Iterative solution methods based on the Hierarchically Semi-Separable Representation

Zhifeng Sheng, P.Dewilde and Nick van der Meijs
{z.sheng, p.dewilde, n.p.vandermeijs}@EWI.TUDelft.NL

***Abstract*— In this paper, we study an important class of structured matrices: "Hierarchically Semi-Separable (HSS)" matrices, for which an efficient hierarchical state based representation called Hierarchically Semi-Separable (HSS) representation can be used to utilize the data sparsity of the HSS matrices. A novel algorithm with $O(n)$ complexity is proposed to construct sub-optimal HSS representations from sparse matrices. Subsequently, the limitation of the direct HSS solution method is discussed in this paper, and a general strategy to combine standard iterative solution methods with the HSS representation is presented. We also describe a number of preconditioner construction algorithms based on the HSS representation. Our numerical experiments indicate that these iterative solution methods have linear complexity in computation time.**

***Index Terms*— Iterative solution method, hierarchically semi-separable representation, preconditioner**

I. Introduction

Matrix operation algorithms based on the HSS representation are quite interesting in its own right. Much of its research has been triggered by a problem that can be posed simply as: Given $A \in C^{m\times n}, b \in C^m$, find solution vector(s) $x \in C^n$ such that $Ax = b$. (If the system A is overdetermined or underdetermined, the Moore-Penrose solution will have to be computed.) Many scientific problems lead to the requirement to solve linear systems of equations or multiply a system matrix with a vector as part of computations. Thus, the efficient fast HSS solution algorithms as well as the fast matrix-vector multiplication algorithm are quite useful, given the computation can be done efficiently.

Unfortunately, matrix operation algorithms based on HSS representation are not always efficient, they are only so when the rank of the off-diagonal sub-matrices are small. In [1], [2], [3], the computation complexity analysis has been done with the assumption that the rank of the off-diagonal sub-matrices are neglectable; while in practice, the system matrix of 2D or 3D problems does not fit in this category. To reduce the rank of the off-diagonal sub-matrices, the rows and columns of the system matrix shall be reordered. Graph theory has been of great help in re-arranging the non-zero entries of sparse matrices to reduce the fill-in or to move as many non-zeros as possible onto the diagonal to reduce the number of iterations needed. The most commonly used heuristic for performing reordering is the minimum degree algorithm [4], [5], An alternative approach, nested dissection ordering [6], has many appealing theoretical properties, but building an implementation that gives comparable ordering qualities and run times to the minimum degree method has been proven to be very difficult. Some promising results have been demonstrated e.g [7], [8].

None of these new methods has produced consistently better ordering than the minimum degree reordering method, and all require a significant amount of run time [9]. What is worse, even with these methods, the rank of the off-diagonal sub-matrices is not as small as expected. For simplicity, we assume that the rank of off-diagonal sub-matrices is less or equal to k, the upper bound of the computation complexity of the fast HSS matrix-vector multiplication is $O(nk^2)$ where n is the dimension of the system matrix. For the direct solving algorithm with HSS representation, the computation complexity would be approximately of $O(nk^3)$.

Therefore, it meaningful to develop iterative HSS solution methods where the approximate solution is improved iteratively. For each iteration, matrix-vector multiplication is the crucial operation. If the number of iterations needed is sufficiently small, the iterative solution methods will take less time than the direct HSS solution method does. Another benefit is that, in our application, we will have to solve a system of Maxwell equations in time domain, where the solution of a time instance does not differ too much from the solution of the previous time instance. This indicates that the solution of the previous time instance can be used as a good initial guess for the solution of the current time instance, the number of iterations can thus be reduced. In practice, it is not necessary to get a very accurate result. In our case, a few percent of error is tolerable. This helps us to further reduce the number of iterations needed. To summarize, all these above give us a good motivation to study iterative solution methods based on the HSS representation.

