Array Decomposition-Fast Multipole Method for finite array analysis

Rick W. Kindt[1] and John L. Volakis[1]

ElectroScience Laboratory, Department of Electrical and Computer Engineering, Ohio State University, Columbus, Ohio, USA

Received 26 February 2003; revised 24 December 2003; accepted 5 February 2004; published 16 April 2004.

[1] An innovative approach is presented for analyzing finite arrays of regularly spaced elements. We review the recently proposed Array Decomposition Method, which exploits the block-Toeplitz property of regularly spaced arrays for significant storage reduction. To further reduce storage, in this paper we incorporate a multipole expansion to treat distant element interactions. The suggested approach overcomes the matrix storage bottleneck associated with integral equation methods, resulting in fixed and minimal matrix storage for any sized array (on the same order as the storage of a single array element). Hence, fast and rigorous analysis of very large finite arrays can be accomplished with limited resources. *INDEX TERMS:* 0604 Electromagnetics: Antenna arrays; 0644 Electromagnetics: Numerical methods; 0669 Electromagnetics: Scattering and diffraction; *KEYWORDS:* finite array decomposition, Fast Multipole Method, tapered-slot antenna

Citation: Kindt, R. W., and J. L. Volakis (2004), Array Decomposition-Fast Multipole Method for finite array analysis, *Radio Sci.*, *39*, RS2018, doi:10.1029/2003RS002887.

1. Introduction

[2] The Fast Multipole Method (FMM) represents a significant advance in generalized electromagnetic modeling [*Coifman et al.*, 1993]. For general problems, the $O(N \log N)$ memory requirements of multilevel FMM (MLFMM) can make large problems solvable, whereas a conventional approach would be hindered by the $O(N^2)$ storage limitations. The net benefit of conventional FMM is that it makes dense matrices sparse. Thus, for very complex problems, such as large finite arrays with arbitrary and intricate antenna elements, a generalized FMM approach can give significant memory savings. In reducing the matrix storage requirements, FMM transfers the burden to the iterative solution process, increasing the cost of each iteration. Hence, excessive iterations would lead to a negative impact on solution speed, a problem that can easily occur with a lack of effective preconditioning. Moreover, even though conventional MLFMM features $O(N \log N)$ CPU and memory requirements, for increasingly large problems one is still hindered by the matrix storage bottleneck.

[3] The obvious shortcoming of applying a generalized solution method to an array problem is that this approach fails to exploit known redundancies in the problem geometry. In a recent paper [*Kindt et al.*, 2003], a method referred to as the Array Decomposition Method (ADM) was proposed and shown to significantly reduce storage when using the Finite Element-Boundary Integral (FE-BI) method to model array elements. ADM features $O(nm^2)$ memory requirements (n being the number of array elements, m being the number of element unknowns, $N = nm$), which is comparable with FMM for large problems and small m. Further, ADM significantly increases solution speed by applying the fast Fourier transform (FFT) to the integral equation matrix operations, and demonstrates superior convergence rates (over conventional FE-BI) due in part to the symmetric matrix layout having a better disposition towards effective preconditioning schemes [*Kindt et al.*, 2002]. However, like FMM, ADM shares the same linear increase in memory requirements as the array size increases.

[4] In this paper, we present a hybrid approach to finite array-type problems that combines the benefits of array decomposition with the benefits of a multipole expansion for treating distant interactions [*Kindt and Volakis*, 2003a, 2003b]. The analysis method presented here is innovative for several reasons. First, the combination of near-zone array decomposition with multipole

[1]Formerly at Radiation Laboratory, University of Michigan, Ann Arbor, Michigan, USA.

expansion for distant elements has the effect of truncating the near-zone matrix storage. That is, near-zone matrix size is fixed for any sized array of regularly spaced and arbitrary elements. This results in extremely fast matrix fill times, and for all practical purposes, matrix storage requirements are on the same order as the storage of a single array element, namely, $O(m^2)$ compared with $O(n^2m^2)$. Hence, for large problems, storage is dominated by the solution vector and not the matrix. That is, for a problem having an overall solution vector of length N, the total storage reduces to $O(N)$. In addition, since each element of the array can be modeled with an identical unit cell, it is only necessary to compute a single set of Fourier coefficients for basis functions corresponding to the unit cell, as compared to the conventional FMM which requires computation of the coefficients for every array element (approximately n sets). Moreover, the far-zone element interactions (translations) have an inherent Toeplitz property corresponding to the array lattice that allows the FFT to be applied in accelerating the solution process, similar to what is done in the Adaptive Integral Method (AIM) [*Bleszynski et al.*, 1996] and in the FMM-FFT method [*Wagner et al.*, 1997]. This hybrid approach, to be referred to as the Array Decomposition-Fast Multipole Method (AD-FMM), benefits from the same disposition towards effective preconditioning as ADM [*Kindt et al.*, 2003], meaning solution convergence in relatively few iterations. Most importantly, AD-FMM is completely rigorous, providing accurate solutions to large coupling problems with negligible resource requirements. As an example application, with AD-FMM it is possible to excite a single array element and measure the resulting voltages at other ports of the array.

