SIAM J. MATRIX ANAL. APPL.
Vol. 29, No. 1, pp. 67–81

A FAST SOLVER FOR HSS REPRESENTATIONS VIA SPARSE MATRICES*

S. CHANDRASEKARAN†, P. DEWILDE‡, M. GU§, W. LYONS¶, AND T. PALS†

Abstract. In this paper we present a fast direct solver for certain classes of dense structured linear systems that works by first converting the given dense system to a larger system of block sparse equations and then uses standard sparse direct solvers. The kind of matrix structures that we consider are induced by numerical low rank in the off-diagonal blocks of the matrix and are related to the structures exploited by the fast multipole method (FMM) of Greengard and Rokhlin. The special structure that we exploit in this paper is captured by what we term the hierarchically semiseparable (HSS) representation of a matrix. Numerical experiments indicate that the method is probably backward stable.

Key words. fast multipole method, low-rank structures, fast solvers, orthogonal factorizations, hierarchically semiseparable representations, sparse matrices, direct sparse solvers

AMS subject classifications. 65F05, 65F50

DOI. 10.1137/050639028

1. Introduction.

Beginning with the early work of Gohberg, Kailath, and Koltracht [6] and Rokhlin [11], and the introduction of the fast multipole method (FMM) of Greengard and Rokhlin [7], it has become clear that many large matrices that arise in practice have a complex low-rank structure in their submatrices that can be exploited efficiently to speed up matrix algorithms. In particular, such structured matrices arise in the numerical solution of integral equations, as fill-in during Gaussian elimination of sparse matrices that come from the discretization of elliptic PDEs, and in many other applications. In earlier work [2] we introduced techniques to design fast and stable direct solvers for such structured matrices based on an implicit ULV factorization algorithm and a matrix representation that we called hierarchically semiseparable (HSS). In this paper we show that linear systems of equations involving such dense structured matrices can be efficiently converted into a larger sparse system of equations that has an ordering of the unknowns permitting a very efficient direct Gaussian elimination solver to be used. This technique has several advantages. First, it makes it possible to exploit the highly developed sparse direct solver technology to attack dense structured problems. Second, it provides a theoretical tool to study these large dense structured matrices. However, in this paper we just concentrate on showing how this technique can be used to design a fast, stable solver for matrices in HSS form only.

*Received by the editors August 26, 2005; accepted for publication (in revised form) by D. A. Bin May 30, 2006; published electronically December 21, 2006.
http://www.siam.org/journals/simax/29-1/63902.html

†Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106 (shiv@ece.ucsb.edu, tpals@engineering.ucsb.edu). The research of these authors was supported in part by NSF grant CCR-0204388.

‡Faculty of Electrical Engineering, Delft University of Technology, Delft, The Netherlands (p.dewilde@its.tudelft.nl).

§Department of Mathematics, University of California at Berkeley, Berkeley, CA 94704 (mgu@math.berkeley.edu). The research of this author was supported in part by NSF grant CCR-0204388.

¶Department of Mathematics, University of California at Santa Barbara, Santa Barbara, CA 93106 (lyons@math.ucsb.edu). The research of this author was supported in part by NSF grant CCR-0204388.

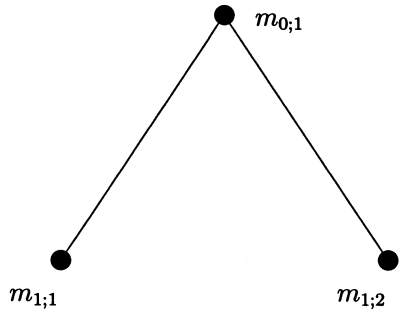

FIG. 1. *One level HSS partition tree with* $m_{0;1} = m_{1;1} + m_{1;2}$.

The idea of explicitly using sparse representations of low-rank structured matrices seems to have first originated in the use of diagonal algebras in time-varying systems theory [5]. Of course, such representations are implicit even in the original FMM papers [7].

2. HSS representations. Usually an $m \times n$ matrix A is represented in terms of its mn entries $A_{i,j}$. The HSS representation of A is another way to present the same information. It tries to exploit the presence of low- (numerical) rank submatrices in A. Of course this presumes that we know which submatrices are potentially of low rank. Fortunately, in the application that we have in mind, namely, the numerical solution of elliptic PDEs, this information is usually available. In particular, the HSS representation assumes that the matrix has its low-rank submatrices in the off-diagonal regions. Historically the HSS representation is just a special case of the representations commonly exploited in the FMM literature.

The HSS representation depends directly on a recursive block partitioning of the matrix. It is natural to use a tree to represent these partitions. Suppose at the first level the matrix is partitioned as follows:

$$A = \begin{array}{c} \\ m_{1;1} \\ m_{1;2} \end{array} \begin{array}{c} \begin{array}{cc} m_{1;1} & m_{1;2} \end{array} \\ \begin{pmatrix} A_{1;1,1} & A_{1;1,2} \\ A_{1;2,1} & A_{1;2,2} \end{pmatrix} \end{array}.$$

Then the corresponding HSS partition tree is shown in Figure 1 where it is assumed that A is an $m_{0;1} \times m_{0;1}$ matrix.

The HSS representation tries to exploit the low (numerical) rank of the off-diagonal blocks. The one level HSS tree, for example, is based on the partitioning

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

where clearly the factorization of the off-diagonal blocks can be chosen to be rank-revealing. The tree is shown in Figure 2. At this stage it is not quite clear why, for example, $U_{1;1}$ is written at the first leaf node. One reason is that that particular node corresponds to the first $m_{1;1}$ rows of the matrix, and $U_{1;1}$ is associated with (a portion) of those rows. A better reason will become obvious once we get into section 3. Similarly the expansion coefficient $B_{1;1,2}$ is placed on the edge connecting

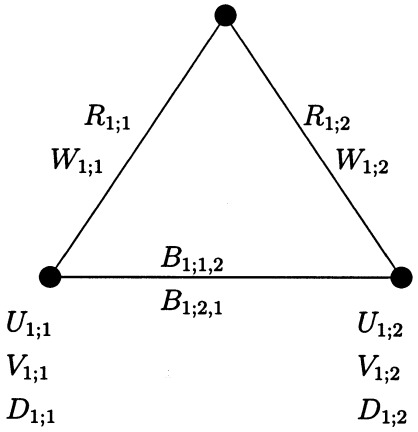

FIG. 2. *One level HSS.*

the two leaves because it sits at the intersection of the rows corresponding to the first leaf node and the columns corresponding to the second leaf node. The matrices $R_{1;i}$ and $W_{1;i}$ have no columns at all and will be explained shortly.

