SIAM J. MATRIX ANAL. APPL.
Vol. 28, No. 3, pp. 603–622

A FAST *ULV* DECOMPOSITION SOLVER FOR HIERARCHICALLY SEMISEPARABLE REPRESENTATIONS*

S. CHANDRASEKARAN†, M. GU‡, AND T. PALS†

Abstract. We consider an algebraic representation that is useful for matrices with off-diagonal blocks of low numerical rank. A fast and stable solver for linear systems of equations in which the coefficient matrix has this representation is presented. We also present a fast algorithm to construct the hierarchically semiseparable representation in the general case.

Key words. fast multipole method, hierarchically semiseparable, fast algorithms, orthogonal factorizations

AMS subject classification. 65F05

DOI. 10.1137/S0895479803436652

1. Introduction.

In this paper we consider a representation of structured dense matrices that we term *hierarchically semiseparable* (HSS). This representation is a direct generalization of the one presented in [3]. It is also a special case of the FMM (fast multipole method) representations [20, 2, 29, 30, 31]. It has also been discussed as H^2 matrices in [24].

This representation is useful for matrices characterized by a hierarchical low numerical rank structure in the off-diagonal blocks. Examples of such matrices are shown in Figure 1. The matrix in Figure 1(a) is obtained,[1] for example, for the matrix $[\log|x_i - x_j|]$, where $0 \le x_i < x_{i+1} \le 1$. Similarly the matrix in Figure 1(b) is obtained for the matrix $[\log|\sin\pi(x_i - x_j)|]$. This class of matrices arises frequently in the numerical solution of partial differential and integral equations.

This work arose in an effort to stabilize the fast solver presented in [30, 31]. Our initial efforts in this direction were presented in [5, 6, 7, 8, 9, 27, 28]. During this time we learned about some work in linear time-varying systems theory [13], and other independent work [17, 15, 16, 23, 24], that encouraged us to generalize our ideas and present them in a more algebraic framework [3]. The corresponding technical report [4] is more comprehensive and will give some indication to the reader of how far the methods presented in this paper can be taken.

However, the FMM [20, 2, 29] is our most direct motivation for this work. In fact, the ideas presented here can be viewed as a stable approach to a fast inverse multipole method. There has been some significant work in this regard in the computational electromagnetics literature [1, 12, 19, 22, 26, 30, 31]. The method presented here is the first stable fast solver. In addition it also presents an algebraic generalization.

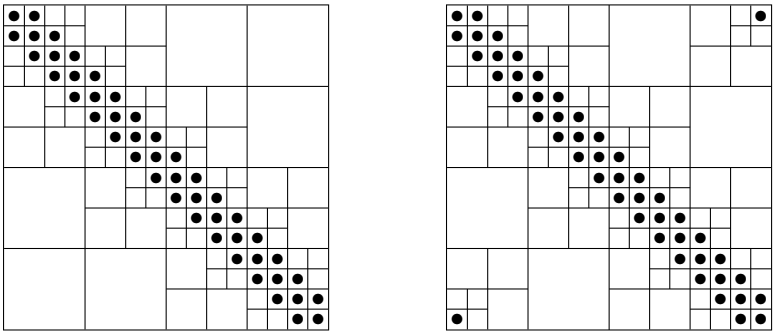
For applications of the fast solver we currently refer to [30, 31] and [4]. However, the applications are much wider than indicated in these references. Some of these will be presented in forthcoming papers.

*Received by the editors October 23, 2003; accepted for publication (in revised form) by V. Mehrmann October 25, 2005; published electronically August 25, 2006.
http://www.siam.org/journals/simax/28-3/43665.html

†Electrical and Computer Engineering Department, University of California, Santa Barbara, CA 93106-9560 (shiv@ece.ucsb.edu, tim@kipling.ece.ucsb.edu).

‡Mathematics Department, University of California, Berkeley, CA (mgu@math.berkeley.edu).

[1] In both cases the particular choice of the diagonal entries is not important.



(a) $\log|x-y|$ on a straight line.

(b) $\log|x-y|$ on a circle.

FIG. 1. *Submatrices labeled with a black dot are full-rank. All other submatrices have low numerical rank.*

For completeness we also present an algorithm to construct a numerical HSS representation of a general matrix. This algorithm requires $O(n^2)$ flops and $O(n)$ space. It is similar to the sequentially semiseparable (SSS) construction algorithm presented in [3, 13]. A construction algorithm for H^2 matrices is presented in [25]. The algorithm we present also ensures that the computed HSS representation satisfies some properties required to enable stability in the solver. We also present two cases where the construction can be carried out in $O(n)$ flops.

2. HSS representation. The representation is identical to the one presented in [30, 31], except that we view it more generically.

The HSS representation is a hierarchical representation that is based on a recursive row and column partitioning of the matrix. For example, for a 2×2 block partitioning of the matrix A its HSS representation is given by

$$A = \begin{pmatrix} D_{1;1} & U_{1;1}B_{1;1,2}V_{1;2}^H \\ U_{1;2}B_{1;2,1}V_{1;1}^H & \end{pmatrix},$$

where the subscripted D, U, V, and B matrices are in the representation. To see the recursive hierarchical nature we consider a block 4×4 partitioning of A and the resultant two-level HSS representation:

$$A = \begin{pmatrix} \begin{pmatrix} D_{2;1} & U_{2;1}B_{2;1,2}V_{2;2}^H \\ U_{2;2}B_{2;2,1}V_{2;1}^H & D_{2;2} \end{pmatrix} & \begin{pmatrix} U_{1;1}B_{1;1,2}V_{1;2}^H \end{pmatrix} \\ \begin{pmatrix} U_{1;2}B_{1;2,1}V_{1;1}^H \end{pmatrix} & \begin{pmatrix} D_{2;3} & U_{2;3}B_{2;3,4}V_{2;4}^H \\ U_{2;4}B_{2;4,3}V_{2;3}^H & D_{2;4} \end{pmatrix} \end{pmatrix}.$$

We first observe that only the two diagonal blocks $D_{1;1}$ and $D_{1;2}$ from the one-level HSS representation have been partitioned at the second level, each of them seemingly assigned their own HSS representations. However, that view is slightly misleading. In fact, in the two-level HSS representation of the matrix A, we do not store the matrices $U_{1;i}$ and $V_{1;i}$ for $i = 1, 2$. Rather we store only the $U_{2;i}$ and $V_{2;i}$ for $i = 1, 2, 3, 4$ and the translation operators $W_{2;i}$ and $R_{2;i}$ for $i = 1, 2, 3, 4$, which can be used to reconstruct the missing $U_{1;i}$ and $V_{1;i}$ via the defining relations

$$U_{1;1} = \begin{pmatrix} U_{2;1}R_{2;1} \\ U_{2;2}R_{2;2} \end{pmatrix},$$

$$U_{1;2} = \begin{pmatrix} U_{2;3}R_{2;3} \\ U_{2;4}R_{2;4} \end{pmatrix},$$
$$V_{1;1} = \begin{pmatrix} V_{2;1}W_{2;1} \\ V_{2;2}W_{2;2} \end{pmatrix},$$
$$V_{1;2} = \begin{pmatrix} V_{2;3}W_{2;3} \\ V_{2;4}W_{2;4} \end{pmatrix}.$$

These translation operators are an integral part of the FMM representation and literature, and their use in getting linear complexity algorithms is well known.

In general in a multilevel HSS representation the diagonal blocks at the ith level are labeled $D_{i;j}$. The $U_{i;j}$ at the lowest levels are used in conjunction with the translation operators $R_{i;j}$ at that level to reconstruct the $U_{i-1,j}$'s at the higher levels via

$$U_{i-1;j} = \begin{pmatrix} U_{i;2j}R_{i;2j-1} \\ U_{i;2j}R_{i;2j} \end{pmatrix}. \tag{1}$$

Similarly for $V_{i;j}$ we have

$$V_{i-1;j} = \begin{pmatrix} V_{i;2j}W_{i;2j-1} \\ V_{i;2j}R_{i;2j} \end{pmatrix}. \tag{2}$$

At every level only the diagonal blocks are eligible for partitioning. The off-diagonal blocks in the upper-triangular part of the ith level are of the form $U_{i;2j-1}B_{i;2j-1,2j}V_{i;2j}^H$ and in the lower-triangular part of the form $U_{i;2j}B_{i;2j,2j-1}V_{i;2j-1}^H$. Therefore, the complete HSS representation of the matrix A consists of the $D_{i;j}$, $U_{i;i}$, and $V_{i;j}$ at the lowest levels along with $B_{i;2j,2j-1}$, $B_{i;2j-1,2j}$, $R_{i;j}$, and $W_{i;j}$ at every level.

In the FMM literature it has been convenient to use a (binary in this case) tree on which all these matrices can be represented. In this notation the root of the tree corresponds to the whole matrix; the two children of the root correspond to the two row (and column) partitions, and so on. In Figure 2 we depict the HSS tree (also called a merge tree) for a uniform three-level HSS representation. We will refer to the ith node at the k-level of the tree as Node(k, i).

