithm was applied to a direct solution of the MoM linear system using LU factorization. It was shown that through careful modification of the original algorithm, the LU block updates and block-RHS solution are carried out in rank-reduced form and are very efficient. Through numerical experiments, it was shown that the off-diagonal LU matrix blocks are highly compressible using MLACA, and that memory requirements of the MLACA-compressed LU matrix scale approximately as $O(N^{4/3} \log N)$, which is consistent with the complexity of the single-level ACA. These experiments have also confirmed that the MLACA retains the accuracy as the single-level ACA for

TABLE VIII
CONE-SPHERE: MEMORY REQUIREMENTS

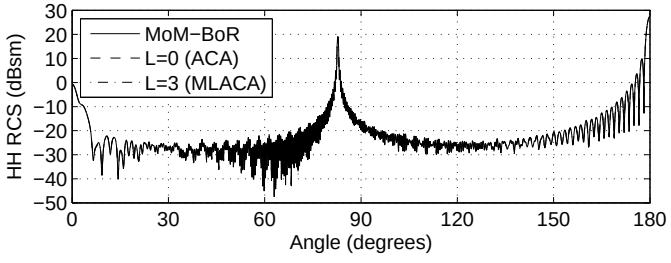
	$M_{\mathbf{Z}}$ (GB)				$M_{\mathbf{LU}}$ (GB)			
f (GHz)	ACA	$L=1$	$L=2$	$L=3$	ACA	$L=1$	$L=2$	$L=3$
9.5	56	49	45	42	76	66	59	55
12.5	149	132	119	112	218	191	170	159
15.0	264	237	215	201	405	357	318	298

TABLE IX
CONE-SPHERE: FILL TIME ($T_{\mathbf{Z}}$) AND LU FACTORIZATION TIME ($T_{\mathbf{LU}}$)

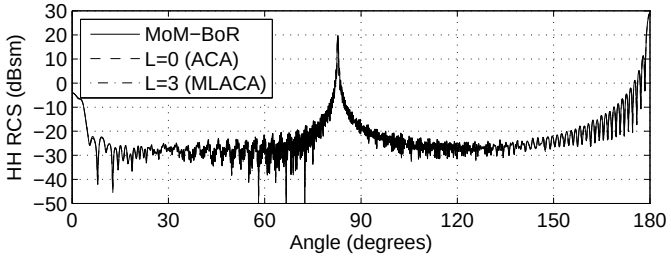
	$T_{\mathbf{Z}}$ (s)				$T_{\mathbf{LU}}$ (s)			
f (GHz)	ACA	$L=1$	$L=2$	$L=3$	ACA	$L=1$	$L=2$	$L=3$
9.5	68	106	128	176	1790	1770	1837	1931
12.5	154	257	329	476	8524	6994	7217	8267
15.0	266	455	596	883	20220	16994	17554	20205



(a)



(b)



(c)

Fig. 9. Cone-Sphere HH-Pol RCS. (a) 9.5 GHz, (b) 12.5 GHz, (c) 15.0 GHz.

problems up to two million unknowns. The additional memory savings afforded by MLACA are substantial, giving it the potential to allow a direct solution of very large scattering and radiation problems on modest consumer-level workstations, as well as highly parallel enterprise and supercomputer-class computer systems. The proposed algorithm is simple and straightforward to implement, and an excellent candidate for implementation on GPUs.

REFERENCES

[1] W. C. Gibson, *The Method of Moments in Electromagnetics*. Taylor and Francis/CRC Press, second ed., 2014.

[2] J. B. Keller, "Geometrical theory of diffraction," *J. Opt. Soc. Amer.*, vol. 52, pp. 116–130, February 1962.

[3] D. Klement, J. Preissner, and V. Stein, "Special problems in applying the physical optics method for backscatter computations of complicated objects," *IEEE Trans. Antennas Propagat.*, vol. 36, pp. 228–237, February 1988.

[4] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Sci.*, vol. 31, pp. 1225–1251, 1996.

[5] Y. Saad, *Iterative Methods for Sparse Linear Systems*. PWS, 1st ed., 1996.

[6] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *J. Comput. Phys.*, vol. 73, pp. 325–348, 1987.