The two level HSS representation is based on the partition

$$A = \begin{array}{c} \begin{array}{cc} (\, m_{2;1} & m_{2;2}\,) \end{array} \\ \begin{pmatrix} m_{2;1} \\ m_{2;2} \end{pmatrix} \\ \begin{pmatrix} m_{2;3} \\ m_{2;4} \end{pmatrix} \end{array} \begin{array}{c} \begin{array}{cc} (\, m_{2;3} & m_{2;4}\,) \end{array} \\ \begin{pmatrix} \begin{pmatrix} A_{2;1,1} & A_{2;1,2} \\ A_{2;2,1} & A_{2;2,2} \end{pmatrix} & A_{1;1,2} \\ A_{1;2,1} & \begin{pmatrix} A_{2;3,3} & A_{2;3,4} \\ A_{2;4,3} & A_{2;4,4} \end{pmatrix} \end{pmatrix} \end{array},$$

where $m_{1;i} = m_{2;2i-1} + m_{2;2i}$ for $i = 1, 2$. The matrices that make up the two level HSS form of A are in turn inferred from the equation

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

However, the matrices $U_{1;i}$ and $V_{1;i}$ are not part of the two level HSS representation. And, equally importantly, $U_{2;i}$ ($V_{2;i}$) is not chosen as a column (row) basis for $A_{2;i,j}$ ($A_{2;j,i}$). Rather we define *translation* operators $R_{2;i}$ and $W_{2;i}$ such that

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

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

Notice that for this to be possible we must choose $U_{2;i}$ such that it forms a column basis for the submatrix

$$(\, A_{2;i,1} \quad \cdots A_{2;i,i-1} \quad A_{2;i,i+1} \quad \cdots \quad A_{2;i,4}\,).$$

Notice that we obtain the above matrix by taking the ith block row from the second level partition of A and dropping the diagonal block $A_{2;i,i}$. Similarly we choose $V_{2;i}$

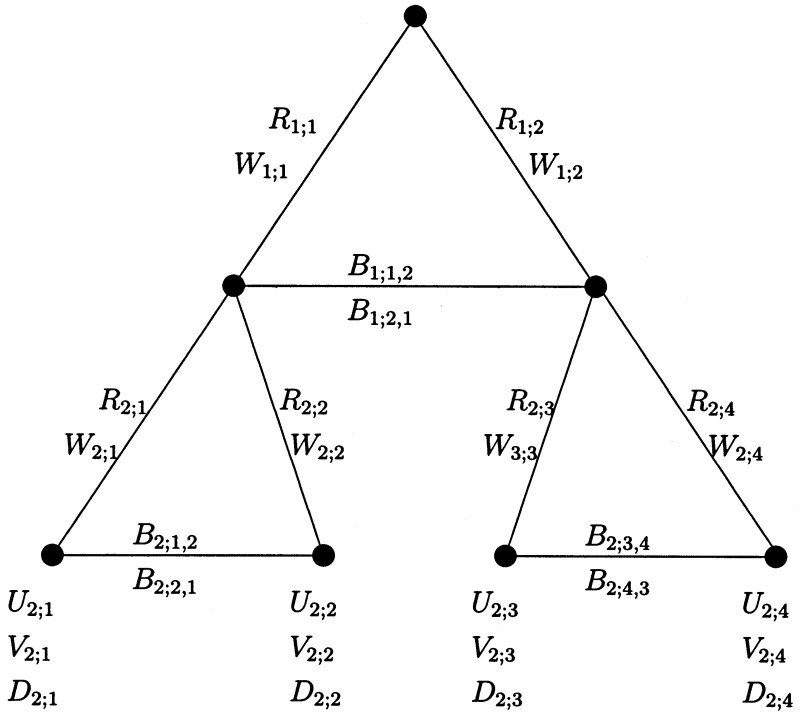

FIG. 3. *Two level HSS.*

to be a row basis for the submatrix

$$\begin{pmatrix} A_{2;1,i} \\ \vdots \\ A_{2;i-1,i} \\ A_{2;i+1,i} \\ \vdots \\ A_{2;4,i} \end{pmatrix}.$$

The two level HSS tree is shown in Figure 3. The translation operators are placed at the edges from the leaves of the second level to their parents in the first level to reflect (1) and (2). The translation operators $R_{1;i}$ and $W_{1;i}$ are not important and are usually chosen to be matrices with no columns at all.

Generally speaking, an HSS representation is a finite binary tree of the type shown in Figure 3, where the dimensions of the matrices at the nodes and leaves must be chosen according to the restrictions that these assemblies

$$\begin{pmatrix} B_{k;2i-1,2i} & R^H_{k+1;2i-1} \\ W_{k+1;2i} & 0 \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} B_{k;2i,2i-1} & R^H_{k+1;2i} \\ W_{k+1;2i-1} & 0 \end{pmatrix}$$

are possible if the node (k, i) is not a leaf,[1] and if it is a leaf, then the assembly

$$\begin{pmatrix} D_{k;i} & U_{k;i} \\ V^H_{k;i} & 0 \end{pmatrix}$$

[1]The node (k, i) is the ith node counting from the left at level k in the tree.

and the multiplications

$$U_{k;i}R_{k;i} \quad \text{and} \quad V_{k;i}W_{k;i}$$

must be possible. In this paper we always assume that $R_{1;i}$ and $W_{1;i}$ have no columns at all.

Given an arbitrary HSS tree and an arbitrary matrix A with the right number of rows and columns, one can *always* find an HSS representation for A conforming with the HSS tree. The $O(n^2)$ flops algorithm to carry this out is presented in [2].

3. Fast multiplication. The key to the fast inversion algorithm is the fast algorithm for multiplying a matrix in HSS form with a vector. In particular, it is the recursions for the multiplication algorithm that are the key. The recursions we present are exactly the same as those used in the FMM [7].