II. Hierarchically Semi-Separable Representation

The Hierarchical Semi-Separable representation (pioneered by Chandrasekaran and Gu [1]) of a matrix (or operator) A is a layered representation of the multi-resolution type, indexed by the hierarchical level. At the top level 1, it is a 2×2 block matrix representation of the form:

$$A = \begin{bmatrix} A_{1;1,1} & A_{1;1,2} \\ A_{1;2,1} & A_{1;2,2} \end{bmatrix} \tag{1}$$

in which we assume that the ranks of the off-diagonal blocks are low so that they can be represented by an 'economical' factorization ('H' indicates Hermitian transposition, for real matrices just transposition), as follows:

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

The second hierarchical level is based on a further but similar decomposition of the diagonal blocks, respectively $D_{1;1}$ and $D_{1;2}$:

$$D_{1;1} = \begin{bmatrix} D_{2:1} & U_{2;1}B_{2;1,2}V_{2;2}^H \\ U_{2;2}B_{2;2,1}V_{2;1}^H & D_{2;2} \end{bmatrix} \quad (3)$$

$$D_{1;2} = \begin{bmatrix} D_{2;3} & U_{2;3}B_{2;3,4}V_{2;4}^H \\ U_{2;4}B_{2;4,3}V_{2;3}^H & D_{2;4} \end{bmatrix} \quad (4)$$

for which we have the further *level compatibility* assumption

$$\text{span}(U_{1;1}) \subset \text{span}\left(\begin{bmatrix} U_{2;1} \\ 0 \end{bmatrix}\right) \oplus \text{span}\left(\begin{bmatrix} 0 \\ U_{2;2} \end{bmatrix}\right) \quad (5)$$

$$\text{span}(V_{1;1}) \subset \text{span}\left(\begin{bmatrix} V_{2;1} \\ 0 \end{bmatrix}\right) \oplus \text{span}\left(\begin{bmatrix} 0 \\ V_{2;2} \end{bmatrix}\right) \quad (6)$$

This spanning property is characteristic for the HSS structure, it allows for a substantial improvement on the numerical complexity for e.g. vector-matrix multiplication as a multiplication with the higher level structures always can be done using lower level operations, using the translation operators R and W

$$U_{k-1;i} = \begin{bmatrix} U_{k;2i-1}R_{k;2i-1} \\ U_{k;2i}R_{k;2i} \end{bmatrix}, \quad i = 1, 2...2^{k-1} \quad (7)$$

$$V_{k-1;i} = \begin{bmatrix} V_{k;2i-1}W_{k;2i-1} \\ V_{k;2i}W_{k;2i} \end{bmatrix}, \quad i = 1, 2...2^{k-1}. \quad (8)$$

Notice the use of indexes: at a given level i rows respectively columns are subdivided in blocks indexed by $1, \cdots, i$. Hence the ordered index $(i; k, \ell)$ indicates a block at level i in the position (k, ℓ) in the original matrix. The same kind of subdivision can be used for column vectors, row vectors and bases thereof (as are generally represented in the matrices U and V).

In [10] it is shown how this multilevel structure leads to efficient matrix-vector multiplication and a set of equations that can be solved efficiently as well. For the sake of completeness we review this result briefly. Let us assume that we want to solve the system $Tx = b$ and that T has an HSS representation with deepest hierarchical level K. We begin by accounting for the matrix-vector multiplication Tx. The dataflow diagram for a two level hierarchy representing operator-vector multiplication is shown in Figure 1. At the leaf node $(K; i)$ we can compute

$$g_{K;i} = V_{K;i}^H x_{K;i} \quad (9)$$

If $(k; i)$ is not a leaf node, we can infer, using the hierarchical relation

$$g_{k;i} = V_{k;i}^H x_{k;i} = W_{k+1;2i-1}^H g_{k+1;2i-1} + W_{k+1;2i}^H g_{k+1;2i} \quad (10)$$

These operations update a 'hierarchical state' $g_{k;i}$ upward in the tree. To compute the result of the multiplication, a new collection of state variables $\{f_{k;i}\}$ is introduced for which it holds that

$$b_{k;i} = T_{k;i,i} + U_{k;i}f_{k;i} \quad (11)$$

and which can also be computed recursively downward by the equations

$$\begin{bmatrix} f_{k+1;2i-1} \\ f_{k+1;2i} \end{bmatrix} = \begin{bmatrix} B_{k+1;2i-1,2i}g_{k+1;2i} + R_{k+1;2i-1}f_{k,i} \\ B_{k+1;2i,2i-1}g_{k+1;2i-1} + R_{k+1;2i}f_{k;i} \end{bmatrix} \quad (12)$$

the starting point being $f_{0;} = []$, an empty matrix. At the leaf level we can now compute (at least in principle - as we do not know x) the outputs from

$$b_{K;i} = D_{K;i}x_{K;i} + U_{K;i}f_{K;i} \quad (13)$$

Solution can also be done in time linear with the dimension of T, but the algorithms are a lot more complicated than fast matrix-vector multiplication, see [1], [3]. Almost all matrix operations have their HSS version. A relatively complete set of Hierarchically Semi-separable Algorithms is given in [2].