It should be observed that every matrix has an HSS representation. However, HSS representations are useful only when the translation operators are small compared to the size of the original matrix. These representations can be constructed in $O(n^2)$ flops and $O(n)$ space; see [10, 24]. In special cases these representations can be constructed in $O(n)$ flops. The FMM literature is rife with such results. Some other interesting instances can also be found in [10].

In this paper we restrict ourselves to HSS trees that are (almost) complete binary trees. In a future paper we will generalize our methods to incomplete binary trees.

3. Fast multiplication. In this section, for the convenience of the reader, we present the standard FMM algorithm for multiplying a matrix in HSS form with a regular vector (or unstructured dense matrix). In particular consider the matrix-vector product $z = Ab$, where A is in HSS form, with $K + 1$ levels in its HSS tree, and m_i indices in Node(K, i). Let $(\, b_{k;i}\,)$ denote a block row partitioning of b such that $b_{k;i}$ has the rows whose indices belong to Node(k, i). We partition z similarly.

We begin by observing that we need to do the multiplication

$$U_{1;1}B_{1;1,2}V_{1;2}^H b_{1;2} \tag{3}$$

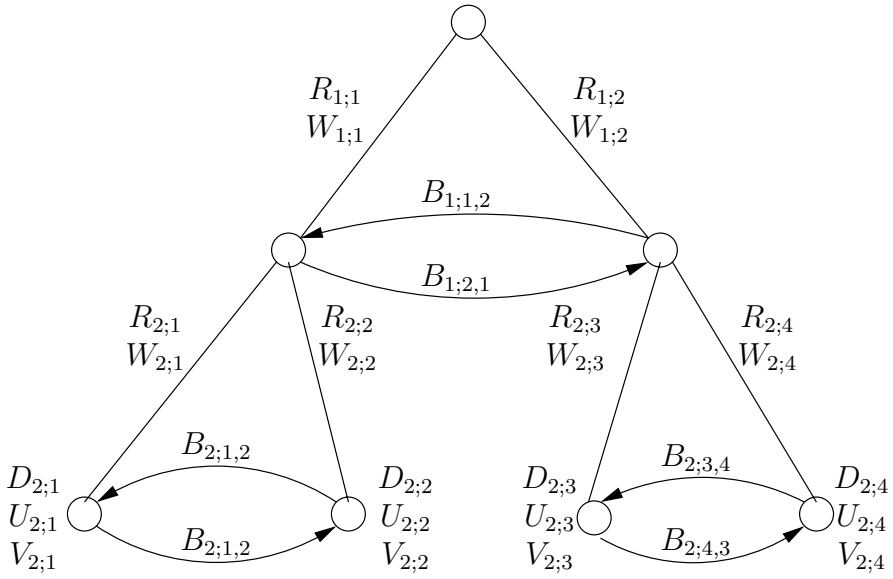


FIG. 2. *Three-level HSS representation on a binary tree.*

and the multiplication

$$U_{2;3}B_{2;3,4}V_{2;4}^H b_{2;4}.$$

Since

$$V_{1;2}^H b_{1;2} = \begin{pmatrix} V_{2;3}W_{2;3} \\ V_{2;4}W_{2;4} \end{pmatrix}^H \begin{pmatrix} b_{2;3} \\ b_{2;4} \end{pmatrix} = W_{2;3}^H V_{2;3}^H b_{2;3} + W_{2;4}^H V_{2;4}^H b_{2;4},$$

we can reduce the number of flops required to compute $V_{1;2}^H b_{1;2}$ in (3) if the number of columns in $W_{2;3}$ and $W_{2;4}$ is small compared to the number of rows in $V_{1;2}$. (This is the basis of all the super-fast algorithms and was first used by Greengard and Rokhlin in the design of FMMs.)

To formalize this idea we define the intermediate quantities

$$G_{k;i} = V_{k;i}^H b_{k;i}$$

and observe that the following recursions, as deduced from the preceding discussion, are available to compute them:

$$G_{K;i} = V_{K;i}^H b_{K;i}, \tag{4}$$

$$G_{k;i} = W_{k+1;2i-1}^H G_{k+1;2i-1} + W_{k+1;2i}^H G_{k+1;2i}. \tag{5}$$

With this notation we have that

$$U_{1;1}B_{1;1,2}V_{1;2}^H b_{1;2} = U_{1;1}B_{1;1,2}G_{1;2}.$$

We now observe that we need to perform the multiplication $U_{1;1}B_{1;1,2}G_{1;2}$. We also observe that we need to do the multiplication

$$U_{2;1}B_{2;1,2}G_{2;2}.$$

Using (1) we see that

$$U_{1;1}B_{1;1,2}G_{1;2} = \begin{pmatrix} U_{2;1}R_{2;1}B_{1;1,2}G_{1;2} \\ U_{2;2}R_{2;2}B_{1;1,2}G_{1;2} \end{pmatrix}.$$

Therefore the computation of $U_{1;1}B_{1;1,2}G_{1;2}$ can be merged with the computations of $U_{2;1}B_{2;1,2}G_{2;2}$ when computing

$$z_{2;1} = \cdots + U_{2;1}\left(B_{2;1,2}G_{2;2} + R_{2;1}B_{1;1,2}G_{1;2}\right) + \cdots,$$

where $\cdots$ denotes other terms that have to be added to produce the correct $z_{2;1}$. Similarly the term $U_{2;2}R_{2;2}B_{2;1,2}G_{2;2}$ from the computation of $A_{1;1,2}b_{1;2}$ can be merged into other terms involving $U_{2;2}$ in the computation of $z_{2;2}$. Clearly there is a recursive process occurring here. This motivates us to define the following intermediate quantities recursively:

$$F_{0;1} = 0, \tag{6}$$

$$F_{k,2i-1} = B_{k;2i-1,2i}G_{k;2i} + R_{k;2i-1}F_{k-1,i}, \tag{7}$$

$$F_{k,2i} = B_{k;2i,2i-1}G_{k;2i-1} + R_{k;2i}F_{k-1;i}. \tag{8}$$

We then observe that

$$z_{K;i} = D_{K;i}b_{K;i} + U_{K;i}F_{K;i}.$$

With that we have described how to compute $z = Ab$ rapidly when A has an HSS representation. Equations (4) and (5) are called the up-sweep recursions, and equations (6), (7), and (8) are called the down-sweep recursions for multiplication in the FMM literature.

4. Fast backward stable solver. In this section we present our fast solver. The algorithm we describe computes a ULV^H decomposition implicitly, where U and V are unitary matrices, and L is a lower-triangular matrix. By implicit, we mean that the factors are not computed and stored explicitly. However, the algorithm and techniques can be modified to compute the factors explicitly if so desired. That will be the subject of a future paper. The algorithm can also be easily modified to permit U and V to be represented as a product of elementary Gauss transforms and permutation matrices. This would lead to a more efficient algorithm but with some chance of numerical instability.

The basic idea of the algorithm is akin to that for the SSS representation. The one major difference is that we operate on all block rows at the same time, whereas in the SSS representation, each block row is operated on in a sequential fashion.

The algorithm is recursive in nature, and the recursion takes one of three possible forms.

4.1. Compressible off-diagonal blocks. This is the first possible way in which the recursion can proceed.

We begin by observing that block row i, excluding the diagonal block $D_{K;i}$, has its column space spanned by the columns of $U_{K;i}$. Hence if the number of columns of $U_{K;i}$, denoted by $n_{K;i}$, is strictly smaller than m_i, the number of rows in that block, we can find a unitary transformation $q_{K;i}$ such that

$$\bar{U}_{K;i} = q_{K;i}^H U_{K;i} = \begin{matrix} m_i - n_{K;i} \\ n_{K;i} \end{matrix} \begin{pmatrix} 0 \\ \hat{U}_{K;i} \end{pmatrix}.$$

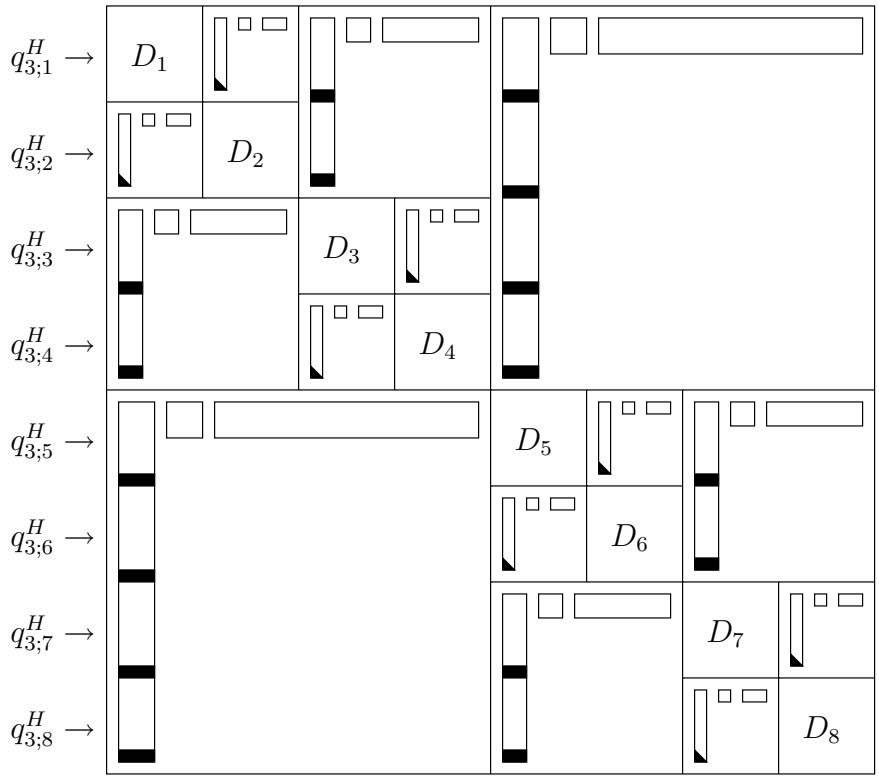


FIG. 3. *A pictorial representation showing the $q_{K;i}$'s compressing the off-diagonal portions of each block row. The black rectangles and triangles show the nonzero positions in the column bases of each off-diagonal block after compression by the $q_{K;i}$'s.*

In the above expression (and in the rest of the paper), variables written to the left of block matrices in parenthesis denote row partition sizes.