To be concrete assume that the HSS form of the matrix A is available and that we want to multiply it rapidly with the vector x to obtain $Ax = b$. Of course one method is to first get the componentwise entries $A_{i,j}$ of A and to then use a conventional algorithm. However, that would not be the most efficient thing to do.

Rather, we first observe that we need to multiply submatrices of x with $V_{k;i}$ for each node in the HSS tree. Of course some of these $V_{k;i}$'s are not directly available, namely, those on the nonleaf nodes, but we can get around that using the translation operators $W_{k;i}$. Before we get into the details we need some notation. We will assume that $x_{k;i}$ denotes a submatrix of x partitioned according to the kth level of the HSS tree. That is,

$$x = \begin{matrix} m_{1;1} \\ m_{1;2} \end{matrix} \begin{pmatrix} x_{1;1} \\ x_{1;2} \end{pmatrix},$$

and

$$x = \begin{matrix} m_{2;1} \\ m_{2;2} \\ m_{2;3} \\ m_{2;4} \end{matrix} \begin{pmatrix} x_{2;1} \\ x_{2;2} \\ x_{2;3} \\ x_{2;4} \end{pmatrix},$$

and so on.

Now we observe that at the leaf node (k, i) we can compute

$$g_{k;i} = V_{k;i}^H x_{k;i}.$$

If (k, i) is not a leaf node we can infer

$$\begin{aligned} g_{k;i} &= V_{k;i}^H x_{k;i} \\ &= \begin{pmatrix} V_{k+1;2i-1}W_{k+1;2i-1} \\ V_{k+1;2i}W_{k+1;2i} \end{pmatrix}^H \begin{pmatrix} x_{k+1;2i-1} \\ x_{k+1;2i} \end{pmatrix} \\ &= W_{k+1;2i-1}^H V_{k+1;2i-1}^H x_{k+1;2i-1} + W_{k+1;2i}^H V_{k+1;2i}^H x_{k+1;2i} \\ &= W_{k+1;2i-1}^H g_{k+1;2i-1} + W_{k+1;2i}^H g_{k+1;2i}. \end{aligned}$$

We see therefore that $g_{k;i} = V_{k;i}^H x_{k;i}$ can be computed at each node of the HSS tree very efficiently via the set of equations

$$g_{k;i} = V_{k;i}^H x_{k;i} \qquad \text{at a leaf,} \tag{3}$$

$$= W_{k+1;2i-1}^H g_{k+1;2i-1} + W_{k+1;2i}^H g_{k+1;2i} \qquad \text{at a nonleaf node.} \tag{4}$$

To complete the multiplication let us look in detail at $b_{2;1}$ for a two level HSS tree

$$b_{2;1} = D_{2;1}x_{2;1} + U_{2;1}B_{2;1,2}g_{2;2} + U_{2;1}R_{2;1}B_{1;1,2}g_{1;2},$$

which we can regroup more carefully as follows:

$$b_{2;1} = D_{2;1}x_{2;1} + U_{2;1}(B_{2;1,2}g_{2;2} + R_{2;1}B_{1;1,2}g_{1;2}).$$

This suggests that we define the auxiliary variables $f_{k;i}$ such that

$$b_{k;i} = A_{k;i,i}x_{k;i} + U_{k;i}f_{k;i}.$$

Of course if (k,i) is not a leaf, then we will not have access to the diagonal block $A_{k;i,i}$ or $U_{k;i}$. But in that case we see that we can split the equation using the translation operators $R_{k;i}$ as follows:

$$\begin{pmatrix} b_{k+1;2i-1} \\ b_{k+1;2i} \end{pmatrix} = \begin{pmatrix} A_{k+1;2i-1,2i-1} & U_{k+1;2i-1}B_{k+1;2i-1,2i}V_{k+1;2i}^H \\ U_{k+1;2i}B_{k+1;2i,2i-1}V_{k+1;2i-1}^H & A_{k+1;2i,2i} \end{pmatrix} \cdot \begin{pmatrix} x_{k+1;2i-1} \\ x_{k+1;2i} \end{pmatrix} + \begin{pmatrix} U_{k+1;2i-1}R_{k+1;2i-1} \\ U_{k+1;2i}R_{k+1;2i} \end{pmatrix} f_{k;i},$$

which simplifies to the pair of equations

$$b_{k+1;2i-1} = A_{k+1;2i-1,2i-1}x_{k+1;2i-1} + U_{k+1;2i-1}\left(B_{k+1;2i-1,2i}g_{k+1;2i} + R_{k+1;2i-1}f_{k;i}\right),$$
$$b_{k+1;2i} = A_{k+1;2i,2i}x_{k+1;2i-1,2i} + U_{k+1;2i}\left(B_{k+1;2i,2i-1}g_{k+1;2i-1} + R_{k+1;2i-}f_{k;i}\right).$$

This does not seem to lead anywhere, but in fact it does tell us that the recursive equations for the auxiliary variables $f_{k;i}$ are

$$f_{k+1;2i-1} = B_{k+1;2i-1,2i}g_{k+1;2i} + R_{k+1;2i-1}f_{k;i}, \tag{5}$$

$$f_{k+1;2i} = B_{k+1;2i,2i-1}g_{k+1;2i-1} + R_{k+1;2i}f_{k;i}. \tag{6}$$

This looks good, but how do we start off the recursion? In other words, what is $f_{0;1}$? Let us look at its defining equation

$$b = Ax = b_{0;1} = A_{0;1}x_{0;1} + U_{0;1}f_{0;1},$$

which implies that

$$f_{0;1} = (\ ), \tag{7}$$

the empty matrix! Of course at the leaf level we can directly compute the outputs from

$$b_{k;i} = D_{k;i}x_{k;i} + U_{k;i}f_{k;i}. \tag{8}$$

With that we have a complete set of efficient recursions for computing $Ax = b$ given x and the HSS form of A.

4. Sparse representation. We will now make the effort to write the multiplication recursions in a compact form using matrix notation, and without indices. This will turn out to be the key step that can reveal the way to the fast solver.