[5] In this paper, we give an overview of how to implement AD-FMM and then compare the theoretical storage and CPU requirements in the context of other methods (conventional FE-BI, FE-BI with MLFMM, and FE-BI with ADM). Subsequently, we examine the benefits of AD-FMM for several practical example problems involving tapered-slot antenna arrays. For a short list of historic and recent methods for array analysis, the interested reader is referred to *Kindt et al.* [2003].

2. Underlying Formulation and Decomposition Method

[6] The presented method can be generalized to any integral equation formulation. Here, we choose to apply the technique to the hybrid FE-BI method, as it allows us to treat inhomogeneous materials and arbitrary element geometries. We represent each individual array element as a closed volume, modeling the inside of the volume with the Finite Element Method (FEM) and discretizing the boundary with surface elements treated via integral equations.

[7] To present the method, let us begin with the generalized FE-BI system given by

$$\begin{bmatrix} A^{II} & A^{IS} & 0 \\ A^{SI} & A^{SS} & B \\ 0 & P & Q \end{bmatrix} \begin{Bmatrix} E^i \\ E^s \\ H^s \end{Bmatrix} = \begin{Bmatrix} b^i \\ b^s \\ b^e \end{Bmatrix}. \tag{1}$$

In (1), the $A^{II,IS,SI,SS}$ and B submatrices are sparse FEM operators, whereas the P and Q sub-matrices are dense, and refer to the BI enclosing the array element. Also, the E and H column vectors refer to the discrete electric and magnetic fields within and on the element boundary. Further, the right-hand side $\{b\}$ vectors refer to internal and external excitations to the system. The details of the FE-BI method are not within the scope of this paper, and the interested reader is referred to *Sheng et al.* [1998] and *Volakis et al.* [1998].

[8] For a general FE-BI or FE-BI with MLFMM approach to a finite array problem, the matrix system will conform to (1). In other words, the matrix is constructed and grouped by operator type. However, this expansion has no inherent symmetry, and preconditioning a large matrix system of this type is not expedient. Consequently, in a recent paper [*Kindt et al.*, 2003] an alternative expansion was suggested for finite array problems. The suggested expansion takes the form

$$\begin{bmatrix} [a]_{11'} & [a]_{12'} & \cdots & [a]_{1n'} \\ [a]_{21'} & [a]_{22'} & \cdots & [a]_{2n'} \\ \vdots & \vdots & \ddots & \vdots \\ [a]_{n1'} & [a]_{n2'} & \cdots & [a]_{nn'} \end{bmatrix} \begin{Bmatrix} \{x\}_1 \\ \{x\}_2 \\ \vdots \\ \{x\}_n \end{Bmatrix} = \begin{Bmatrix} \{b\}_1 \\ \{b\}_2 \\ \vdots \\ \{b\}_n \end{Bmatrix}, \tag{2}$$

where the diagonal terms $[a_{uu'}]$ are full FE-BI coupling systems given by

$$\begin{bmatrix} A^{II}_{uu'} & A^{IS}_{uu'} & 0 \\ A^{SI}_{uu'} & A^{SS}_{uu'} & B_{uu'} \\ 0 & P_{uu'} & Q_{uu'} \end{bmatrix}, \quad u = u', \tag{3}$$

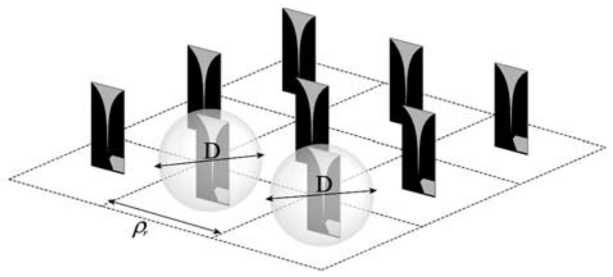

Figure 1. Depiction of element clustering grid.

and the cross terms are the interelement coupling matrices given by

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & P_{uv'} & Q_{uv'} \end{bmatrix}, \ u \neq v'. \tag{4}$$

The unknown vector can be expanded as

$$\{x\}_u = \left\{ \{E^i\}_u \quad \{E^s\}_u \quad \{H^s\}_u \right\}^T, \tag{5}$$

and likewise

$$\{b\}_u = \left\{ \{b^i\}_u \quad \{b^s\}_u \quad \{b^e\}_u \right\}^T. \tag{6}$$

[9] In (2), the system has been recast and grouped by the physics of element interactions. This intuitive layout leads to the strong self-coupling terms being located along the main diagonal of the matrix system with the nearest neighbor coupling terms adjacent to the main diagonal (for consecutively numbered array elements). This matrix restructuring induces a degree of symmetry that (1) does not have. As suggested in *Kindt et al.* [2003], an LU-decomposition of the self-cells $[a_{uu'}]$ along the diagonal of the matrix can be used as a highly effective block-diagonal preconditioner. Perhaps more importantly, this layout results in a block-Toeplitz matrix structure, allowing us to easily reduce the dense integral equation matrix storage from $O(n^2m^2)$ down to roughly $O(nm^2)$ with a simple array decomposition. Moreover, the FFT can be used to significantly speedup the matrix-vector product operations of the iterative system solution.