We now multiply block row i by q_i^H. (See Figure 3.) The change in the off-diagonal blocks is represented by the above equation since all of them have $U_{K;i}$ as the leading term. The ith block of the right-hand side changes to become

$$q^H_{K;i} b_{K;i} = \begin{matrix} m_i - n_{K;i} \\ n_{K;i} \end{matrix} \begin{pmatrix} \beta_{K;i} \\ \gamma_{K;i} \end{pmatrix}.$$

We also observe that $D_{K;i}$, the diagonal block, has become $q^H_{K;i} D_{K;i}$. Now we pick a unitary transformation $w_{K;i}$ such that

$$\bar{D}_{K;i} = (q^H_{K;i} D_{K;i}) w^H_{K;i} = \begin{matrix} \\ m_i - n_{K;i} \\ n_{K;i} \end{matrix} \begin{matrix} \begin{matrix} m_i - n_{K;i} & \quad n_{K;i} \end{matrix} \\ \begin{pmatrix} D_{K;i;1,1} & 0 \\ D_{K;i;2,1} & D_{K;i;2,2} \end{pmatrix} \end{matrix}.$$

We then multiply the block column i from the right by $w^H_{K;i}$. (See Figure 4.) The change in the diagonal block is represented by the above equation. The off-diagonal blocks in block column i have $V^H_{K;i}$ as the common last term. Hence we just need to multiply $V_{K;i}$ to obtain

$$\bar{V}_{K;i} = w_{K;i} V_{K;i} = \begin{matrix} m_i - n_{K;i} \\ n_{K;i} \end{matrix} \begin{pmatrix} \breve{V}_{K;i} \\ \hat{V}_{K;i} \end{pmatrix}.$$

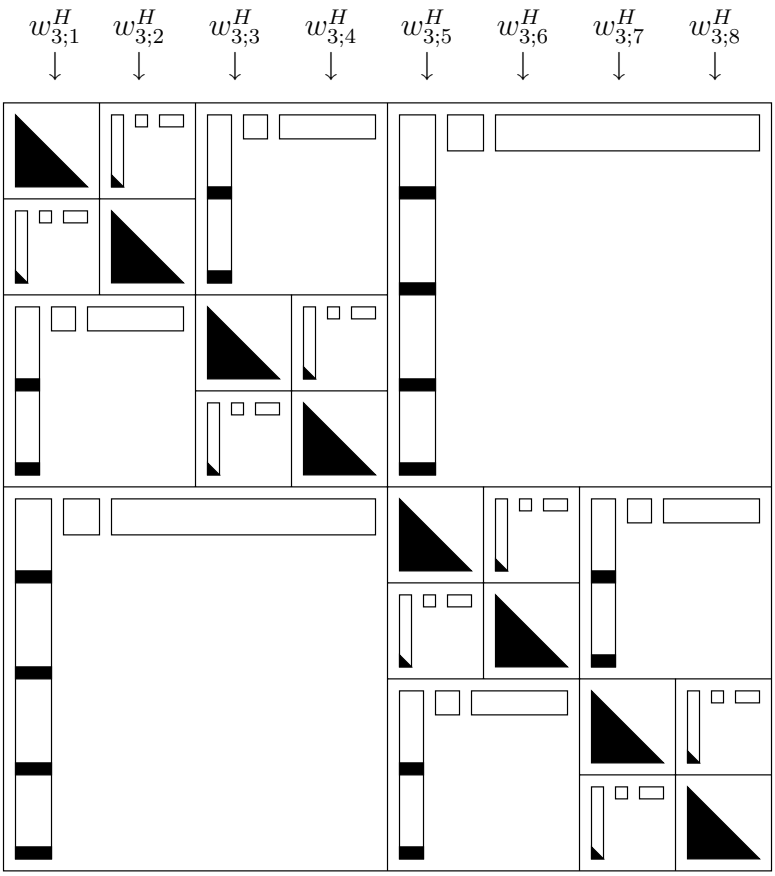


FIG. 4. *A pictorial representation showing the $w_{K;i}$'s lower triangularizing the diagonal blocks after the compression of the off-diagonal blocks by the $q_{K;i}$'s (see Figure 3). The black rectangles and triangles show the nonzero positions in the column bases and the diagonal blocks.*

Since we multiplied block column i from the right by $w^H_{K;i}$, we need to replace the unknowns $x_{K;i}$ by $w_{K;i}x_{K;i}$:

$$w_{K;i}x_{K;i} = \begin{matrix} m_i - n_{K;i} \\ n_{K;i} \end{matrix} \begin{pmatrix} z_{K;i} \\ \hat{x}_{K;i} \end{pmatrix}. \tag{9}$$

At this stage the first $m_i - n_{K;i}$ equations in block row i read as follows:

$$D_{K;i;1,1}z_{K;i} = \beta_{K;i},$$

which can be solved for $z_{K;i}$ to obtain $z_{K;i} = D^{-1}_{K;i;1,1}\beta_{K;i}$. We now need to multiply the first $m_i - n_{K;i}$ columns in the block column i by $z_{K;i}$ and subtract it from the right-hand side. To do this efficiently we observe that the system of equations has been transformed as follows:

$$\left(\operatorname{diag} q^H_{K;i}\, A\, \operatorname{diag} w^H_{K;i}\right)\left(\operatorname{diag} w_{K;i}\, x\right) = \operatorname{diag} q^H_{K;i}\, b.$$

If we define the vector

$$\bar{z}_{K;i} = \begin{matrix} m_i - n_{K;i} \\ n_{K;i} \end{matrix} \begin{pmatrix} z_{K;i} \\ 0 \end{pmatrix},$$

we then observe that the stated subtraction can be rewritten as follows:

$$\bar{b} = \operatorname{diag} q_{K;i}^H b - \left(\operatorname{diag} q_{K;i}^H A \operatorname{diag} w_{K;i}^H\right) \bar{z}.$$

We can do this operation rapidly by observing that

$$\left(\operatorname{diag} q_{K;i}^H A \operatorname{diag} w_{K;i}^H\right)$$

has the HSS representation $\{\bar{D}_{K;i}\}_{i=1}^{2^K}$, $\{\bar{U}_{K;i}\}_{i=1}^{2^K}$, $\{\bar{V}_{K;i}\}_{i=1}^{2^K}$, $\{\{R_{k;i}\}_{i=1}^{2^k}\}_{k=0}^{K}$, $\{\{W_{k;i}\}_{i=1}^{2^k}\}_{k=0}^{K}$, $\{\{B_{k;2i-1,2i}\}_{i=1}^{2^{k-1}}\}_{k=0}^{K}$, $\{\{B_{k;2i,2i-1}\}_{i=1}^{2^{k-1}}\}_{k=0}^{K}$ and by using the fast multiplication algorithm in section 3. Of course, the algorithm can (and should) be modified to take advantage of the zeros in $\bar{D}_{K;i}$, $\bar{U}_{K;i}$, and $\bar{z}_{K;i}$.

Once the subtraction has been done, we discard the first $m_i - n_{K;i}$ columns of block column i and the first $m_i - n_{K;i}$ rows of block row i. We observe that this leads to a new system of equations of the form

$$\hat{A}\hat{x} = \hat{b},$$

where

$$\bar{b}_{K;i} = \begin{matrix} m_i - n_{K;i} \\ n_{K;i} \end{matrix} \begin{pmatrix} * \\ \hat{b}_{K;i} \end{pmatrix},$$

and $\hat{A}$ has the HSS representation $\{D_{K;i;2,2}\}_{i=1}^{2^K}$, $\{\hat{U}_{K;i}\}_{i=1}^{2^K}$, $\{\hat{V}_{K;i}\}_{i=1}^{2^K}$, $\{\{R_{k;i}\}_{i=1}^{2^k}\}_{k=0}^{K}$, $\{\{W_{k;i}\}_{i=1}^{2^k}\}_{k=0}^{K}$, $\{\{B_{k;2i-1,2i}\}_{i=1}^{2^{k-1}}\}_{k=0}^{K}$, $\{\{B_{k;2i,2i-1}\}_{i=1}^{2^{k-1}}\}_{k=0}^{K}$.