We first define some block diagonal matrices. Let $\mathbf{D}$ be a block diagonal matrix formed by ordering $D_{k;i}$ in, say, breadth-first order.[2] Similarly we define block diagonal matrices $\mathbf{U}$ and $\mathbf{V}$. For example, for a two level HSS form, we would have

$$\mathbf{U} = \begin{pmatrix} U_{2;1} & & & \\ & U_{2;2} & & \\ & & U_{2;3} & \\ & & & U_{2;4} \end{pmatrix}.$$

We also arrange all the translation operators $R_{k;i}$ in a block diagonal matrix $\mathbf{R}$, in breadth-first order. Note that there is one $R_{k;i}$ per parent node. So $\mathbf{R}$ will be a block diagonal matrix with a potentially different number of diagonal blocks than, say, $\mathbf{U}$. Similarly we define the block diagonal matrix $\mathbf{W}$. For example, for a two level HSS representation we would have

$$\mathbf{W} = \begin{pmatrix} W_{1;1} & & & & & & \\ & W_{1;2} & & & & & \\ & & W_{2;1} & & & & \\ & & & W_{2;2} & & & \\ & & & & W_{2;3} & & \\ & & & & & W_{2;4} & \end{pmatrix}.$$

We also arrange the $B_{k;i,j}$ in a block diagonal matrix $\mathbf{B}$, with the $B_{k;i,j}$ in breadth-first order, and within a node we place $B_{k+1;2i-1,2i}$ before $B_{k+1;2i,2i-1}$. So for a two level HSS form we would have

$$\mathbf{B} = \begin{pmatrix} B_{1;1,2} & & & & & & \\ & B_{1;2,1} & & & & & \\ & & B_{2;1,2} & & & & \\ & & & B_{2;2,1} & & & \\ & & & & B_{2;3,4} & & \\ & & & & & B_{2;4,3} & \end{pmatrix}.$$

We next define the shift-down operator $Z_\downarrow$ on trees. Given a binary tree with matrices on each node, the action of $Z_\downarrow$ on the binary tree is to produce an identical tree in which the matrix on every parent node has been moved into the children. The matrices at the leaves are dropped off. The root node acquires a zero matrix. For example for a two level HSS tree the action of $Z_\downarrow$ (in the depth-first order for input and output) is expressed by the following equation corresponding to Figure 4:

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}}_{Z_\downarrow} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{pmatrix} = \begin{pmatrix} 0 \\ a \\ a \\ b \\ b \\ c \\ c \end{pmatrix}. \tag{9}$$

[2] We are free to pick this order, but once we have chosen an order we must stick with it for the remaining diagonal matrices too.

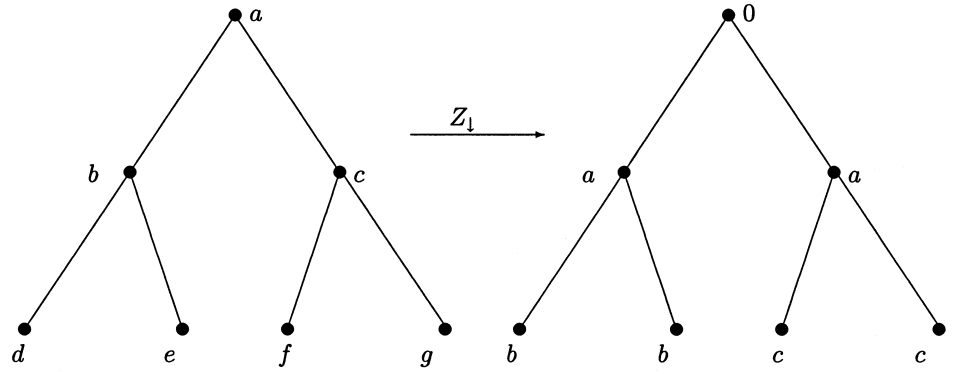

FIG. 4. *Action of $Z_\downarrow$ on a two level HSS tree.*

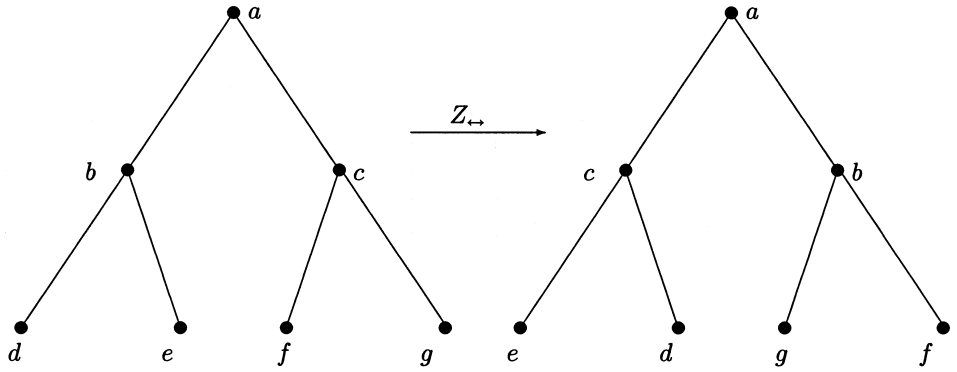

FIG. 5. *Action of $Z_\leftrightarrow$ on a two level HSS tree.*

As can be seen $Z_\downarrow$ is very sparse and noninvertible.

Now we define the twiddle operator $Z_\leftrightarrow$ on trees. When $Z_\leftrightarrow$ acts on a binary tree with matrices on each node, it exchanges the matrices on sibling nodes. The following equation gives an explicit representation for $Z_\leftrightarrow$ on a two level HSS tree (which is shown pictorially in Figure 5):

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}}_{Z_\leftrightarrow} \begin{pmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \end{pmatrix} = \begin{pmatrix} a \\ c \\ b \\ e \\ d \\ g \\ f \end{pmatrix}.$$

Note that $Z_\leftrightarrow$ is a permutation matrix (which are always very sparse, of course).