[10] The material presented thus far accounts for the near-zone array decomposition portion of AD-FMM. This decomposition approach is described in greater detail in *Kindt et al.* [2003]. To further reduce storage requirements and allow for practical array simulation, we next introduce a multipole expansion on the array elements, with clusters conforming to the same grid used for our near-zone decomposition. We then apply the same decomposition principles to the far-zone interaction of the elements.

3. Multipole Expansion

[11] It is assumed that the reader has sufficient background in the Fast Multipole Method, and the interested reader is referred to *Coifman et al.* [1993]. As described in *Kindt et al.* [2003], ADM treats every element of the array as being in the near-zone, resulting in a linear increase of the matrix storage, proportional to $O(nm_{BI}^2)$, where m_{BI} is the number of unknowns for the boundary integrals. However, this rigor is not necessary or prudent, as distant array interactions can be treated with far-zone approximations at little cost to solution accuracy. In the proposed AD-FMM approach, each element of the array is treated as an inclusive cluster or unit cell [*Kindt and Volakis*, 2003a, 2003b]. A regular clustering grid conforming to our array lattice is created and used to expand the basis functions of each array element around the center of the element's lattice position (see Figure 1).

[12] The unit cluster has a maximum diameter of extent that encompasses every basis function of the array element, denoted here as D. The degree to which these cluster diameters overlap determines the overall cost of the near-zone storage. In an array of identical elements, each element will have a unique field solution, but identical basis representations, denoted as $a(\bar{r})$, where $\bar{r}$ is the usual position vector referenced to the center of the array element or lattice grid. These basis functions have an equivalent far-zone representation, which can be calculated via the integral

$$V\left(w, \hat{k}\right) = \int_S a(\bar{r}) e^{i\bar{k}\bar{r}} dS. \tag{7}$$

Again, as all clusters are identical, this is a single calculation used to represent all array elements. In this expression, $w = 1..m$, the basis function index. Note also that $\bar{k} = k_0\hat{k}$, where the quantity k_0 is the free-space wave number and $\hat{k}$ is a single k-space direction in the range $\hat{k} = 1..K$. The number of k-space directions $K \simeq 2L^2$ required for accurate far-zone representation in θ and ϕ has already been considered in the development of FMM, where L is given by [*Coifman et al.*, 1993; *Sertel and Volakis*, 1999]

$$L = k_0 D + \alpha_L \ln(k_0 D + \pi). \tag{8}$$

Here, α_L is chosen to generate the desired accuracy for the application type, but should be between 1 and 10. As usual, it is required that $L > k_0D$, which has a direct effect on the required near-zone storage. It is important to note that the required number of spectral points to represent the element in the far-zone is dependent on the electrical size D of the array element, and has no relation to the complexity of the element features.

[13] To couple distant elements via the FMM, it is necessary to precompute the translation operators to map the currents of any given element of the array onto any other element. If we impose regular spacing of the element clusters and sequential numbering in each dimension, the unique translation operators are defined by the difference in their array numbering, and are given by [*Coifman et al.*, 1993]

$$T\left(q_r,\hat{k}\right) = \frac{k_0}{(4\pi)^2}\sum_{l=1}^{L} i^l(2l+1)h_l^{(1)}(k_0|\bar{\rho}_r|)P_l\left(\hat{k}\hat{\rho}_r\right). \quad (9)$$

In this expression, $h_l^{(1)}$ is a spherical Hankel function of the first kind, P_l is a Legendre function of order l, and $\bar{\rho}_r$ is the vector separating the source and testing array elements. In the method proposed here, each unique vector between source and testing array elements $\bar{\rho}_r$ is given a unique Toeplitz storage index q_r. Consequently, $T(q_r, \hat{k})$ will be a nearly full matrix of the same dimension as the antenna array, plus one additional dimension for $\hat{k}$. The precomputed translation operators are preserved in Toeplitz storage format for later acceleration of the translation/interaction of array elements via the FFT. Because the entire structure has been allocated for the FFT acceleration, the placeholders corresponding to interactions between nearby array elements (handled in the near-zone) are filled with zeros. This introduces some inefficiency, but is slight in the context of large array structures. The FFT acceleration process is described later in this section.

[14] To use (9), it is necessary that L is not larger than $k_0|\bar{\rho}_r|$, otherwise the Hankel function will oscillate, causing inaccuracies in the translation operators. This imposes the primary criterion that determines which elements can be treated as being in the far-zone. Specifically, it must be that

$$R_{near} > L/k_0. \quad (10)$$

Any two array elements separated by less than R_{near} will be treated as being in the near-zone, and thus contribute to matrix storage. While a larger L gives greater accuracy, it also increases R_{near}, the necessary distance between array elements for which the far-zone expansion is applicable.