Therefore we are left with a system of equations identical to the one we started with, and we can proceed to solve it recursively. Once we have done that we can recover the unknowns x from z and $\hat{x}$ using the formulas

$$x_{K;i} = w_{K;i}^H \begin{pmatrix} z_{K;i} \\ \hat{x}_{K;i} \end{pmatrix}.$$

Note that we have tacitly assumed that all block rows are such that $m_i > n_{K;i}$. However, it is easy to modify the equations so that only those block rows that satisfy $m_i > n_{K;i}$ have their off-diagonal blocks compressed.

4.2. Incompressible off-diagonal blocks. This is the second possibility for the recursion. It occurs when *all* block rows for the system cannot be compressed any further by invertible transformations from the left. In this case we proceed to merge block rows and columns that correspond to siblings in the merge tree. In particular consider the first two block rows:

$$\begin{pmatrix} D_{3;1;2,2} & U_{3;1}B_{3;1,2}V_{3;2}^H & U_{3;1}R_{3;1}B_{2;1,2}W_{3;3}^H V_{3;3}^H & U_{3;1}R_{3;1}B_{2;1,2}W_{3;4}^H V_{3;4}^H & \cdots \\ U_{3;2}B_{3;2,1}V_{3;1}^H & D_{3;2;2,2} & U_{3;2}R_{3;2}B_{2;1,2}W_{3;3}^H V_{3;3}^H & U_{3;2}R_{3;2}B_{2;1,2}W_{3;4}^H V_{3;4}^H & \cdots \end{pmatrix}.$$

We observe that these two block rows can be rewritten as

$$\begin{pmatrix} \begin{pmatrix} D_{3;1;2,2} & U_{3;1}B_{3;1,2}V_{3;2}^H \\ U_{3;2}B_{3;2,1}V_{3;1}^H & D_{3;2;2,2} \end{pmatrix} & \left(\begin{pmatrix} U_{3;1}R_{3;1} \\ U_{3;2}R_{3;2} \end{pmatrix} B_{2;1,2} \begin{pmatrix} V_{3;3}W_{3;3} \\ V_{3;4}W_{3;4} \end{pmatrix}^H \right) & \cdots \end{pmatrix}.$$

This immediately suggests that we merge as follows:

$$\hat{D}_{K-1;i} = \begin{pmatrix} D_{K;2i-1;2,2} & U_{K;2i-1}B_{K;2i-1,2i}V_{K;2i}^H \\ U_{K;2i}B_{K;2i,2i-1}V_{K;2i-1}^H & D_{K;2i;2,2} \end{pmatrix},$$

$$\hat{U}_{K-1;i} = \begin{pmatrix} U_{K;2i-1}R_{K;2i-1} \\ U_{K;2i}R_{K;2i} \end{pmatrix},$$

$$\hat{V}_{K-1;i} = \begin{pmatrix} V_{K;2i-1}W_{K;2i-1} \\ V_{K;2i}W_{K;2i} \end{pmatrix}.$$

We then see that A has an HSS representation (with a merge tree with K, as opposed to $K+1$, levels) given by the sequences $\{\hat{D}_{K-1;i}\}_{i=1}^{2^{K-1}}$, $\{\hat{U}\}_{i=1}^{2^{K-1}}$, $\{\hat{V}\}_{i=1}^{2^{K-1}}$, $\{\{R_{k;i}\}_{i=1}^{2^k}\}_{k=0}^{K-1}$, $\{\{W_{k;i}\}_{i=1}^{2^k}\}_{k=0}^{K-1}$, $\{\{B_{k;2i-1,2i}\}_{i=1}^{2^{k-1}}\}_{k=0}^{K-1}$, $\{\{B_{k;2i,2i-1}\}_{i=1}^{2^{k-1}}\}_{k=0}^{K-1}$. Let us denote by $\hat{A}$ the matrix with this HSS representation (of course, $A=\hat{A}$). We then observe that the system of equations is now in the form

$$\hat{A}x = b, \tag{10}$$

which is exactly in the form we started with, except that the new HSS representation has only K levels in the merge tree. Hence we can solve this system of equations recursively for x. That is, we check if there are compressible off-diagonal blocks. If so, we use the algorithm in section 4.1. If it is not compressible, we use the algorithm in this section. If the tree is just a leaf, we use the algorithm in section 4.3.

4.3. No off-diagonal blocks. Observe that if $K=0$, the equations read $D_1 x = b$, which can be solved by traditional means for x. This case terminates the recursion.

With this we have given a complete account of the algorithm.

4.4. Flop count. We use the flop counts in Table 1 of the basic matrix operations that can be found, for example, in [18].

TABLE 1
Flop counts of basic matrix operations.

Operation	Flops
QL factorization of skinny $m \times n$ matrix	$2n^2(m-n/3)$
Q times $m \times k$ matrix	$2kn(2m-n)$
Forward substitution of $n \times n$ matrix with k right-hand sides	n^2k
$m \times n$ times $n \times k$ matrix	$2mnk$

We begin by estimating the flop count for the fast multiplication algorithm, as that is an integral part of the solver. For simplicity we will assume that the ranks $n_{k;i}$ are independent of i, and that there are l indices in each of the leaves of the merge tree.

Computing $G_{K;i}$ will cost us $2ln_K r2^K$ flops, where r is the number of columns in the right-hand side. Computing $G_{k;i}$ from $G_{k+1;i}$ costs $4n_k n_{k+1} r 2^k$ flops. Computing $F_{k+1;i}$ from $F_{k;i}$ costs $42^{k+1} n_k n_{k+1} r$ flops. Finally computing $z_{K;i}$ costs $2lr(l+n_K)2^K$ flops. Summing these costs over k we obtain

$$2lr(l+n_K)2^K + 2ln_K r2^K + 8r\sum_{k=1}^{K-1} n_k n_{k+1} 2^k$$

as the total cost. Letting $N = 2^K l$ be the order of the matrix, we can simplify this to obtain

$$2Nr(l+2n_K) + 8r\sum_{k=1}^{K-1} n_k n_{k+1} 2^k$$

as the number of flops for the fast multiplication algorithm.

We now proceed to estimate the flops for the fast backward stable solver. To keep the calculations simple we will assume that each level of the tree undergoes a

compression step before going through a merge step. We will also assume that $m_{k;i}$, the size of the block rows at the kth stage, is independent of i.

Let us start with the compression step. We first need to compute the QL factorizations of $U_{k;i}$. This will cost us $2n_k^2(m_k - n_k/3)2^k$ flops. Then we need to apply $q_{k;i}$ to the right-hand side. This costs us $2rm_k(2m_k - n_k)2^k$ flops. We also need to apply $q_{k;i}$ to D_i (at the kth level), which costs us $2m_k n_k(2m_k - n_k)2^k$ flops. Next the LQ factorization of the diagonal blocks costs us $4m_k^3 2^k/3$ flops. Applying $w_{k;i}$ to $V_{k;i}$ costs $2n_k m_k^2 2^k$ flops. The partial forward-substitution at level k costs $(m_k - n_k)^2 r 2^k$ flops. Subtracting the computed unknowns from the right-hand side costs

$$2^k 2m_k r(m_k + 2n_k) + 8r\sum_{s=1}^{k-1} n_s n_{s+1} 2^s$$

flops. Recovering $x_{k;i}$ from $z_{k;i}$ will cost $2rm_k^2 2^k$ flops. That completes the compression stage.

For the merge step, forming the new diagonal blocks costs $8m_k n_k^2 2^k$ flops. Merging $U_{k;i}$ and $V_{k;i}$ costs $8m_k n_k^2 2^k$ flops.

Therefore the total cost of the fast backward stable solver is

$$\begin{aligned}\sum_{k=1}^{K} \Bigg(& 2n_k^2(m_k - n_k/3)2^k + 2rm_k(2m_k - n_k)2^k + 2m_k n_k(2m_k - n_k)2^k \\ & + 4m_k^3 2^k/3 + 2n_k m_k^2 2^k + (m_k - n_k)^2 r 2^k + 2^k 2m_k r(m_k + 2n_k) \\ & + 8r\sum_{s=1}^{k-1} n_s n_{s+1} 2^s + 16 m_k n_k^2 2^k \Bigg),\end{aligned}$$

which can be simplified to

$$\sum_{k=1}^{K} 2^k \left(\frac{4}{3} m_k^3 + 6m_k^2 n_k - \frac{2}{3} n_k^3 + r\left(7m_k^2 + n_k^2 + 8\sum_{s=1}^{k-1} n_s n_{s+1} 2^s \right) \right).$$

The terms not involving r can be thought of as the cost of factorization.