Now let us assign the intermediate quantities $g_{k;i}$ and $f_{k;i}$ to the corresponding nodes on the HSS tree. Naturally we can then stack them up in breadth-first ordering in a single block vector and call them $\mathbf{g}$ and $\mathbf{f}$. For example, for the two level HSS

tree we would have

$$\mathbf{f} = \begin{pmatrix} f_{0;1} \\ f_{1;1} \\ f_{1;2} \\ f_{2;1} \\ f_{2;2} \\ f_{2;3} \\ f_{2;4} \end{pmatrix}.$$

We also need to define a projection operator $\mathbf{P}_{\text{leaf}}$ that acting on a block vector like $\mathbf{f}$ would return the restriction of it to the leaf nodes. For example, for the two level HSS tree we would have

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{P}_{\text{leaf}}} \begin{pmatrix} f_{0;1} \\ f_{1;1} \\ f_{1;2} \\ f_{2;1} \\ f_{2;2} \\ f_{2;3} \\ f_{2;4} \end{pmatrix} = \begin{pmatrix} f_{2;1} \\ f_{2;2} \\ f_{2;3} \\ f_{2;4} \end{pmatrix}. \tag{10}$$

We also define $\mathbf{x}$ and $\mathbf{b}$ as the two block vectors obtained by arranging the submatrices of x and b according to the leaf partitions of the HSS tree. So, for example, in the case of a two level HSS tree we would have

$$\mathbf{x} = \begin{pmatrix} x_{2;1} \\ x_{2;2} \\ x_{2;3} \\ x_{2;4} \end{pmatrix}.$$

Of course this is just x in this case. But if we had ordered the tree nodes (and hence leaves) in some other order this may not have been the case.

With these new matrices we can rewrite the fast multiplication recursions in compact form. Let us start with the pair (3) and (4) which can be written together as

$$\mathbf{g} = \mathbf{P}_{\text{leaf}}^H \mathbf{V}^H \mathbf{x} + Z_{\downarrow}^H \mathbf{W}^H \mathbf{g}. \tag{11}$$

It is very important for the reader to understand why the single equation above is exactly equivalent to the pair (3) and (4). For example, let us check if (3) is captured correctly.

To do that we can apply the leaf projection operator from the left in (11) and obtain

$$\mathbf{P}_{\text{leaf}} \mathbf{g} = \mathbf{P}_{\text{leaf}} \mathbf{P}_{\text{leaf}}^H \mathbf{V}^H \mathbf{x} + \mathbf{P}_{\text{leaf}} Z_{\downarrow}^H \mathbf{W}^H \mathbf{g}. \tag{12}$$

We need to understand the significance of $\mathbf{P}_{\text{leaf}}^H$ and $Z_{\downarrow}^H$ and their relationship to $\mathbf{P}_{\text{leaf}}$.

For example, $\mathbf{P}_{\text{leaf}}^H$ is the pseudoinverse of $\mathbf{P}_{\text{leaf}}$, as $\mathbf{P}_{\text{leaf}}$ is an orthogonal projector

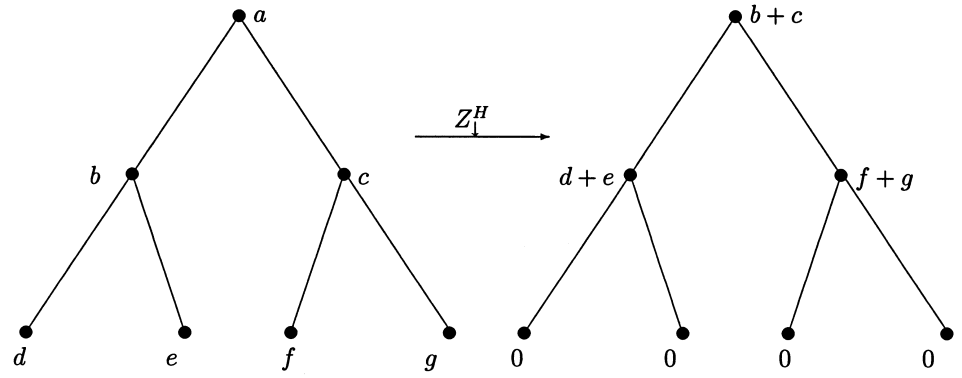

FIG. 6. *Action of $Z_{\downarrow}^H$ on a two level HSS tree.*

"onto the leaves of the HSS tree." So if we look at the example in (10) we have

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ f_{2;1} \\ f_{2;2} \\ f_{2;3} \\ f_{2;4} \end{pmatrix} = \mathbf{P}_{\text{leaf}}^H \begin{pmatrix} f_{2;1} \\ f_{2;2} \\ f_{2;3} \\ f_{2;4} \end{pmatrix}.$$

From this we can see that $\mathbf{P}_{\text{leaf}}^H$ *embeds* a block vector inside an HSS tree with the blocks assigned to the leaves and zeros assigned to the parent nodes. It follows that $\mathbf{P}_{\text{leaf}}\mathbf{P}_{\text{leaf}}^H = I$.

Next we look at $Z_{\downarrow}^H$. Since the action of $Z_{\downarrow}$ on an HSS tree is to move the vectors at the parent node down into the child nodes, it is not surprising to learn that $Z_{\downarrow}^H$ does nearly the opposite; it adds the vectors in the child nodes together and assigns them to the parent nodes, while the leaf nodes are assigned zeros. This is depicted in Figure 6. From this it follows that $\mathbf{P}_{\text{leaf}}Z_{\downarrow}^H = 0$.

Putting all this together we see that (12) can be simplified to

$$\mathbf{P}_{\text{leaf}}\mathbf{g} = \mathbf{V}^H\mathbf{x},$$

which is exactly (3) written using block matrices.

Next we quickly describe how (4) is embedded in (11). For this we need to consider the nonleaf nodes on both sides of (11). We can do so, for example, by multiplying both sides by $I - \mathbf{P}_{\text{leaf}}^H\mathbf{P}_{\text{leaf}}$. The latter acts on an HSS tree by setting all the vectors at the leaf nodes to zero. From this, and our earlier description of $\mathbf{P}_{\text{leaf}}^H$ and $Z_{\downarrow}^H$, it is easy to verify that $(I - \mathbf{P}_{\text{leaf}}^H\mathbf{P}_{\text{leaf}})\mathbf{P}_{\text{leaf}}^H = 0$ and $(I - \mathbf{P}_{\text{leaf}}^H\mathbf{P}_{\text{leaf}})Z_{\downarrow}^H = Z_{\downarrow}^H$. Therefore when we multiply both sides of (11) by $(I - \mathbf{P}_{\text{leaf}}^H\mathbf{P}_{\text{leaf}})$ we obtain

$$(I - \mathbf{P}_{\text{leaf}}^H\mathbf{P}_{\text{leaf}})\mathbf{g} = Z_{\downarrow}^H\mathbf{W}^H\mathbf{g},$$

which when written out componentwise for each nonleaf node yields (4).