[15] Given the above discussion, to perform the equivalent of the matrix-vector product using far-zone interactions of the array elements, the following steps are taken. First, the far-zone fields of each array element are generated from the precomputed far-zone basis function expansions and stored in a vector grid via the step

$$s_u\left(\hat{k}\right) = \sum_{w=1}^{m} V^*\left(w,\hat{k}\right)\{x\}_u, \quad (11)$$

where $\{x\}_u$ is the unknown current vector of array element u, $u = 1\ldots n$ in an n element array of arbitrary dimensions. Recall that $\{x\}_u$ contains coefficients corresponding to each basis function $w = 1..m$ of the array element u. For this implementation, $s(u, \hat{k})$ will be a matrix with the same dimensions as the array, plus an additional dimension for $\hat{k}$. In the notation used here, $s_u(\hat{k})$ signifies all k-space fields $\hat{k} = 1..K$ for array element u, whereas $s_{\hat{k}}(u)$ signifies all array element signatures $u = 1..n$ at the single k-space direction $\hat{k}$. The coupling between source and testing elements is then performed (for each k-space direction) via the arbitrary-dimensional convolution

$$g_{\hat{k}}(u) = T_{\hat{k}}(q_r) * s_{\hat{k}}(u). \quad (12)$$

For large systems, this is an expensive operation, but can easily be accelerated with K FFT operations of the same dimension as the finite array. More specifically, given a finite array with *NDIMS* dimensions (e.g., $NDIMS = 1, 2$, or 3, etc.), for each k-space direction, there will exist a translation matrix $T_{\hat{k}}(q_r)$ of dimension $(2n_1 - 1) \times (2n_2 - 1) \times \cdots \times (2n_{NDIMS} - 1)$, where n_d is the number of array elements in dimension d ($d = 1..NDIMS$). For each $\hat{k}$ direction, we precompute the Fourier transform of $T_{\hat{k}}(q_r)$ using an FFT of the same dimension, to give $\tilde{T}_{\hat{k}}(q_r)$. For each matrix-vector product operation then, the convolution in (12) is replaced with $2K$ FFT operations on $s_{\hat{k}}(u)$ (forward and inverse) of a size matching $\tilde{T}_{\hat{k}}(q_r)$, plus $K\prod_{d=1}^{NDIMS}(2n_d - 1)$ point-by-point multiplications between $\tilde{T}_{\hat{k}}(q_r)$ and $\tilde{s}_{\hat{k}}(q_r)$ (the FFT of $s_{\hat{k}}(u)$). The overall savings of this alternative solution approach is proportional to the size of the array. The coupling to each testing basis of element u (from other array elements in the far-zone) is then distributed via incoming plane waves as

$$\{b\}_u = \int_{\Omega} V\left(w,\hat{k}\right)g_u\left(\hat{k}\right)d\Omega, \quad (13)$$

where $d\Omega$ is the incremental portion of the solid angle ($\sin\theta d\theta d\phi$). Thus, $\{b\}_u$ represents the portion of the

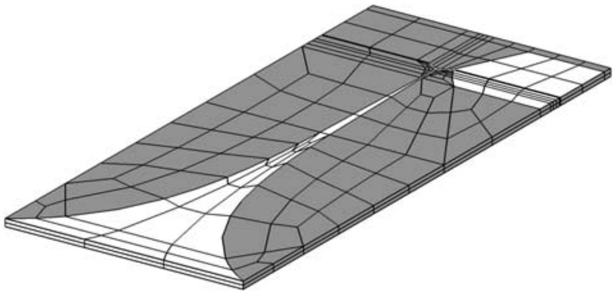
Figure 2. Tessellation of a tapered-slot antenna element.

matrix-vector product of the operation in (2) carried out in the far-zone. This result is simply added to the near-zone contribution of the matrix-vector product operation to give the overall contribution. The near-zone interaction of nearby elements and interaction of basis functions within each of the elements are performed via a standard matrix-vector multiplication. We remark that the storage expense of the near-zone terms is not too excessive when the Toeplitz property of the array decomposition is used. After the FMM expansion is applied, many of the array interactions are no longer carried out in the near-zone. The remaining unique near interaction terms can be found in a tight cluster in the upper left-hand corner of (2). Because of the block-Toeplitz property of the near-zone matrix, this allows the near interactions to approximately be described by the block convolution operation

$$\left[[a]_{21'}\ [a]_{11'}\ [a]_{12'}\right] * \{x\}^T = \{b\}^T. \tag{14}$$

From this, it can be seen that the near zone storage requirement for an entire array is within an order of magnitude of the storage required for a single array element, $[a_{11'}]$. Typically, for $\lambda_0/2$ element spacing, one might expect near terms from one or two neighboring elements in each dimension. However, for elongated array elements, the near-zone influence will increase. We also remark that because of the truncated near-zone storage, it is not possible to apply the FFT in accelerating the near-zone interactions (as is done in standard ADM). However, since the number or near-zone terms is now small, there is no advantage in using the FFT.