We now observe that under our assumptions $m_k = 2n_{k+1}$ for $k < K$. Making this substitution we can simplify the count to

$$\begin{aligned} & 2^K m_K^2 \left(\frac{4}{3} m_K + 6n_K \right) + 24 \sum_{k=1}^{K-1} 2^k n_{k+1}^2 n_k + \frac{14}{3} \sum_{k=2}^{K} 2^k n_k^3 - \frac{4}{3} n_1^3 \\ & + r\left(7m_K^2 2^K + 15\sum_{k=2}^{K} 2^k n_k^2 + n_1^2 + 8\sum_{k=1}^{K}\sum_{s=1}^{k-1} 2^s n_s n_{s+1} \right).\end{aligned}$$

To simplify further we assume that $n_k \geq n_{k+1}$. Then we can get an upper bound on the flop count

$$\begin{aligned} & 2^K m_K^2 \left(\frac{4}{3} m_K + 6n_K \right) + \frac{86}{3} \sum_{k=1}^{K} 2^k n_k^3 - \frac{4}{3} n_1^3 \\ & + r\left(7m_K^2 2^K + 15\sum_{k=2}^{K} 2^k n_k^2 + n_1^2 + 8\sum_{k=1}^{K}\sum_{s=1}^{k-1} 2^s n_s^2 \right).\end{aligned}$$

We now compute the flop counts for some canonical examples. First we consider the case when $n_k = p$, a constant. In this case the upper bound on the flop count simplifies to

$$2^K m_K^2 \left(\frac{4}{3} m_K + 6p\right) + \frac{86}{3} p^3 2^{K+1} + r\left(7 m_K^2 2^K + 23 p^2 2^{K+1}\right).$$

Using $N = 2^K m_K$, and assuming that $m_K = 2p$, we get

$$46 N p^2 + 37 N p r.$$

As can be seen, the constants are modest. By switching to Gauss transforms rather than Householder transforms we can reduce the constants even further.

In many cases this flop count is sufficient to give an indication of the performance of the algorithm. However, for theoretical purposes we also provide an upper bound on the flop count under the assumption that $n_k \leq \gamma^k n_0$. This model is useful when applying the algorithm to matrices of the form $A_{ij} = f(x_i, x_j)$, when the points x_i lie in high-dimensional spaces.

For example, when $f(x_i, x_j) = \log|x_i - x_j|$, and x_i is a point in the two-dimensional plane, we can take $n_0 = \alpha N^{\frac{1}{2}}$ and $\gamma = \frac{1}{\sqrt{2}}$. For there to be any speed-up possible at all we must have that

$$\alpha \leq \frac{N^{\frac{1}{2}}}{\sqrt{2}}.$$

For simplicity, and since it is common in practice, we assume that $\alpha \geq 1$. We then observe that $m_k = N 2^{-k} \geq 2\alpha N^{1/4} 2^{-k/2}$, provided $k < \log_2 N - 2(\log_2 \alpha + 1)$. Hence we take the depth of the tree to be

$$K = \lfloor \log_2 N - 2 \log_2 \alpha - 1 \rfloor.$$

Note that m_K is approximately $4\alpha^2$ in this scenario. Under these assumptions the flop count for the fast solver is not more than

$$98 N^{\frac{3}{2}} \alpha^3 + 70 N \alpha^4 + N \alpha^2 r (4 \log_2^2 N + 11 \log_2 N + 28).$$

As can be seen the constant is quite sensitive to the size of α.

Next we consider three-dimensional problems. For example, when $f(x_i, x_j) = \|x_i - x_j\|^{-1}$, and x_i is a point in three-dimensional space, we can take $n_0 = \alpha N^{\frac{2}{3}}$ and $\gamma = \frac{1}{\sqrt[3]{4}}$. To obtain any speed-up at all, we must ensure that $\alpha < (N/2)^{1/3}$. For the sake of simplicity we will also assume that $\alpha \geq 1$. We can determine the maximum depth of the tree from the constraint $m_k \geq 2n_k$, which yields

$$K \leq \lfloor \log_2 N - 3 \log_2 \alpha - 1 \rfloor.$$

Under this scenario m_K is approximately $2\alpha^3$. With these assumptions the flop count for the fast solver is less than

$$58 N^2 \alpha^3 + 18 N \alpha^6 + r(39 N^{\frac{4}{3}} \log_2 N \alpha^2 + 74 N^{\frac{4}{3}} \alpha^2 + 14 N \alpha^3).$$

As can be seen the constant is modest.

Observe that in both cases the fast dense solver matches the asymptotic complexity of the corresponding sparse direct finite-element and finite-difference solvers

TABLE 2

CPU run-times in seconds for both the fast stable algorithm and standard solver for random HSS matrices with $m_i = n_{k;i} = p_{k;i}$ for all k and i. Timings are not reported when there was insufficient main memory. (GEPP = Gaussian elimination with partial pivoting.)

	Size							
$m_i/n_{k;i}/p_{k;i}$	256	512	1024	2048	4096	8192	16,384	32,768
16	0.03	0.06	0.13	0.27	0.49	0.99	2.10	4.74
32	0.05	0.12	0.27	0.56	1.14	2.34	4.75	9.81
64	0.09	0.26	0.58	1.27	2.60	5.33	11.11	23.19
128	0.09	0.63	1.71	3.92	8.45	17.14	35.16	74.07
GEPP	0.07	0.33	2.12	14.81	113.75	891.59	...	...

	Size				
$m_i/n_{k;i}/p_{k;i}$	65,536	131,072	262,144	524,288	1,048,576
16	9.93	27.95	64.28	224.73	889.65
32	20.56	44.53	106.35	408.39	...
64	50.24	129.81	405.13	...	...
128	158.97	380.89	...	...	...

of the same dimension. Of course, many times the integral equations corresponding to a particular PDE will be one dimension smaller, frequently yielding the advantage to the integral equation method. However, the quadratic dependence on N for three-dimensional problems makes this algorithm suitable only when the linear system is highly ill-conditioned and a suitable preconditioner is lacking. In fact, this solver can serve as an ideal preconditioner in this and other situations. Another situation where this method is suitable even for three-dimensional problems is when there is a large number of right-hand sides.

We remark that if many of the leaves at level K are empty, then the algorithm we have specified will become inefficient. A more complicated algorithm that does not suffer from this deficiency will be presented in a future paper.

4.5. Experimental run-times. We now present CPU run-times for our fast solver. These timings were obtained on an Apple dual 1GHz PowerPC G4 machine with 1.5GB of RAM, though no explicit use was made of the dual processors. Vendor supplied BLAS [14] (uniprocessor) and LAPACK 3.0 were used in all routines. We report on problem sizes ranging from 256 unknowns to 1,048,576 unknowns. Off-diagonal ranks $n_{k;i}$ and $p_{k;i}$ were chosen to range from 16 to 128. In every instance we chose $m_i = n_{k;i} = p_{k;i}$ for all i. The matrices were generated randomly to these specifications.

The CPU run-times in seconds are reported in Table 2. Also shown are CPU run-times in seconds for the standard Gaussian elimination with row pivoting solver from LAPACK. This routine is highly tuned and essentially runs at peak flop-rates. As can be seen our fast solver breaks even with the standard solver for reasonably small matrix sizes, as predicted by the flop count. Entries marked by ellipses indicate instances where there was insufficient memory to run the test. Again this also indicates another reason why the fast solver might be preferred: memory efficiency.

4.6. Stability. The fast solver we presented can be shown to be numerically backward stable, provided the HSS representation is in the *proper form*. However, the proof would detract from the main ideas of this paper and will be presented elsewhere. By proper form we mean that $\|R_{k;i}\| \leq 1$ and $\|W_{k;i}\| \leq 1$ for a submultiplicative norm. We observe that the HSS construction algorithm presented in section 5 satisfies this requirement for the 2-norm.

However, the algorithm can also be shown to be backward stable to first order in machine precision even if the weaker condition $\|R_{k;i}R_{k+1;2i(-1)}\cdots\| \leq p(n)$ and $\|W_{k;i}W_{k+1;2i(-1)}\cdots\| \leq q(n)$ is satisfied, where $p(n)$ and $q(n)$ are low-degree polynomials in n. This condition is satisfied by the fast HSS construction algorithm presented in subsection 5.1.

The reason for the claimed stability of the fast solver is due to the use of unitary transformations and a single forward substitution. The proof is similar to the one for the sequentially semiseparable representation [3] and will be presented elsewhere.

In Table 3 we present computed experimental backward errors for the fast solver on a wide class of problems which lends credence to our claims of stability. These experiments were carried out in double precision for matrix sizes ranging from 256 to 4096. The ranks of the off-diagonal blocks $n_{k;i}$ and $p_{k;i}$ were chosen to range from 16 to 128. Although the HSS forms were generated randomly, we did not ensure proper form. We only ensured the milder condition that the entries of $W_{k;i}$ and $R_{k;i}$ were no larger than 1 in magnitude. As can be seen from the backward errors presented in Table 3 the fast solver was backward stable even in this case.

TABLE 3

One-norm backward errors $\|Ax-b\|_1/(\epsilon_{mach}(\|A\|_1\|x\|_1+\|b\|_1))$ of the fast solver in double precision with $|W_{k;i}| \leq 1$ and $|R_{k;i}| \leq 1$. Entries much larger than 1 indicate a potential lack of backward stability.