Next we observe that (7), (5), and (6) can be combined and written as the single equation

$$\mathbf{f} = \mathbf{R}Z_{\downarrow}\mathbf{f} + \mathbf{B}Z_{\leftrightarrow}\mathbf{g}. \tag{13}$$

Finally, we can write the output (8) as

$$\mathbf{b} = \mathbf{D}\mathbf{x} + \mathbf{U}\mathbf{P}_{\text{leaf}}\mathbf{f}. \tag{14}$$

It is more convenient to combine the three equations (11), (13), and (14) into the single equation

$$\underbrace{\begin{pmatrix} \mathbf{D} & 0 & \mathbf{U}\mathbf{P}_{\text{leaf}} \\ 0 & \mathbf{B}Z_{\leftrightarrow} & \mathbf{R}Z_{\downarrow} - I \\ \mathbf{P}_{\text{leaf}}^H\mathbf{V}^H & Z_{\downarrow}^H\mathbf{W}^H - I & 0 \end{pmatrix}}_{\mathbf{S}} \begin{pmatrix} \mathbf{x} \\ \mathbf{g} \\ \mathbf{f} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ 0 \\ 0 \end{pmatrix}. \tag{15}$$

We first observe that the matrix $\mathbf{S}$ is extremely *block* sparse. In particular, $\mathbf{S}$ is a block matrix with at most three nonzero blocks in every block row. For example, the first block row of (15) reads

$$D_{2;1}x_{2;1} + U_{2;1}f_{2;1} = b_{2;1},$$

and it shows that $\mathbf{S}$ has only two nonzero blocks, $D_{2;1}$ and $U_{2;1}$, in its first block row. The general observation, that $\mathbf{S}$ has at most three nonzero blocks in any block row, follows from the recursions for the fast multiplication algorithm, (3) to (7).

It is now convenient to look at a graph representation of $\mathbf{S}$. We will use the standard one from text books (see [1, section 6.4.2]). Usually sparse matrices are viewed elementwise and the corresponding graphs have elements on the edges. In our case it is best to view $\mathbf{S}$ as a block sparse matrix and to look at the corresponding graph instead. First we observe that even though $\mathbf{S}$ is not a Hermitian matrix, its nonzero blocks form a structurally symmetric matrix; that is, if the (i, j) block of $\mathbf{S}$ is a structural nonzero block, then the (j, i) block is also a structural nonzero block. Therefore, we can use an undirected graph to represent the block sparsity of $\mathbf{S}$. An example of the graph we will use for a two level HSS form is shown in Figure 7. The graph is set up as follows. First certain block rows and columns are assigned (or associated) with a node of the graph. For example, in Figure 7, the block column corresponding to the unknowns $f_{0;1}$ and $g_{0;1}$ is assigned to the topmost node in the figure. Similarly the block column corresponding to the unknowns $f_{2;1}$, $g_{2;1}$, and $x_{2;1}$ is assigned to the bottom leftmost leaf node. Once a block column has been associated with a node of the graph the corresponding block row is also associated with the same node. Note that the nodes for the graph representation of $\mathbf{S}$ are exactly the nodes of the HSS tree. This is not a coincidence. Once the nodes have been assigned the edges for the graph are picked according to the structural nonzero blocks of $\mathbf{S}$. For example, the equation

$$g_{1;2} - W_{2;3}^H g_{2;3} - W_{2;4}^H g_{2;4} = 0$$

is one of the rows of the equation expressed by (15). Since this equation connects the block variable $g_{2;3}$ with the block variable $g_{1;2}$ there is an edge in the graph of Figure 7 between the two nodes connecting these two variables. We do a similar thing for every block row equation of (15), drawing an edge between two nodes of the graph if there is a block equation connecting unknowns in the two nodes. The resulting graph representation for $\mathbf{S}$ for a two level HSS form is shown in Figure 7. The assignment of block rows and columns in this figure might seem arbitrary but the intention will become clear soon. Definitely one of the reasons was to make clear that

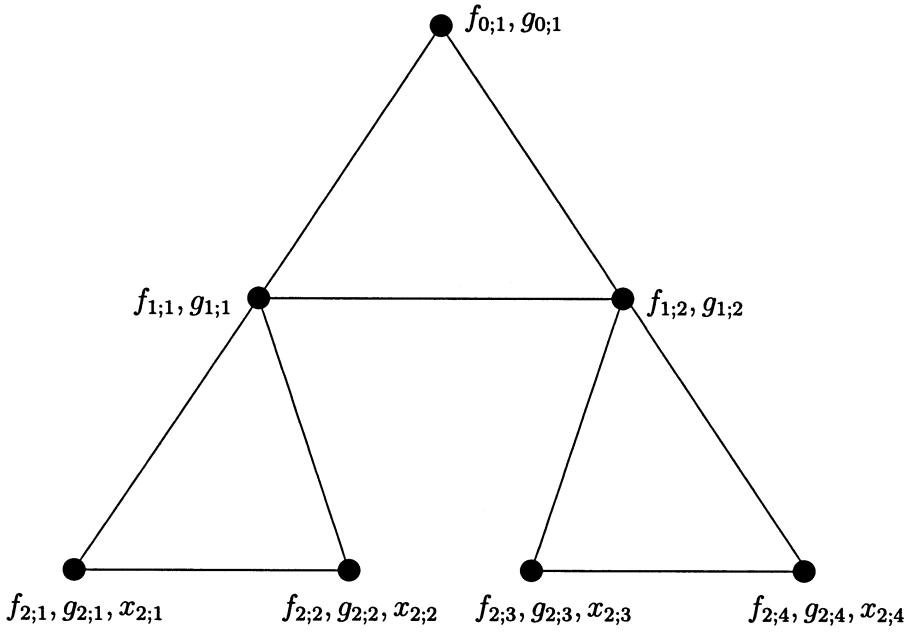

FIG. 7. *Block sparse graph of* $\mathbf{S}$ *arising from two level HSS form.*

the graph representing the sparsity of $\mathbf{S}$ is closely related to the graph representing the HSS form of A.