4. Storage Cost and Performance Analysis

[16] In this section, we explicitly evaluate the storage and computational cost of AD-FMM and compare the hybrid method with other methods. We evaluate the two costs (storage and computation) separately, since it is insightful to highlight the differences between the various approaches. Consider a volumetric antenna array element of arbitrary materials and shape. The element is modeled as having a bounding surface S and a volume V (see Figure 2). With FE-BI, the array element is subdivided into cells aligned along faces and edges. In our case, we tessellate the volume V with curvilinear hexahedral cells, which by default divide the surface S into curvilinear quadrilateral patches. This meshing procedure is depicted in Figure 2 for an antenna element used in finite arrays (in this case, a tapered-slot antenna).

[17] Let us suppose that upon tessellation and assignment of boundary conditions there are m_{FEM} FEM edges or unknowns and m_{BI} boundary integral unknowns for each antenna element. Roughly speaking, we will have $O(m_{FEM})$ FEM storage and $O(m_{BI}^2)$ boundary integral storage in a conventional FE-BI formulation for this single element. Now let us consider an arbitrary array of n antennas, with the total number of FEM unknowns equal to $N_{FEM} = nm_{FEM}$, and the total BI unknowns equal to $N_{BI} = nm_{BI}$. For a conventional FE-BI approach, we find that we have $O(nm_{FEM})$ FEM storage, and $O(n^2 m_{BI}^2)$ boundary integral matrix storage.

[18] Next, let us consider a generalized FMM expansion of the same array problem, again implemented for FE-BI. Like the conventional FE-BI, FMM will have $O(nm_{FEM})$ FEM storage for the finite array. Regarding the integral equation terms, FMM reduces the problem storage down to $O(nm_{BI}\ \log(nm_{BI}))$ for a multilevel implementation.

[19] For ADM, since the method exploits the repeatability of the array elements, the FEM storage is only $O(m_{FEM})$, the same as for a single element. Thus, the FEM storage is n times less than that of conventional FE-BI or even FE-BI with FMM. For the boundary integral equation terms, the storage is $O(nm_{BI}^2)$, which is typically larger than the corresponding FMM storage for large array elements. Though it may require slightly more BI storage, ADM has the advantage that matrix fill times are considerably faster than FMM because of much lower overhead. As will be shown later, the same is true for solution times.

[20] For AD-FMM, like standard ADM, the FEM storage is merely $O(m_{FEM})$, because the decomposition recognizes that each element has the same FEM coefficients. However, unlike FMM or ADM, both of which experience linear increase in the BI matrix storage, AD-FMM has only $O(m_{BI}^2)$ integral equation coefficient storage, i.e., the same order of magnitude as that of a single element. This is remarkable because it will allow solution of very large finite arrays using rigorous means. However, this is with some caveats, because additional overhead exists for the AD-FMM approach to finite arrays verses the simpler analysis of a single array element. For example, it is necessary to store the far-

Table 1. Storage Cost and Performance Analysis of a Finite Array Problem

	Storage Cost FEM	Storage Cost BI	Computational Cost - FEM	Computational Cost - BI
Conventional FE-BI	$O(nm_{FEM})$	$O(n^2m_{BI}^2)$	$O(nm_{FEM})$	$O(n^2m_{BI}^2)$
Conventional MLFMM	$O(nm_{FEM})$	$O(nm_{BI}\log(nm_{BI}))$	$O(nm_{FEM})$	$O(nm_{BI}\log(nm_{BI}))$
ADM	$O(m_{FEM})$	$O(nm_{BI}^2)$	-	$O(nm_{BI}^2 + m_{BI}n\log(\mathrm{n}))$
AD-FMM	$O(m_{FEM})$	$O(m_{BI}^2)$ (small array) $O(nm_{BI})$ (large array)	-	$O(nm_{BI}^2 + Kn\log(\mathrm{n}))$

zone signatures of the array element for AD-FMM. Fortunately, for an array of identical elements, the signatures are identical as well, and only one set needs to be stored. It is also necessary to precompute the translation operators for interacting the elements in the far-zone. However, as the clusters are aggregated to a regular grid, the translation operators have a Toeplitz property with the required storage of $O(nK)$, K being the number of points or k-space directions used to represent the element signatures in the far-zone. Upon implementation, for very large problems the memory requirements for storing solution vectors, excitation vectors, and solver work vectors will exceed the storage of the translation operators, assuming that the number of far-zone directions K will be less than the number of surface unknowns per cluster, or $K < m_{BI}$. Hence, the total storage requirements for larger problems will be proportional to the length of the overall solution vector, or $O(n(m_{FEM} + m_{BI})) = O(N)$.

[21] A performance analysis on the FEM portion of the matrix-vector product reveals that both conventional FE-BI and FMM have an $O(nm_{FEM})$ cost. However, both ADM and AD-FMM presolve or precondition the FEM portions of the overall matrix system prior to the iterative solution process [*Kindt et al.*, 2003]. Iterations over FEM matrices are typically costly due to the conditions of these systems. For ADM and AD-FMM, this cost is avoided entirely.