III. Hierarchically Semi-Separable representation construction from sparse matrices

The construction of HSS representations has been discussed in [11], in which $O(n^2)$ construction algorithms have been given. In this section we describe a $O(n)$ HSS representation construction algorithm for sparse matrices. We will show that the construction algorithm needs almost no computation, but matrix search operation is needed to find empty rows and columns in the sparse matrix.

The construction algorithm is a downsweep algorithm. Once the root is constructed, its leaves can be split recursively until a certain criterion is satisfied. In order to develop the top-down HSS construction algorithm for sparse matrices, we need the leaf-split algorithm presented in [1], [11]. Two other lemmas will help us with the HSS construction from sparse matrices.

A. Leaf-split

One big leaf can be split into two smaller leaves, and the input-output relation is not changed. Leaf split modification can be done on any leaf. Its dataflow modification and the dataflow notations have been studied in [12]. The formula to generate the matrices of the new leaves from those of the large one is given below:

1) Partitioning: Firstly, we partition the translation matrices in the following way. Note that the dimensions of the partitions should be appropriate, so that the following calculations below are legal. Let:

$$U_{k-1;i} = \begin{bmatrix} U_{k-1;i;(1)} \\ U_{k-1;i;(2)} \end{bmatrix}, V_{k-1;i} = \begin{bmatrix} V_{k-1;i;(1)} \\ V_{k-1;i;(2)} \end{bmatrix} \quad (14)$$

$$b_{k-1;i} = \begin{bmatrix} b_{k;2i-1} \\ b_{k;2i} \end{bmatrix}, x_{k-1;i} = \begin{bmatrix} x_{k;2i-1} \\ x_{k;2i} \end{bmatrix} \quad (15)$$

$$D_{k-1;i} = \begin{bmatrix} D_{k;2i-1} & D_{k-1;i;(12)} \\ D_{k-1;i;(21)} & D_{k;2i} \end{bmatrix} \quad (16)$$