	Size				
$m_i/n_{k;i}/p_{k;i}$	256	512	1024	2048	4096
16	0.31	0.27	0.32	0.25	0.16
32	0.34	0.33	0.24	0.22	0.20
64	0.54	0.38	0.33	0.28	0.25
128	0.47	0.43	0.36	0.28	0.28

5. Computing the HSS representation. In this section we describe an $O(n^2)$ algorithm to compute the HSS representation of an arbitrary matrix to a given tolerance.

The key idea is to compute the singular value decomposition (SVD) of the matrices

$$H_{k;i} = \begin{pmatrix} A_{k;i,1} & A_{k;i,2} & \cdots & A_{k;i,i-1} & A_{k;i,i+1} & A_{k;i,i+2} & \cdots & A_{k;i,2^k} \end{pmatrix}. \tag{11}$$

Notice that $H_{k;i}$ is essentially block row i of the matrix when partitioned according to level k of the merge tree, except that the diagonal block corresponding to that level $A_{k;i,i}$ is missing.

Similarly we also need to compute the SVD of the matrices

$$G_{k;i} = \begin{pmatrix} A^H_{k;1,i} & A^H_{k;2,i} & \cdots & A^H_{k;i-1,i} & A^H_{k;i+1,i} & A^H_{k;i+2,i} & \cdots & A^H_{k;2^k;i} \end{pmatrix}^H. \tag{12}$$

Suppose we have the SVD of $H_{k;i}$ and $G_{k;i}$ for $k = 1$ to K and for $i = 1$ to 2^k:

$$H_{k;i} = U_{k;i}C_{k;i}J^H_{k;i},$$
$$G_{k;i} = L_{k;i}M_{k;i}V^H_{k;i}.$$

Observe that these equations directly define the auxiliary quantities $U_{k;i}$ and $V_{k;i}$ that appear in (1) and (2). In particular we obtain $U_{K;i}$ and $V_{K;i}$. Using (1) and (2) we

can also compute

$$\begin{aligned} R_{k+1;2i-1} &= U^H_{k+1;2i-1}(U_{k;i})_1, \\ R_{k+1;2i} &= U^H_{k+1;2i}(U_{k;i})_2, \\ W_{k+1;2i-1} &= V^H_{k+1;2i-1}(V_{k;i})_1, \\ W_{k+1;2i} &= V^H_{k+1;2i}(V_{k;i})_2, \end{aligned}$$

where we have the conforming partitions

$$\begin{aligned} U_{k;i} &= \begin{pmatrix} (U_{k;i})_1 \\ (U_{k;i})_2 \end{pmatrix} = \begin{pmatrix} U_{k+1;2i-1}R_{k+1;2i-1} \\ U_{k+1;2i}R_{k+1;2i} \end{pmatrix}, \\ V_{k;i} &= \begin{pmatrix} (V_{k;i})_1 \\ (V_{k;i})_2 \end{pmatrix} = \begin{pmatrix} V_{k+1;2i-1}W_{k+1;2i-1} \\ V_{k+1;2i}W_{k+1;2i} \end{pmatrix}. \end{aligned}$$

This leaves us only with determining formulas for $B_{k;2i-1,2i}$ and $B_{k;2i,2i-1}$. Observe that $A_{k;2i-1,2i}$ is the $2i-1$ submatrix of $H_{k;2i-1}$ and $G_{k;2i}$ in the partitioning in (11) and (12). Therefore assuming that $(J_{k;2i-1})_{2i-1}$ and $(L_{k;2i})_{2i-1}$ denote the appropriate submatrices, we have that

$$A_{k;2i-1,2i} = U_{k;2i-1}B_{k;2i-1,2i}V^H_{k;2i} = U_{k;2i-1}C_{k;2i-1}(J_{k;2i-1})^H_{2i-1} = (L_{k;2i})_{2i-1}M_{k;2i}V^H_{k;2i}.$$

This immediately gives us the formulas

$$B_{k;2i-1,2i} = C_{k;2i-1}(J_{k;2i-1})^H_{2i-1}V_{k;2i} = U^H_{k;2i-1}(L_{k;2i})_{2i-1}M_{k;2i}.$$

Similarly $A_{k;2i,2i-1}$ is the $2i-1$ submatrix of $H_{k;2i}$ and $G_{k;2i-1}$ in the partitioning in (11) and (12). Therefore assuming that $(J_{k;2i})_{2i-1}$ and $(L_{k;2i-1})_{2i-1}$ denote the appropriate submatrices, we have that

$$A_{k;2i,2i-1} = U_{k;2i}B_{k;2i,2i-1}V^H_{k;2i-1} = U_{k;2i}C_{k;2i}(J_{k;2i})^H_{2i-1} = (L_{k;2i-1})_{2i-1}M_{k;2i-1}V^H_{k;2i-1}.$$

This immediately gives us the formulas

$$B_{k;2i,2i-1} = C_{k;2i}(J_{k;2i})^H_{2i-1}V_{k;2i-1} = U^H_{k;2i}(L_{k;2i-1})_{2i-1}M_{k;2i-1}.$$

All we need now is an efficient way to compute the needed SVDs. To this end we observe that $H_{k;i}$ is closely related to $H_{k+1;2i-1}$ and $H_{k+1;2i}$. In fact, by dropping the $2i-1$ block column from

$$\begin{pmatrix} H_{k+i;2i-1} \\ H_{k+1;2i} \end{pmatrix},$$

we obtain $H_{k;i}$. Similarly, by dropping the $2i-1$ block row from

$$\begin{pmatrix} G_{k+1;2i-1} & G_{k+1;2i} \end{pmatrix},$$

we obtain $G_{k;i}$. Hence we can obtain the SVD of $H_{k;i}$ efficiently from the SVDs of $H_{k+1;2i-1}$ and $H_{k+1;2i}$. Similarly for $G_{k;i}$.

Assuming that $B_{k;2i-1,2i}$ is an $n_{k;i} \times n_{k;i}$ matrix and $B_{k;2i,2i-1}$ is a $p_{k;i} \times p_{k;i}$ matrix, the complexity of the above algorithm is $O(N(N + \sum_{k;i}(n^2_{k;i} + p^2_{k;i})))$.

The cost of the algorithm can be reduced by replacing the SVD with a rank-revealing QR factorization [11, 21] instead.

5.1. Smooth matrices. When the matrix entry A_{ij} is specified by a function $f(x_i, x_j)$ that is smooth away from the diagonal, the HSS representation can be computed more rapidly than in the general case. In this section we consider the special case when the points x_i lie on the real line. The more general case is beyond the scope of this paper. Important examples of the function $f(x, y)$ include $\log \|g(x) - g(y)\|$ and $\|g(x) - g(y)\|^\alpha$, where $g : \mathcal{R} \to \mathcal{R}^d$ represents a simple closed or nonclosed curve in d-dimensional space. For applications see [30, 31].

Since we are restricting ourselves to uniform HSS representations in this paper, we will assume that the points x_i are distributed uniformly in the interval $[0, 1]$. Note that this does not mean that the points x_i are equispaced. Furthermore, for simplicity, we will assume the function f has at most singularities at 0 and 1, and that it is analytic away from these singularities. A good example to keep in mind is $f(x, y) = \log |x - y|$.

From the basic theory of polynomial approximation of such functions it follows that if $T_k(x)$ denotes the kth Chebyshev polynomial

$$T_k(x) = \cos(k \arccos x), \qquad -1 \le x \le 1,$$

and if

$$\phi_{a,b} : [a, b] \to [-1, 1], \qquad \phi_{a,b}(x) = -1 + 2\frac{x - a}{b - a}$$

denotes the affine-linear function that maps the interval $[a, b]$ to $[-1, 1]$, then on any rectangle $[a, b] \times [c, d]$ such that $a < b < c < d$ and $\min(d - c, b - a) > c - b$, we can find a short two-sided Chebyshev expansion of $f(x, y)$ to a given accuracy:

$$f(x, y) \approx \sum_{p,q} \beta_{p,q} T_p(\phi_{a,b}(x)) T_q(\phi_{c,d}(y)).$$

More specifically, the (i, j)th entry of the matrix can be represented to a prescribed accuracy by a short expansion of the form

$$f(x_i, x_j) \approx \sum_{p,q} \beta_{p,q} T_p(\phi_{a,b}(x_i)) T_q(\phi_{c,d}(y_j)).$$

We shall now show how these expansion coefficients can be used to compute an HSS representation for the matrix quickly.

We first need to specify the merge tree we are going to use. We do so as follows. We will assume that all the points x_i lie in the interval $[0, 1]$. Hence we will associate the interval $[0, 1]$ (and hence all the points x_i, and hence all indices) with the root node. With the left child of the root we associate the interval $[0, 0.5)$ and with the right child the interval $[0.5, 1]$. This means that we associate all points x_i in the interval $[0, 0.5)$ with the left child and hence all the corresponding indices with the left child. Similarly for the right child. To the left child of the left child of the root node, namely, Node$(2, 1)$, we associate the interval $[0, 0.25)$, to Node$(2, 2)$ we associate the interval $[0.25, 0.5)$, and so on. In this way we assign the indices to the merge tree.

Note that the number of indices in two different nodes at the same level can be different. Also note that we do not assume that the points x_i are equispaced.