[22] The computational cost of the BI portion of the matrix-vector product for conventional FE-BI is proportional to its storage cost of $O(n^2m_{BI}^2)$. The same principle applies to MLFMM, which has a computational cost of $O(nm_{BI}\log(nm_{BI}))$. However, we would like to remark that this estimate does not reflect the cost of excessive iterations. The computational cost of ADM is based on m_{BI} FFT operations of cost $n\log n$, plus n matrix-vector products with m_{BI}^2 operations. This gives a combined total cost of $O(nm_{BI}^2 + m_{BI}n\log(n))$. Though it is hard to make a direct comparison, in practice there is very little overhead for ADM, resulting in significantly faster solution times as compared to FMM or FE-BI for finite arrays. This can be directly observed in the results of the next section.

[23] By comparison, AD-FMM has a near-zone computational cost of $O(nm_{BI}^2)$, plus a far-zone cost of roughly $O(Kn\log(n))$. For large problems, the far-zone cost is largely dominated by the translation operations, which can be reduced to $O(n\log n)$ operations using the FFT. In the same way that the FEM operations can be precomputed, it is also possible to avoid a large fraction of the BI matrix-vector product cost for the near-zone interactions using block-diagonal preconditioning. The fractional savings depends on the array element size and spacing. For example, in a case where the element spacing is larger than the array element diameter, the near-zone CPU cost is reduced to zero, as the near-zone interactions can be entirely precomputed. A cursory glance shows that the computational cost of ADM and AD-FMM are on par. Assuming that $K < m_{BI}$, one may expect the solution times for AD-FMM to be faster than ADM. However, due to the additional overhead cost associated with AD-FMM, whether or not AD-FMM is faster will be implementation dependent. The storage cost and performance analysis of the various methods discussed here are summarized in Table 1.

5. Results

[24] In this section, we evaluate the storage cost and performance of the various methods we discussed up to this point using a specific array example. The analysis is carried out on planar arrays of increasing size ranging from 3×3 to 64×64. To form the array, the antenna element (see Figure 2) is placed on a regularly spaced grid of 6.25 centimeters in x and y – exactly $\lambda_0/2$ at 2.4 GHz, the frequency used in this analysis. The element is a double-sided exponentially tapered-slot antenna, fed via stripline and matched with a double-Y balun (not depicted here). The element dimensions are $11.45 \times 5.0 \times 0.1524$ cm. For the evaluation cases, this element is modeled with $m_{FEM} = 805$ and $m_{BI} = 892$ unknowns, based on the tessellation depicted in Figure 2. We remark that modeling structures of this type is challenging due to the detailed feed characteristics and the electrically large element size, as it can easily lead to edge ratios of 50 to 1. For reference, the storage cost of this single element is about 14MB.

[25] To begin, we compare the E-plane and H-plane patterns for the 5×5 array shown in Figure 3. The corresponding patterns for all methods are shown in

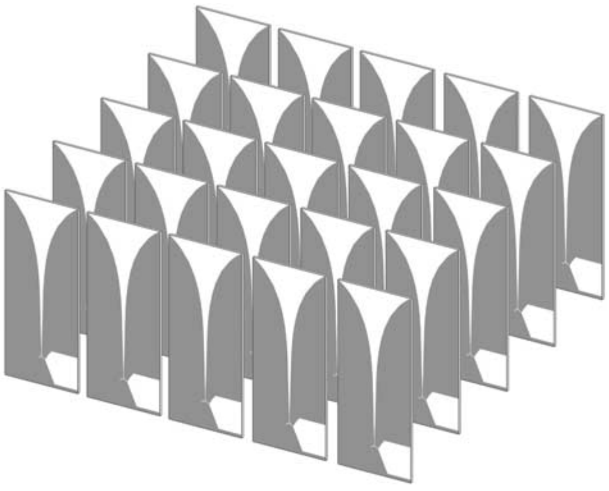

Figure 3. The 5 × 5 array used for consistency comparisons.

Figure 4, and it can be seen that the results from all methods are nearly identical. While a 5 × 5 array may seem small, for this electrically large element a MoM or conventional FE-BI solution requires roughly 8.5GB of matrix storage (apart from the challenges associated with the poor conditioning of the matrix). For equivalent comparisons, the analysis was performed on a 800MHz Itanium with 12GB of RAM.

[26] Of particular interest in this comparison is the required matrix storage for each problem, the matrix fill time, the number of required iterations for solution, as well as the full iterative solution time for each of the methods. Each figure of merit has its own virtues. The storage requirements are a measure of how well the method can be implemented on various platforms, in the sense that someone with no memory constraints may opt to choose the method with the fastest solution times. The matrix fill time is significant as well, as this affects the overall solution time. The number of required iterations reflects on the matrix conditioning as the effect of the employed matrix preconditioning method, whereas the CPU time per iteration reflects directly on the CPU speed of the method. The results for the various figures of merit on several example arrays of different sizes are summarized in Table 2.