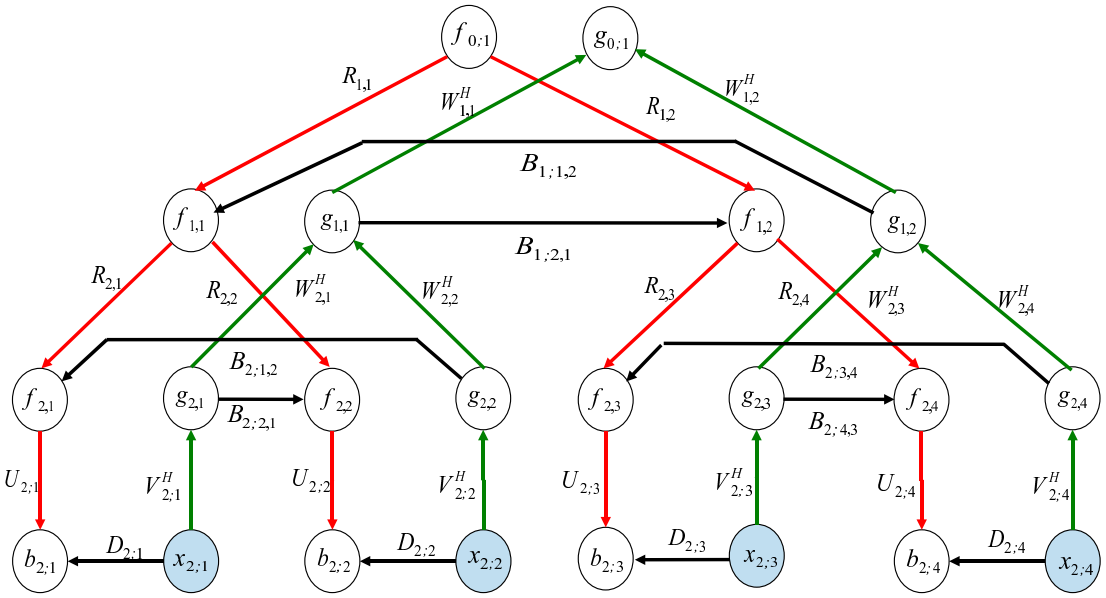

Fig. 1. HSS Data-flow diagram; arrows indicate matrix-vector multiplication of sub-data, nodes correspond to states and are summing incoming data (the top levels f_0 and g_0 are empty).

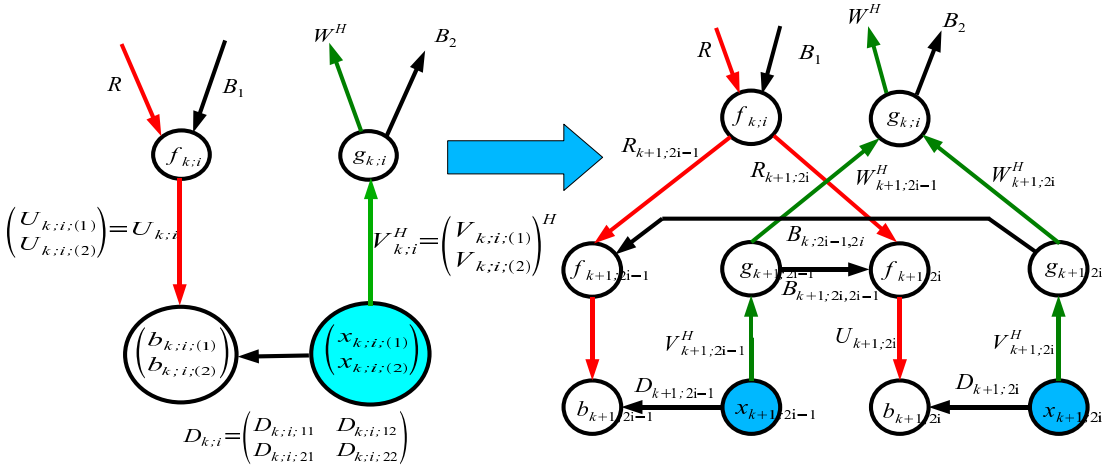

Fig. 2. Leaf split dataflow diagram: the leaf split algorithm takes one leaf and returns a node that keeps the same input-output relation

2) Factorization: Then, the matrices in the right hand of Figure 2 should hold the following equations (see Lemma 1):

$$\begin{bmatrix} U_{k-1;i;(1)} & D_{k-1;i;(12)} \\ 0 & V^H_{k-1;i;(2)} \end{bmatrix} = \begin{bmatrix} U_{k;2i-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} R_{k;2i-1} & B_{k;2i-1,2i} \\ 0 & W^H_{k;2i} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & V^H_{k;2i} \end{bmatrix} \tag{17}$$

$$\begin{bmatrix} V^H_{k-1;i;(1)} & 0 \\ D_{k-1;i;(21)} & U_{k-1;i;(2)} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & U_{k;2i} \end{bmatrix} \begin{bmatrix} W^H_{k;2i-1} & 0 \\ B_{k;2i,2i-1} & R_{k;2i} \end{bmatrix} \begin{bmatrix} V^H_{k;2i-1} & 0 \\ 0 & I \end{bmatrix} \tag{18}$$

Note that: U, V should be chosen such that they are column orthonormal. $\begin{bmatrix} R_{k;2i-1} \\ R_{k;2i} \end{bmatrix}$ and $\begin{bmatrix} W_{k;2i-1} \\ W_{k;2i} \end{bmatrix}$ should also be column orthonormal. In this way, we guarantee that the error is not amplified through propagation, we call a HSS representation with these properties a HSS representation in the proper form. Note that: In the proper form, the matrices B do not have to be column orthonormal, but they should be of low rank such that the HSS representation is efficient.