Let us denote the set of points x_i that belong to Node(k, i) by $x_{k;i}$. Let us denote by $\Gamma_{k;i}$ the Chebyshev–Vandermonde matrix evaluated at the points $x_{k;i}$. We will assume that the number of columns in $\Gamma_{k;i}$ is fixed at p to ease the exposition.

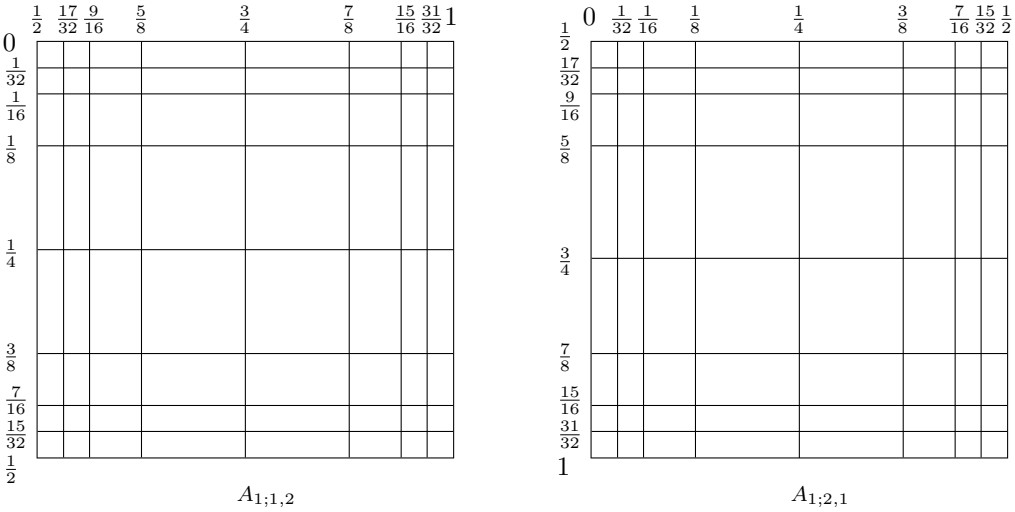


FIG. 5. *Block partitioning of $A_{1;1,2}$ and $A_{1;2,1}$ suitable for Chebyshev expansions. The vertical and horizontal lines are labeled according to the interval boundaries.*

Each node of the merge tree is associated with a particular interval of the real line. In particular Node(k,i) is associated with the interval $2^{-k}[i-1,i]$. Therefore it follows that $A_{k;i,j}$ is associated with the *rectangle* $2^{-k}[i-1,i] \times 2^{-k}[j-1,j]$.

Figure 5 displays a partitioning of $A_{1;1,2}$ and $A_{1;2,1}$ that will prove useful. We observe that each off-diagonal block, except possibly the bottom-left and upper-right blocks, in the displayed partition is associated with a rectangle on which the function f has a short two-sided Chebyshev expansion. However, note that the blocks are sometimes specified by intervals on two different levels of the merge tree. Hence we will use the notation $A_{(k;i),(r;j)}$ to denote the submatrix whose row indices come from Node(k,i) and column indices come from Node(r,j). We shall also use the notation

$$A_{(k;i),(r;j)} = \Gamma_{k;i} C_{(k;i),(r;j)} \Gamma_{r;j}^H \tag{13}$$

for the corresponding two-sided Chebyshev expansion. Observe that $C_{(k;i),(r;j)}$ can be computed in time independent of the size of $A_{(k;i),(r;j)}$. Since the block $A_{K;i,i+1}$ does not necessarily have a short two-sided Chebyshev expansion, we will assume instead that it has fewer than p rows and columns, in which case it trivially has an expansion of the form (13).

To construct the HSS representation we remind the reader that it is the low-rank expansions of $H_{k;i}$ and $G_{k;i}$ that are crucial. Hence in Figure 6 we show the partitioning of $H_{2;3}$ that we will use. Now observe that we can construct a low-rank expansion for $A_{k;i,i+1}$ and $A_{k;i+1,i}$ for odd i, as follows. Let

$$\Delta_{k;i} = \operatorname{diag}\begin{pmatrix} \Gamma_{K;2^{K-k}(i-1)+1} \\ \Gamma_{K;2^{K-k}(i-1)+2} \\ \vdots \\ \Gamma_{k+2;2(2i-1)} \\ \Gamma_{k+2;2(2i)-1} \\ \vdots \\ \Gamma_{K;2^{K-k}i-1} \\ \Gamma_{K;2^{K-k}i} \end{pmatrix} \tag{14}$$

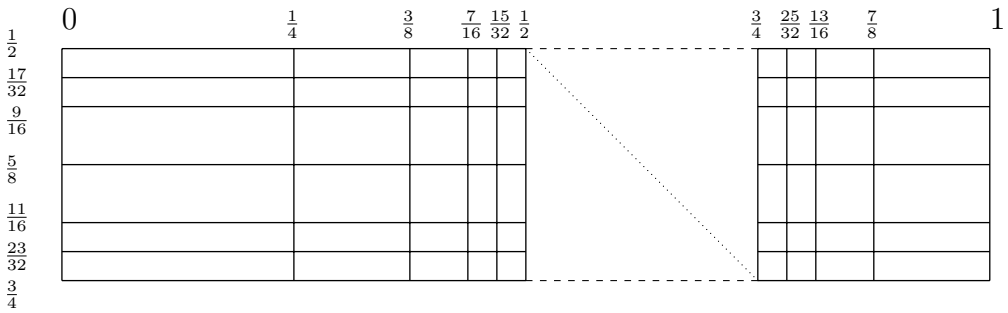


FIG. 6. *Block partitioning of $H_{2;3}$ suitable for Chebyshev expansions. The vertical and horizontal lines are labeled according to the associated interval boundaries. The missing block in the middle is the diagonal block. The dotted diagonal line shows the position of the diagonal.*

be a block-diagonal matrix. In the above notation we assume that $\Delta_{K;i} = \Gamma_{K;i}$ and

$$\Delta_{K-1;i} = \begin{pmatrix} \Gamma_{K;2i-1} \\ \Gamma_{K;2i} \end{pmatrix}.$$

Note that these formulas are consistent with (14). Then let

$$\begin{aligned} U_{k;i} &= \Delta_{k;i}, \\ V_{k;i} &= \Delta_{k;i}, \end{aligned}$$

and let $B_{k;i,i+1}$ be the block matrix with block entries

$$(B_{k;i,i+1})_{r,s} = C_{(k_r;i_r),(k_s;(i+1)_s)},$$

where $\mathrm{Node}(k_r, i_r)$ is the node corresponding to the rth diagonal block of $\Delta_{k;i}$. Similarly we define

$$(B_{k;i+1,i})_{r,s} = C_{(k_r;(i+1)_r),(k_s;i_s)}.$$

All we have to specify now is $R_{k;i}$ and $W_{k;i}$. First observe that $R_{k;i} = W_{k;i}$ since $U_{k;i} = V_{k;i}$. From the definition of $\Delta_{k;i}$ observe that

$$\Delta_{k;i} = \begin{pmatrix} \Delta_{k+1;2i-1}\Omega_{k+1;2i-1} & 0 \\ 0 & \Delta_{k+1;2i}\Omega_{k+1;2i} \end{pmatrix},$$

where

$$\begin{aligned} R_{k;2i-1} &= \begin{pmatrix} \Omega_{k;2i-1} & 0 \end{pmatrix}, \\ R_{k;2i} &= \begin{pmatrix} 0 & \Omega_{k;2i} \end{pmatrix}. \end{aligned}$$

Hence it is sufficient to specify the $\Omega_{k;i}$'s. To do that we first specify the two sets of auxiliary matrix-valued functions

$$\begin{aligned} \sigma_u(0) &= I, \\ \sigma_u(i+1) &= \begin{pmatrix} \sigma_u(i)C_L \\ C_R \end{pmatrix}, \end{aligned}$$

and

$$\sigma_l(0) = I,$$
$$\sigma_l(i+1) = \begin{pmatrix} C_L \\ \sigma_l(i) C_R \end{pmatrix}.$$

Then

$$\Omega_{k;2i} = \begin{pmatrix} \sigma_u(K-k-1) & 0 \\ 0 & I \end{pmatrix},$$
$$\Omega_{k;2i-1} = \begin{pmatrix} I & 0 \\ 0 & \sigma_l(K-k-1) \end{pmatrix},$$

with the understanding that $\sigma_u(-1)$ and $\sigma_l(-1)$ denote the empty matrices.

With this we have given a complete specification for computing the HSS representation (assuming a uniform tree) of a smooth matrix with a one-dimensional kernel function.

However, given the sparse structure of $U_{k;i}$ and $R_{k;i}$, the fast solvers and multipliers presented in this paper can, and should, be modified to exploit the extra structure. This is important, as the Chebyshev expansions are not optimal low-rank expansions.

5.2. Sparse matrices. In the previous subsection we showed how to construct rapidly the HSS representation of matrices whose entries are given by kernel functions that are smooth away from the diagonal. Such matrices are intimately associated with the fast multipole method and integral equations. In this subsection we consider sparse matrices. For sparse matrices we can quickly construct a possibly suboptimal HSS representation. For many sparse matrices this construction will actually lead to the optimal HSS representation.