[27] For many of the test cases, the more expensive methods (FE-BI, MLFMM) cannot solve the problems with the allotted resources (or time constraints), and the required resources were projected where possible. For conventional FE-BI and FE-BI with MLFMM, it is nearly impossible to project the solution times. Preconditioning is difficult for these large systems, and the number of iterations for convergence is very large (even if the system converged at all). The results for AD-FMM given in Table 2 speak for themselves. AD-FMM is superior to all other methods compared here for finite array analysis. In all test cases, a block-diagonal preconditioner was employed with the BICGSTAB(L) iterative solver, having the equivalent of eight matrix-vector product operations per iteration [*Sleijpen and Fokkema*, 1993]. We remark that the overall storage cost for AD-FMM consists of the matrix storage, the unknown and excitation vector storage, the far-zone basis representations and the precomputed translation operators. While the 64 × 64 array required only 193MB for near-zone matrix storage, a total storage of 432MB was required for carrying out the solution (not counting solver work vectors). This particular problem has crossed the size threshold beyond which unknown and excitation storage exceeds near-zone matrix storage. As a conclusion, we performed here the rigorous analysis of a 7-million unknown problem using less than half a gigabyte of total storage. The mere fact of being able to rigorously solve such large systems in a reasonable

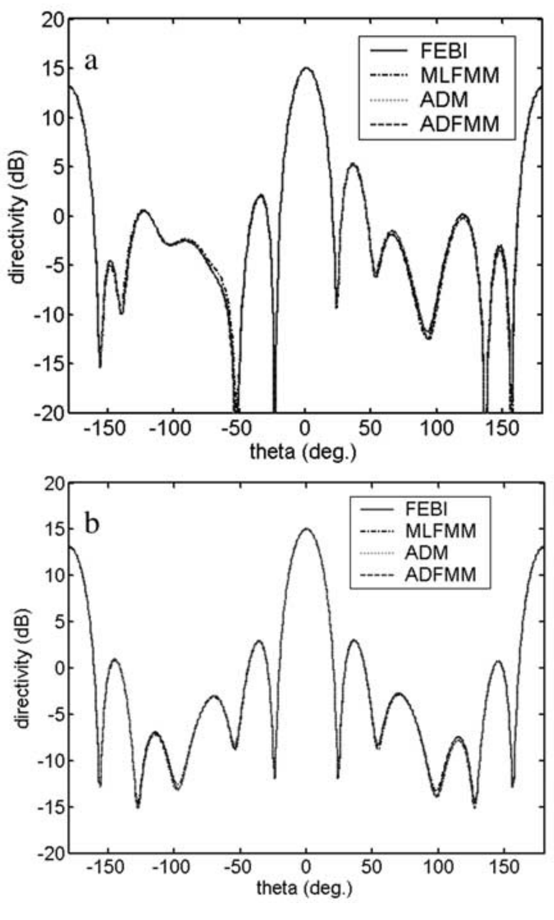

Figure 4. Pattern comparisons of each method for the 5 × 5 array. (a) E-plane and (b) H-plane.

Table 2. Comparison of a Finite Array Problem

		Array Size				
		3 × 3	5 × 5	16 × 16	32 × 32	64 × 64
Unknowns		15,273	42,425	434,432	1,737,728	6,950,912
Matrix Storage Requirements	FE-BI	1.1GB	8.5GB	868TB	13PB	217PB
	MLFMM	162MB	502MB	5.3GB	21GB	88GB
	ADM	193MB	612MB	7.2GB	29GB	121GB
	AD-FMM	193MB	193MB	193MB	193MB	193MB
Matrix Fill-Time	FE-BI	24m	3h	14d[a]	218d[a]	9.5y[a]
	MLFMM	5m	18m	6h	28h[a]	5d[a]
	ADM	8m	25m	5h	20h[a]	3d[a]
	AD-FMM	7m	7m	7m	7m	7m
Iterations	FE-BI	6	108	-	-	-
	MLFMM	6	69	-	-	-
	ADM	2	4	19	62[a]	100[a]
	AD-FMM	2	4	19	62	100
Iterative Solution Time	FE-BI	2m	1h	-	-	-
	MLFMM	1m	51m	-	-	-
	ADM	6s	27s	42m	7h[a]	25h[a]
	AD-FMM	10s	1m	1h	17h	2d
Total Storage Cost	FE-BI	1.1GB	8.5GB	868TB	13PB	217PB
	MLFMM	175MB	536MB	5.6GB	23GB	94GB
	ADM	194MB	613MB	7.2GB	29GB	121GB
	AD-FMM	200MB	201MB	214MB	258MB	432MB
Total Solution Time	FE-BI	26m	4h	-	-	-
	MLFMM	6m	1h	-	-	-
	ADM	9m	26m	6h	1d[a]	4d[a]
	AD-FMM	8m	9m	1h	17h	2d

[a]Estimated results.

amount of time is significant in itself. What is not shown in the table is the overhead time associated with the FMM based methods necessary to calculate and fill the far-zone operators. For FE-BI with MLFMM, this cost can be quite high. However, for AD-FMM the overhead cost is quite low (due to the hybrid decomposition approach).

[28] Finally, we remark that the matrix preconditioning for the array decomposition methods is quite good, as evidenced by the low number of iterations (to achieve 1% solution error). As mentioned, good preconditioning is critical for iterative solution methods, especially for FMM methods that shift computational burden to the iterative process.