Since $\mathbf{S}$ is very sparse it naturally raises the idea that we could solve the sparse system of equations (15) for x efficiently using a *standard* sparse solver. However, to establish that we must first establish that the system is invertible (if the original matrix A is) and that we will not incur too much fill-in during Gaussian elimination on $\mathbf{S}$.

We begin with the first issue: does $\mathbf{S}^{-1}$ exist whenever A^{-1} exists? While resolving this question we will discover a remarkable diagonal formula for HSS representations. First, observe that the bottom 2×2 principal submatrix of $\mathbf{S}$ is invertible with an inverse given by the explicit formula

$$\begin{pmatrix} \mathbf{B}Z_{\leftrightarrow} & \mathbf{R}Z_{\downarrow} - I \\ Z_{\downarrow}^H \mathbf{W}^H - I & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & (Z_{\downarrow}^H \mathbf{W}^H - I)^{-1} \\ (\mathbf{R}Z_{\downarrow} - I)^{-1} & -(\mathbf{R}Z_{\downarrow} - I)^{-1} \mathbf{B}Z_{\leftrightarrow} (Z_{\downarrow}^H \mathbf{W}^H - I)^{-1} \end{pmatrix}.$$

Of course the validity of this formula hinges upon the existence of the two inverses

$$(\mathbf{W}Z_{\downarrow} - I)^{-1} \quad \text{and} \quad (\mathbf{R}Z_{\downarrow} - I)^{-1}.$$

But these two inverses always exist. The reasoning is as follows. We see from (9) that $Z_{\downarrow}$ is nilpotent. Since $\mathbf{W}$ and $\mathbf{R}$ are block diagonal matrices with block sizes chosen to be compatible with the block identities in $Z_{\downarrow}$, it follows that $\mathbf{W}Z_{\downarrow}$ and $\mathbf{R}Z_{\downarrow}$ are also nilpotent matrices. From this it follows that the above two inverses always exist. This proves our assertion.

Now we can *always* solve (15) for $\mathbf{x}$ and obtain

$$\left(\mathbf{D} + \mathbf{U}\mathbf{P}_{\text{leaf}}(I - \mathbf{R}Z_{\downarrow})^{-1}\mathbf{B}Z_{\leftrightarrow}(I - Z_{\downarrow}^H \mathbf{W}^H)^{-1}\mathbf{P}_{\text{leaf}}^H \mathbf{V}^H\right)\mathbf{x} = \mathbf{b}. \tag{16}$$

Since this is true for all x, it follows that

$$A = \mathbf{D} + \mathbf{U}\mathbf{P}_{\text{leaf}}(I - \mathbf{R}Z_{\downarrow})^{-1}\mathbf{B}Z_{\leftrightarrow}(I - Z_{\downarrow}^{H}\mathbf{W}^{H})^{-1}\mathbf{P}_{\text{leaf}}^{H}\mathbf{V}^{H}. \tag{17}$$

This is a compact diagonal representation of the HSS form of A. It therefore follows that if A is invertible then the sparse matrix $\mathbf{S}$ in (15) is also invertible, since A is just the $(1,1)$ Schur complement of $\mathbf{S}$.

So it is clear that to solve $Ax = b$ for x we could solve the sparse system of equations (15) instead. But to establish that there is a computational advantage in doing so, we must show that the sparse system (15) has an ordering that will not fill in during Gaussian elimination. Of course if we first eliminate $\mathbf{f}$ and then $\mathbf{g}$, we will get exactly A, which is the original matrix and completely filled in!

To find a better ordering we look at the block sparse graph for the system of which an example for the two level HSS representation is shown in Figure 7. From that figure it is obvious that there will be no block fill-in for the nested dissection ordering of the unknowns, that is, if we eliminate in the following block order: $(\, f_{2;1} \quad g_{2;1} \quad x_{2;1}\,)$, $(\, f_{2;2} \quad g_{2;2} \quad x_{2;2}\,)$, $(\, f_{2;3} \quad g_{2;3} \quad x_{2;3}\,)$, $(\, f_{2;4} \quad g_{2;4} \quad x_{2;4}\,)$, $(\, f_{1;1} \quad g_{1;1}\,)$, $(\, f_{1;2} \quad g_{1;2}\,)$, $(\, f_{0;1} \quad g_{0;1}\,)$. To see why this is so, note that after eliminating the variables $f_{2;1}$, $g_{2;1}$, and $x_{2;1}$, for example, the remaining equations will have no new nonzero blocks (see [1, sections 6.4.4 and 6.5.3] for further explanations on how determine fill-in during Gaussian elimination from the graph representation).

In general, in the nested dissection ordering all the variables on the left subtree are ordered before all the variables in the right subtree, with the variables on the root node coming last. Of course, the variables in the left and right subtrees are themselves ordered recursively in nested dissection order.

The bottom line is that there *exists* a no fill-in Gaussian elimination order. But what about pivoting to ensure numerical stability? That is a more complicated question, and we do not answer it here. Rather, we just observe that the block sparse graph also shows that we can get an efficient sparse QR factorization in the nested dissection ordering.

To see this let us follow through the first step of a block Givens QR factorization algorithm on the two level HSS form shown in Figure 7. We first try to eliminate the node containing $f_{2;1}$, $g_{2;1}$, and $x_{2;1}$. We have to first apply a block Givens rotation involving the pivot row and the row corresponding to the variables $f_{2;2}$, $g_{2;2}$, and $x_{2;2}$. We note that the only possible fill-in in this row and the pivot row must correspond to the variables $f_{1;1}$ and $g_{1;1}$. But both these positions are already nonzero, so no fill-in edges have to be added to the graph. If we now proceed with the nested dissection ordering, we can argue similarly that no fill-in edges at all will be added to the block sparse graph (see [1, section 6.6.4]).

Hence, to avoid numerical instabilities, we can just use a sparse QR factorization method. Since there is essentially no fill-in, we obtain a solver that is numerically stable and is linear in the dimension of the matrix A, with a constant that depends on the size of the $B_{k;i,j}$ matrices. In particular, the number of flops is a constant times the sum of the cube of the sizes of the $B_{k;i,j}$'s. This can be inferred as follows.