[7] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Propagat. Magazine*, vol. 35, pp. 7–12, June 1993.

[8] J. M. Song, C. C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects," *IEEE Trans. Antennas Propagat.*, vol. 45, pp. 1488–1493, October 1997.

[9] M. Bebendorf, "Approximation of boundary element matrices," *Numer. Math.*, vol. 86, pp. 565–589, June 2000.

[10] K. Zhao, M. N. Vouvakis, and J.-F. Lee, "The adaptive cross approximation algorithm for accelerated method of moments computations of EMC problems," *Electromagnetic Compatibility, IEEE Transactions on*, vol. 47, pp. 763–773, Nov 2005.

[11] J. Shaeffer, "Direct solve of electrically large integral equations for problem sizes to 1 M unknowns," *IEEE Trans. Antennas Propagat.*, vol. 56, pp. 2306–2313, August 2008.

[12] E. Michielssen and A. Boag, "A multilevel matrix decomposition algorithm for analyzing scattering from large structures," *IEEE Trans. Antennas Propagat.*, vol. 44, pp. 1086–1093, August 1996.

[13] J. Tamayo, A. Heldring, and J. Rius, "Multilevel adaptive cross approximation (MLACA)," *IEEE Trans. Antennas Propagat.*, vol. 59, pp. 4600–4608, December 2011.

[14] Y. Liu, H. Guo, and E. Michielssen, "An HSS matrix-inspired butterfly-based direct solver for analyzing scattering from two-dimensional objects," *IEEE Antennas Wireless Propag. Lett.*, vol. 16, p. 11791183, 2017.

[15] H. Guo, Y. Liu, J. Hu, and E. Michielssen, "A butterfly-based direct integral-equation solver using hierarchical LU factorization for analyzing scattering from electrically large conducting objects," *IEEE Trans. Antennas Propagat.*, vol. 65, pp. 4742–4750, September 2017.

[16] H. Guo, Y. Liu, J. Hu, and E. Michielssen, "A butterfly-based direct solver using hierarchical LU factorization for Poggio-Miller-Chang-Harrington-Wu-Tsai equations," *Microw. Opt. Tech. Lett.*, vol. 60, pp. 1381–1387, June 2018.

[17] E. Liberty, F. Woolfe, P. Martinsson, V. Rokhlin, and M. Tygert, "Randomized algorithms for the low-rank approximation of matrices," *Proc. Nat. Acad. Sci. USA*, vol. 104, p. 2016720172, December 2007.

[18] P. Martinsson, V. Rokhlin, and M. Tygert, "A randomized algorithm for the decomposition of matrices," *Appl. Comput. Harmon. Anal.*, vol. 30, pp. 47–68, January 2011.

[19] M. Bebendorf and S. Kunis, "Recompression techniques for adaptive cross approximation," *J. Integral Equations Applications*, vol. 21, pp. 331–357, July 2009.

[20] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, third ed., 1999.

[21] X. Chen, C. Gu, A. Heldring, Z. Li, and Q. Cao, "Error bound of the multilevel adaptive cross approximation (MLACA)," *IEEE Trans. Antennas Propagat.*, vol. 64, pp. 374–378, January 2016.

[22] S. Rao, D. Wilton, and A. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. Antennas Propagat.*, vol. 30, pp. 409–418, May 1982.

[23] G. T. Ruck, ed., *Radar Cross Section Handbook*. Plenum Press, 1970.

[24] R. Mautz and R. Harrington, "H-field, E-field and combined solutions for bodies of revolution," Tech. Rep. Report RADC-TR-77-109, Rome Air Development Center, Griffiss AFB, N.Y., March 1977.

Walton C. Gibson was born in Birmingham, Alabama, USA on December 9, 1975. He received the B.S. degree in electrical engineering from Auburn University in 1996, and the M.S. degree in electrical engineering from the University of Illinois Urbana-Champaign in 1998. He is the author of the *lucernhammer* radar cross section solver code, as well as *The Method of Moments in Electromagnetics*, a textbook geared to graduate-level courses in computational electromagnetics as well as the research community. His professional interests include electromagnetic theory, computational electromagnetics, moment methods, numerical algorithms and parallel computing.