Lemma 1: Assuming A_{11} and A_{12} has dependent rows, it is possible (nontrivially) to factor a block upper triangular matrix into the form:

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} = \begin{bmatrix} Q_1 & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & Q_2 \end{bmatrix} \tag{19}$$

Proof: First consider a rank revealing factorization of the top row:

$$\begin{bmatrix} A_{11} & A_{12} \end{bmatrix} = Q_1 \begin{bmatrix} R_{11} & R_{12} \end{bmatrix} \tag{20}$$

followed by an rank revealing factorization of

$$\begin{bmatrix} R_{12} \\ A_{22} \end{bmatrix} = \begin{bmatrix} L_{12} \\ L_{22} \end{bmatrix} Q_2 \tag{21}$$

Then it follows that:

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix} = \begin{bmatrix} Q_1 & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} R_{11} & L_{12} \\ 0 & L_{22} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & Q_2 \end{bmatrix} \tag{22}$$

and our result follows with $B_{11} = R_{11}, B_{12} = L_{12}, B_{22} = L_{22}$. ■

Lemma 2: A sparse matrix A with empty rows can be trivially decomposed into Q and $\bar{A}$ where:

$$A = Q\bar{A} \tag{23}$$

Q is column orthonormal, and $\bar{A}$ collects all non-empty rows in A. Suppose A has n non-empty rows. Q is an incidence matrix.
Example:

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} = Q\bar{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

Lemma 3: A sparse matrix A with empty columns can be trivially decomposed into $\bar{A}$ and Q where:

$$A = \bar{A}Q^H \tag{24}$$

Q is column orthonormal, and $\bar{A}$ collects all non-empty columns in A. Suppose A has n non-empty columns. Q is an incidence matrix.
Example:

$$\begin{bmatrix} 1 & 2 & 3 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} = \bar{A}Q^H = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 2 & 1 \\ 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}^H$$

3) Formal algorithm: With the leaf-split method at our disposal, we can then replace the rank revealing factorizations in Lemma 1 with the sub-optimal factorizations in Lemma 2 and 3. The pseudo-code of this top-down method follows:

Downsweep: Input a leaf Leaf, a certain condition d to terminate the construction process; output a tree that is constructed from the leaf.

1) *if ($d_{k;i}$ = false) then (node,leftleaf,rightleaf) = leaf-split(Leaf);*
   return
   Node(node, Downsweep(leftleaf, $d_{k+1;2i-1}$), Downsweep(rightleaf,$d_{k+1;2i}$));
2) *else return Leaf;*

Main: Input a plain matrix A. Output the HSS representation of A.

1) *Build a dummy leaf L with A, of which $U = |$, $V = |$, $D = A$;*
2) *HSSrepresentation = Downsweep(L, $d_{0,1}$);*

This construction algorithm needs not computation. All splits can be done with sparse matrix arithmetics; therefore, it is efficient in terms of memory usage. With the data structure that enables fast matrix search for empty rows and columns, the algorithm is also quite fast. However, the HSS representation constructed with this algorithm is not optimal. It produces a sub-optimal HSS representation in return some shorter computational time.

IV. Design of the HSS iterative solver

Practical iterative solvers consist of standard iterative solution methods (CG, CGS, GMRES, etc), appropriate preconditioners, efficient matrix-vector multiplication methods, and accurate convergence estimation. We have implemented some iterative algorithms with OCAML [13] and camlfloat [14]. For the Krylov space iterative solvers, we have implemented solvers like CG, CGS, BiCG, Bi-CGSTAB and so on (all based on the HSS representation). With all the algorithms under the HSS framework, it is quite easy to combine the HSS representation and its fast algorithms with any iterative solution methods.

V. Preconditioners

As well studied by other researchers, the convergence rate of various iterative methods depends on spectral properties of the coefficient matrix. Thus, the system matrix can be transformed into an equivalent one in the sense that it has the same solution, but has more favorable spectral properties. A preconditioner is the matrix that effects such transformation[15].

A preconditioner is in fact an approximation to the original system matrix A; In order to archive any speedup, this preconditioner should be easy to compute and the inverse of this approximation matrix should be easy to apply on any vector. For the solution problem ($Ax = b$, knowing A, b, compute x), suppose the left preconditioner M approximates A in some way, the transformed system would be:

$$M^{-1}Ax = M^{-1}b \tag{25}$$

In this section, we shall describe a few preconditioners, for which the OCAML implementations of their construction algorithms and solution algorithms are available, to accelerate the convergence rate.