6. Conclusion

[29] The main theme of this work was exploitation of known geometrical redundancies and the decomposition of these redundancies into reusable cells where the Toeplitz property of the Green's function can be used for reduced storage and accelerated solution. The concept applies also to many simpler problems, such as large flat or smoothly curving surfaces (namely, a cylindrical airplane fuselage). The idea we would like to impress in the mind of the reader is that while a generalized solution approach is insensitive to geometric restrictions, explicit decomposition of a problem into identical cells (when possible) is extremely advantageous in terms of storage reductions and solution speeds. In combination with a multipole expansion method, array decomposition leads to remarkable speed-ups for finite array-type problems.

[30] The method presented here is a major step for rigorous analysis of large finite arrays without the restrictions of matrix storage limitations. It is understood that for very large array elements or closely packed arrays of elongated elements, the near-zone storage can potentially become prohibitively large. This drawback can be circumvented by decomposing the array elements into smaller clusters using intraelement decomposition. However the concept of multidimensional decomposition presents further challenges to be explored in future work.

[31] As a stand-alone method, AD-FMM has limited utility. The advantage of this method is realized when implemented for multiple systems. In other words, finite arrays can be modeled in their actual environment, which may be on the surface of a vehicle or in the presence of other structures. This hybrid multicell approach to real-world coupling problems has even greater promise [*Kindt and Volakis*, 2003a, 2002b].

[32] It is important to consider the ramifications of using methods that are not limited by matrix storage. Typically, problems with large matrix storage are split up

using distributed memory and analyzed in pieces on multiple machines, but the full solution vector is present on all nodes. From the analysis presented here, we can conclude that AD-FMM has the potential of analyzing problems for which it is not possible to store the entire solution vector on a single machine with 32-bit memory addressing (making the present distributed memory model obsolete). That is, using AD-FMM, it is possible to analyze problems with 100 million or more unknowns, requiring over two gigabytes to store the solution and excitation vector alone using complex(8) data types (exceeding the limit of 32-bit addressing). The advent of this type of problem and larger ones will bring about the necessity of a new paradigm in distributed processing methods, a challenge to be explored in future work.

[33] **Acknowledgments.** The authors would like to thank Dr. Kubilay Sertel for providing the original FE-BI and MLFMM formulations that were used in these comparisons. Dr. Sertel provided expert technical advice on FMM, and in the past served as a mentor to Dr. Kindt.

References

Bleszynski, E., M. Bleszynski, and T. Jaroszewicz (1996), AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems, *Radio Sci.*, *31*, 1225–1251.

Coifman, R., V. Rokhlin, and S. Wandzura (1993), The Fast Multipole Method for the wave equation: A pedestrian prescription, *IEEE Antennas Propag. Mag.*, *35*, 7–12.

Kindt, R., and J. L. Volakis (2003a), The Array Decomposition-Fast Multipole Method, paper presented at Antennas and Propagation Society International Symposium, Inst. of Electr. and Electron. Eng., Columbus, Ohio.

Kindt, R., and J. L. Volakis (2003b), A multi-cell array decomposition approach to composite finite array analysis, paper presented at Antennas and Propagation Society International Symposium, Inst. of Electr. and Electron. Eng., Columbus, Ohio.

Kindt, R., K. Sertel, E. Topsakal, and J. L. Volakis (2002), A domain decomposition of the Finite Element-Boundary Integral Method for finite array analysis, paper presented at Annual Review of Progress in Applied Computational Electromagnetics, Appl. Comput. Electromagn. Soc., Monterey, Calif.

Kindt, R. W., K. Sertel, E. Topsakal, and J. L. Volakis (2003), Array Decomposition Method for the accurate analysis of finite arrays, *IEEE Trans. Antennas Propag.*, *51*(6), 1364–1372.

Sertel, K., and J. L. Volakis (1999), Effects of the Fast Multipole Method (FMM) parameters on RCS computations, paper presented at Antennas and Propagation Society International Symposium, Inst. of Electr. and Electron. Eng., Orlando, Fla.

Sheng, X. Q., J. M. Jin, J. M. Song, C. C. Lu, and W. C. Chew (1998), On the formulation of hybrid finite-element and boundary-integral methods for 3D scattering, *IEEE Trans. Antennas Propag.*, *46*(3), 303–311.

Sleijpen, G. L. G., and D. R. Fokkema (1993), BICGSTAB(L) for linear equations involving unsymmetric matrices with complex spectrum, *Electron. Trans. Numer. Anal.*, *1*, 11–32.

Volakis, J. L., A. Chatterjee, and L. C. Kempel (1998), *Finite Element Method for Electromagnetics*, IEEE Press, Piscataway, N. J.

Wagner, R. L., J. Song, and W. C. Chew (1997), Monte Carlo simulation of electromagnetic scattering from two-dimensional random rough surfaces, *IEEE Trans. Antennas Propag.*, *45*(2), 235–245.

R. W. Kindt and J. L. Volakis, ElectroScience Laboratory, Ohio State University, Columbus, OH 43212-1191, USA. (kindt.2@osu.edu; volakis.1@osu.edu)