First note that every block equation has at most three block terms. Therefore, every stage of the block QR factorization involves a constant number of matrix multiplications. For the sake of simplicity, let us consider the case when every matrix in the HSS form is no bigger than $p \times p$ for some integer p. Then, it is clear that at every stage of the block QR algorithm, the number of flops will not exceed some constant times p^3. Since there is no fill-in during the QR factorization, it also follows that the

total number of flops will not exceed the number of nodes in the HSS tree times p^3 times some constant. But the number of nodes in the HSS tree cannot exceed n/p times some constant, where n is the dimension of the matrix A. Therefore, in this case, the number of flops is $O(np^2)$.

We caution that to construct an HSS representation from a dense matrix will in general require $O(n^2)$ flops (using the algorithm presented in [2], for example). The same paper [2] also describes examples where the HSS representation can be computed in $O(n)$ flops. Similarly, the entire FMM literature can be viewed as a repository of examples where the FMM representation can be computed in $O(n)$ flops, and from which the HSS representation in turn can be computed in $O(n)$ flops.

5. Numerical experiments. We now describe some numerical experiments that exhibit the efficiency and the stability of the sparse solver approach. All experiments were carried out on a 1GHz PowerPC G4 machine with 1.5GB RAM and a 167MHz bus. We used the vendor supplied BLAS.

The $n \times n$ matrix A was chosen according to the formula $A_{i,j} = \sqrt{|x_i^{(n)} - x_j^{(n)}|}$, with the points $x_i^{(n)} = \cos(\pi(2i+1)/2n)$ as the zeros of the nth Chebyshev polynomial. The HSS tree was decided by a standard dyadic division of the interval $[-1, 1]$. The intervals were repeatedly divided in half until there were less than p points left. The value of p was chosen according to the matrix size to enable better memory behavior. Since the zeros of the Chebyshev polynomial cluster at the end points the resulting HSS tree was not uniform. We measure the skewness of the HSS tree as the ratio of the longest path (shortest distance) from a root to a leaf to the shortest path from a root to a leaf. The HSS form of the matrix was computed beforehand using the algorithm in the earlier paper [2] to eight digits of accuracy. It is well known that for matrices of the form we are considering in this experiment the ranks of the $B_{k;i,j}$'s are essentially proportional to the logarithm of the accuracy. Therefore, in this experiment the sizes of the $B_{k;i,j}$'s were essentially constant, independent of the matrix size. Therefore, we should expect the CPU time of the solver to scale linearly in the matrix size.

The experimental data are reported in Table 1. The first column shows that we tried matrices that varied in size from 256×256 to 131072×131072. The second column shows the factor p that decides the maximum number of rows (and columns) in a leaf node. The skewness of the HSS tree of the various matrices that we tried is shown in column three. The fourth column shows the CPU time required by the sparse solver.

The sparse solver we used was a custom built block QR solver. We ordered the sparse matrix in (15) in nested dissection order. As can be seen from column four of the table the solver is essentially a linear time solver as predicted by the theory, and that it is not affected adversely by the skewness of the HSS tree.

In column five we show the backward error for each solve. The backward error for solving the system $Ax = b$ with computed solution $\hat{x}$ is defined to be the ratio of the smallest 2-norm of any matrix E that satisfies the equation $(A+E)\hat{x} = b$, and the 2-norm of A (see [1, section 1.4.6]). As can be seen from column five, the backward error for our method is essentially machine precision. This shows that the method behaved in a backward stable manner in this set of experiments.

6. Conclusion. We have shown that a fast direct solver for linear systems of equations with the coefficient matrix in HSS form can be easily constructed from a sparse solver. The resulting algorithm is fast and stable.

It is easy to see that this idea can easily be extended to handle more complex

TABLE 1
Speed and stability of sparse solver for HSS forms.

Matrix size	Leaf block size	Skewness of tree	CPU time in seconds	Backward error
256	13	2	0.07	9.90765e-17
512	14	1.8	0.15	1.01727e-16
1024	15	1.83333	0.32	2.86709e-16
2048	16	1.85714	0.68	5.5083e-17
4096	17	1.875	1.43	8.24819e-17
8192	18	1.88889	2.87	4.0822e-17
16384	19	1.9	5.57	5.32472e-17
32768	20	2	11.29	4.96643e-17
65536	21	2	25.43	8.64522e-17
131072	22	2	53.88	8.51812e-17

partitions of the matrix than the one used in the HSS representation. In particular, the method can easily be extended to handle a full FMM representation of the matrix (see [10]), the hierarchical matrix representation (see [8, 9]), and the sequentially semiseparable representation (see [3, 4]). These matters will be presented in a companion paper.

REFERENCES

[1] A. BJÖRCK, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[2] S. CHANDRASEKARAN, M. GU, AND T. PALS, *A fast ULV decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[3] S. CHANDRASEKARAN, P. DEWILDE, M. GU, T. PALS, X. SUN, A.-J. VAN DER VEEN, AND D. WHITE, *Some fast algorithms for sequentially semiseparable representations*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 341–364.

[4] 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 HiPC 202, S. Sahni, ed., Lecture Notes in Comput. Sci. 2552, Springer-Verlag, Berlin, 2002, pp. 545–554.

[5] P. DEWILDE AND A. VAN DER VEEN, *Time-Varying Systems and Computations*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[6] I. GOHBERG, T. KAILATH, AND I. KOLTRACHT, *Linear complexity algorithms for semiseparable matrices*, Integral Equations Operator Theory, 8 (1985), pp. 780–804.

[7] L. GREENGARD AND V. ROKHLIN, *A fast algorithm for particle simulations*, J. Comput. Phys., 73 (1987), pp. 325–348.

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

[9] W. HACKBUSCH, B. N. KHOROMSKIJ, AND S. SAUTER, *On H^2-Matrices*, preprint 50, MPI, Leipzig, 1999.

[10] T. PALS, *Multipole for Scattering Computations: Spectral Discretization, Stabilization, Fast Solvers*, Ph.D. thesis, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, 2004.

[11] V. ROKHLIN, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), pp. 187–207.