A. Block diagonal preconditioner

Given the solution problem ($Ax = b$), with the assumption that A is given in its HSS representation. The simplest preconditioner M consists of just the diagonal blocks of the HSS matrix A.

$$M = \mathbf{D} \tag{26}$$

where $\mathbf{D}$ collects only the on-diagonal sub-matrices($D_{k;i}$) of the HSS representation of the matrix A. This is also known as the block Jacobi preconditioner. The inverse of this block diagonal matrix M can be computed by inverting the matrix block-wise.

B. Symmetric Successive Overrelaxation preconditioniner

Another 'cheap' preconditioner is the SSOR preconditioner. Like the Block Jacobi preconditioner, this preconditioner can be derived without any work and additional storage.

Suppose the original system A is symmetric, we shall decompose A as

$$A = D + L + L^T \tag{27}$$

where L is a block lower triangular HSS matrix and D is a block diagonal matrix. The SSOR matrix is defined as

$$M = (D + L)D^{-1}(D + L)^T \tag{28}$$

usually, M is parameterized by ω as follows:

$$M(\omega) = \frac{1}{(2-\omega)}(\frac{1}{\omega}D + L)(\frac{1}{\omega}D)^{-1}(\frac{1}{\omega}D + L)^T \tag{29}$$

The optimal value of ω will reduce the number of iteration needed significantly. However, computing the value of the optimal ω needs the spectral information which is normally not available in advance and prohibitively expensive to compute. The direct solution method of such block triangular HSS system($\frac{1}{\omega}D + L$) has been discussed in [2], [11].

C. Fast model reduced preconditioner

A downsweep model reduction can be done on the HSS representation to reduce its HSS complexity at the cost of loss in data. Here, we only review the algorithm, for details on proof and analysis, refer to [12].

Suppose A is a HSS matrix of which the HSS representation is defined by sequences U, V, R, W, B, D. The downsweep model reduction algorithm consists of two possible operations:

1) Reduction at node/leaf: When needed, model reduction could be done on nodes. Given a node like the one shown on the left of Figure 3, with the tolerance specified, we can decompose the translation matrices with economical rank revealing factorization as follows:

$$\begin{bmatrix} R_{k;2i-1} & B_{k;2i-1,2i} \end{bmatrix} = \bar{U}_{k;2i-1} \begin{bmatrix} \bar{R}_{k;2i-1} & \hat{B}_{k;2i-1,2i} \end{bmatrix} + O(\epsilon) \tag{30}$$

$$\begin{bmatrix} R_{k;2i} & B_{k;2i,2i-1} \end{bmatrix} = \bar{U}_{k;2i} \begin{bmatrix} \bar{R}_{k;2i} & \hat{B}_{k;2i,2i-1} \end{bmatrix} + O(\epsilon) \tag{31}$$

$$\begin{bmatrix} W^H_{k;2i} \\ \hat{B}_{k;2i,2i-1} \end{bmatrix} = \begin{bmatrix} \bar{W}^H_{k;2i} \\ \bar{B}_{k;2i-1,2i} \end{bmatrix} \bar{V}^H_{k;2i} + O(\epsilon') \tag{32}$$

$$\begin{bmatrix} W^H_{k;2i-1} \\ \hat{B}_{k;2i,2i-1} \end{bmatrix} = \begin{bmatrix} \bar{W}^H_{k;2i-1} \\ \bar{B}_{k;2i-1,2i} \end{bmatrix} \bar{V}^H_{k;2i-1} + O(\epsilon') \tag{33}$$

Or equivalently:

$$\begin{bmatrix} R_{k;2i-1} & B_{k;2i-1,2i} \\ 0 & W^H_{k;2i} \end{bmatrix}$$

$$= \begin{bmatrix} \bar{U}_{k;2i-1} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \bar{R}_{k;2i-1} & \bar{B}_{k;2i-1,2i} \\ 0 & \bar{W}^H_{k;2i} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \bar{V}^H_{k;2i} \end{bmatrix} + O(\epsilon') \tag{34}$$

$$\begin{bmatrix} R_{k;2i} & B_{k;2i,2i-1} \\ 0 & W^H_{k;2i-1} \end{bmatrix}$$

$$= \begin{bmatrix} \bar{U}_{k;2i} & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \bar{R}_{k;2i} & \bar{B}_{k;2i,2i-1} \\ 0 & \bar{W}^H_{k;2i-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \bar{V}^H_{k;2i-1} \end{bmatrix} + O(\epsilon') \tag{35}$$