We proceed as follows. First we must determine the row and column partition sizes. In this paper we will assume that these two partitions are identical. Suppose m_i denotes the size of the ith partition. We will again assume that the HSS tree is going to be uniform and that the number of partitions is 2^K for some K. The matrices D_i are straightforward to compute.

We form U_i as follows. Suppose the j_ith row in the ith partition is the first row in that partition to have a nonzero entry that is not in D_i; then the first column of U_i will be the zero column with a one in the j_ith position. Suppose g_i is the next row after the j_ith one in the ith partition that has a nonzero entry outside D_i; then the second column of U_i will be a zero column with a one in the g_ith position. We proceed until we have exhausted all the rows in the ith partition. Notice that we have constructed U_i such that it is guaranteed to be the column basis for $H_{K;i}$, as it must.

Next we form V_i. The construction is similar to that for U_i, except that we must deal with the columns of the ith partition, and in particular the nonzero entries in that partition that do not lie in D_i. Suppose the j_ith column in the ith partition is the first column in that partition to have a nonzero entry that is not in D_i; then the first column of V_i will be the zero column with a one in the j_ith position. Suppose g_i is the next column after the j_ith one in the ith partition that has a nonzero entry outside D_i; then the second column of V_i will be a zero column with a one in the g_ith position. We proceed until we have exhausted all the columns in the ith partition. Notice that we have constructed V_i such that it is guaranteed to be column basis for $G_{K;i}$, as it must.

Now we specify how to form $R_{k;i}$. First we observe that we could compute $U_{k;i}$ using the same ideas we used to compute $U_i = U_{K;i}$. From that we could then recover $R_{k;i}$. However, we can also do this in a direct fashion. As usual, let

$$U_{k;i} = \begin{pmatrix} (U_{k;i})_1 \\ (U_{k;i})_2 \end{pmatrix} = \begin{pmatrix} U_{k+1;2i-1} R_{k+1;2i-1} \\ U_{k+1;2i} R_{k+1;2i} \end{pmatrix}.$$

Then we observe that $R_{k+1;2i}$, for example, must drop the right columns in $U_{k+1;2i}$ so as to produce $(U_{k;i})_2$. Hence by looking at the nonzero entries of $A_{k+1;2i,2i-1}$ and $A_{k+1;2i;2i+1}$, we can determine potential columns of $U_{k+1;2i}$ that must be dropped. We pick $R_{k+1;2i}$ so that it drops just those columns. Note that not every nonzero row in $A_{k+1;2i,2i-1}$ and $A_{k+1;2i;2i+1}$ induces a drop in $U_{k+1;2i}$, since some other column in the same row might still have a nonzero entry.

We compute $W_{k;i}$ in a fashion similar to that for $R_{k;i}$ but with respect to $V_{k;i}$ rather than $U_{k;i}$.

All that is left to be specified is $B_{k;i,j}$. But this is easy now. $B_{k;i,j}$ is just the matrix obtained by dropping all zero rows and columns of $A_{k;i,j}$.

As can be seen, the HSS representation of a sparse matrix can be computed in time proportional to the number of nonzeros in the matrix, provided, of course, that the sparse matrix data structure supports efficient access for sequential reads of the nonzeros entries of any row or column. Many common sparse matrix data structures do exactly this, so we do not comment on it any further.

REFERENCES

[1] F. X. Canning and K. Rogovin, *Fast direct solution of moment-method matrices*, IEEE Antennas and Propagation Magazine, 40 (1998), pp. 15–26.

[2] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 669–686.

[3] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A. van der Veen, *Fast stable solver for sequentially semi-separable linear systems of equations*, in High Performance Computing—HiPC 2002: 9th International Conference, Lecture Notes in Comput. Sci. 2552, S. Sahni et al., eds., Springer-Verlag, Heidelberg, 2002, pp. 545–554.

[4] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, and A. van der Veen, *Fast Stable Solvers for Sequentially Semi-separable Linear Systems of Equations*, Technical report, Mathematic Department, University of California, Berkeley, 2003.

[5] S. Chandrasekaran and M. Gu, *Fast and stable algorithms for banded plus semiseparable systems of linear equations*, SIAM J. Matrix Anal. Appl., 25 (2003), pp. 373–384.

[6] S. Chandrasekaran and M. Gu, *A fast and stable solver for recursively semi-separable systems of linear equations*, in Structured Matrices in Mathematics, Computer Science, and Engineering, II, Contemp. Math. 281, V. Olshevsky, ed., AMS, Providence, RI, 2001, pp. 39–53.

[7] S. Chandrasekaran and M. Gu, *Fast and stable eigendecomposition of symmetric banded plus semi-separable matrices*, Linear Algebra Appl., 313 (2000), pp. 107–114.

[8] S. Chandrasekaran and M. Gu, *A divide-and-conquer algorithm for the eigendecomposition of symmetric block-diagonal plus semiseparable matrices*, Numer. Math., 96 (2004), pp. 723–731.

[9] S. Chandrasekaran, M. Gu, and T. Pals, *A Fast and Stable Solver for Smooth Recursively Semi-separable Systems*, Paper presented at the SIAM Annual Conference, San Diego, CA, 2001, and SIAM Conference of Linear Algebra in Controls, Signals and Systems, Boston, MA, 2001.

[10] S. Chandrasekaran, M. Gu, and T. Pals, *Fast and Stable Algorithms for Hierarchically Semi-separable Representations*, Technical report, Department of Mathematics, University of California, Berkeley, 2004.

[11] S. Chandrasekaran and I. C. F. Ipsen, *On rank-revealing factorisations*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 592–622.

[12] Y. Chen, *Fast direct solver for the Lippmann–Schwinger equation*, Adv. Comput. Math., 16 (2002), pp. 175–190; also available online at http://www.math.nyu.edu/faculty/yuchen/onr/intro.htm.

[13] P. Dewilde and A. van der Veen, *Time-Varying Systems and Computations*, Kluwer Academic, Boston, MA, 1998.

[14] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, *Algorithm* 679: *A set of level* 3 *basic linear algebra subprograms: Model implementation and test programs*, ACM Trans. Math. Softw., 16 (1990), pp. 18–28.

[15] Y. Eidelman and I. Gohberg, *On a new class of structured matrices*, Integral Equations Operator Theory, 34 (1999), pp. 293–324.

[16] Y. Eidelman and I. Gohberg, *A modification of the Dewilde van der Veen method for inversion of finite structured matrices,* Linear Algebra Appl., 343/344 (2001), pp. 419–450.

[17] I. Gohberg, T. Kailath, and I. Koltracht, *Linear complexity algorithms for semiseparable matrices*, Integral Equations Operator Theory, 8 (1985), pp. 780–804.

[18] G. Golub and C. van Loan, *Matrix Computations*, 3rd ed., Johns Hopkins University Press, Baltimore, MD, 1996.

[19] D. Gope and V. Jandhyala, *An iteration-free fast multilevel solver for dense method of moment systems,* in Proceedings of the IEEE 10th Topical Meeting on Electrical Performance of Electronic Packaging, IEEE Press, Piscataway, NJ, 2001, pp. 177–180.

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

[21] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing QR factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.

[22] L. Gurel and W. C. Chew, *Fast direct (noniterative) solvers for integral-equation formulations of scattering problems,* in Antennas: Gateways to the Global Network, Vol. 1, IEEE Antennas and Propagation Society International Symposium, Vol. 1, IEEE Press, New York, 1998, pp. 298–301.

[23] W. Hackbusch, *A sparse arithmetic based on $\mathcal{H}$-matrices. Part* I: *Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[24] W. Hackbusch, B. N. Khoromskij, and S. Sauter, *On H^2-Matrices*, preprint 50, MPI, Leipzig, 1999.

[25] W. Hackbusch and S. Borm, *Data-sparse approximation by adaptive H^2-matrices*, Computing, 69 (2002), pp. 1–35.

[26] P. Jones, J. Ma, and V. Rokhlin, *A fast direct algorithm for the solution of the Laplace equation on regions with fractal boundaries,* J. Comput. Phys., 113 (1994), pp. 35–51.

[27] N. Mastronardi, S. Chandrasekaran, and S. van Huffel, *Fast and stable two-way chasing algorithm for diagonal plus semi-separable systems of linear equations*, Numer. Linear Algebra Appl., 38 (2000), pp. 7–12.

[28] N. Mastronardi, S. Chandrasekaran, and S. van Huffel, *Fast and stable algorithms for reducing diagonal plus semi-separable matrices to tridiagonal and bidiagonal form*, BIT, 41 (2001), pp. 149–157.

[29] V. Rokhlin, *Applications of volume integrals to the solution of PDEs*, J. Comput. Phys., 86 (1990), pp. 414–439.

[30] P. Starr, *On the Numerical Solution of One-Dimensional Integral and Differential Equations*, Thesis advisor: V. Rokhlin, Research Report YALEU/DCS/RR-888, Department of Computer Science, Yale University, New Haven, CT, 1991.

[31] P. Starr and V. Rokhlin, *On the numerical solution of* 2*-point boundary value problem.* 2, Comm. Pure Appl. Math., 47 (1994), pp. 1117–1159.