Thus the translation matrices of this node have been reduced as: $\bar{R}_{k;2i-1}$, $\bar{R}_{k;2i}$, $\bar{W}_{k;2i-1}$, $\bar{W}_{k;2i}$, $\bar{B}_{k;2i-1,2i}$ and $\bar{B}_{k;2i,2i-1}$. The factors $\bar{U}_{k;2i-1}$, $\bar{U}_{k;2i}$, $\bar{V}_{k;2i-1}$ and $\bar{V}_{k;2i}$ will be propagated to its children and modified their translation matrices.

2) Downsweep modification: After these factors $\bar{U}_{k;i}$, $\bar{V}_{k;i}$ of a node are computed, they will be sweep to the children of this node, and modify their translation matrices as in Figure 3.

1) If the child is a non-leaf node

$$\bar{R}_{k;2i-1} = R_{k;2i-1}\bar{U}_{k;i}, \bar{R}_{k;2i} = R_{k;2i}\bar{U}_{k;i} \tag{36}$$

$$\bar{W}_{k;2i-1} = W_{k;2i-1}\bar{V}_{k;i}, \bar{W}_{k;2i} = W_{k;2i}\bar{V}_{k;i} \tag{37}$$

2) If the child is a leaf

$$\hat{U}_{k;i} = U_{k;i}\bar{U}_{k;i}, \hat{V}_{k;i} = V_{k;i}\bar{V}_{k;i} \tag{38}$$

After this modification, reduction method can be done on this modified node to reduce its complexity. When the downsweep recursion reaches the leaves of the HSS representation, the whole HSS representation has been model reduced under a certain tolerance.

D. Fast model reduction with HSS LU factorization preconditioner

It is known that the standard CG method only works for symmetric positive definite matrices. For the matrices that are not symmetric positive definite, the standard CG method would converge quite slowly or not at all. We will of course expect the transformed system to be symmetric positive definite, if the original system is so.

The left preconditioner alone is often not what is used in practice; because the transformed matrix $M^{-1}A$ is generally not symmetric, even though A and M are symmetric. Therefore, the standard CG method is not immediately applicable to this system. We can of course use the CGS and the BICG method which can handle nonsymmetric positive definite systems; however, it is advantageous to use the standard CG method due to its simplicity and low computational cost in each iteration.

One way to remedy the preconditioner for the standard CG method is to LU factorize the left preconditioner as $M = M_1M_2$, and apply M_1 and M_2 separately as the left preconditioner and the right preconditioner. Then the original system would be transformed into the following:

$$M_1^{-1}AM_2^{-1}(M_2x) = M_1^{-1}b \tag{39}$$

Here M_1 is called the left preconditioner; M_2 is called the right precondtioner. If M is symmetric, that is $M_1 = M_2^T$ (note that if the original HSS matrix is symmetric, the preconditioner constructed by the algorithm presented in section V-C is symmetric as well), one can easily prove that the transformed coefficient matrix $M_1^{-1}AM_2^{-1}$ is symmetric. Thus the standard CG method is applicable again. M_1 and M_2 can be constructed by a LU factorization (details with proof in [2]) on the HSS matrix M.

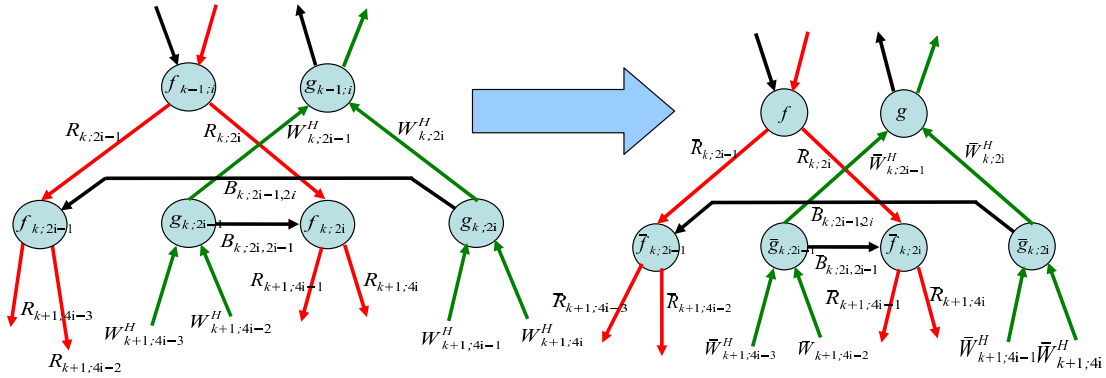

Fig. 3. Fast model reduction on nodes. It reduces the HSS complexity of a node at the cost of loss in data

E. Preconditioners summary

Summarizing Table I compares the preconditioners and their solution methods we proposed, including effort and storage needed to construct these preconditioners.

VI. Numerical result

To study the behavior of the iterative HSS solver we developed, we choose to experiment with the HSS CG, HSS CGS, HSS BiCG, HSS Bi-CGSTAB method and the direct HSS solver on smooth matrices A defined as:

$$A_{ij} = \begin{cases} c \times n & i = j \\ \frac{n}{|i-j|} & i \neq j \end{cases} \tag{40}$$

(n is the dimension of the matrix, c is a parameter to control the diagonal dominance. We choose the value of c to be 2, so that the matrix is positive definite.) The required solution accuracy of iterative solver is specified to be 10^{-6}; the initial guess for the solution is given as vector of zeros; the right hand side is a random vector. The goal is to compute x so that $Ax = b$.

It can be seen that from Figure 4 that the CPU time needed by the HSS CG method is comparable with that of the HSS direct solver. Among the CG like methods, the standard CG method takes the least time, however, its applicability is not as good as that of the others. The HSS CGS method takes about half of the time needed by the HSS BiCG method; however, it is worth mentioning that the behavior of the CGS method is highly irregular. It may even fail to deliver a solution when other CG variants do (the diverging cases are not plotted in Figure 4). Bi-CGSTAB is more stable than CGS and it does not require the matrix transpose. These are consistent with the analysis of these CG variants in [15]. It can also be seen from Figure 4 that the time curve of iterative methods are irregular in general, while the direct solver scales well with the size of the matrices (if the system matrices are smooth on off-diagonal sub-matrices).

One question still remains: under what situation should the iterative methods be preferred over the direct solver? As we mentioned, the core operation of iterative methods is matrix-vector multiplication; this operation scales better with the HSS complexity than the direct HSS solution method does. This indicates that the iterative methods should be adapted under

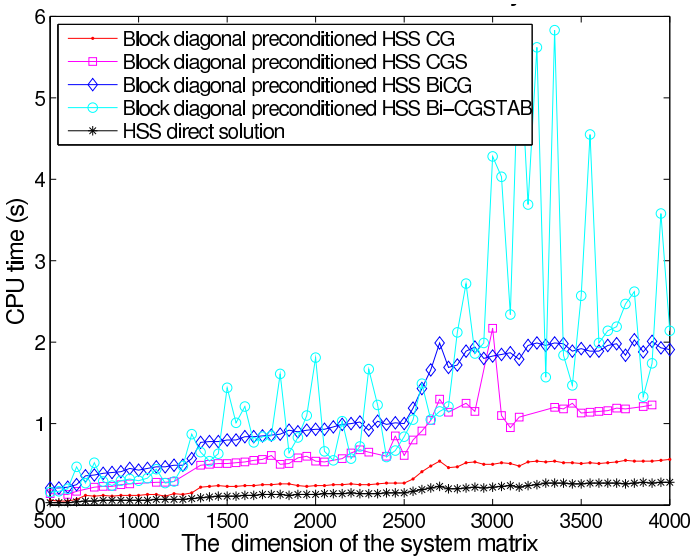

Fig. 4. Numerical experiment with solvers: CPU time needed to solve system matrices of different sizes with different solution methods

the circumstance that the off-diagonal sub-matrices of the HSS matrices is not of low rank. We shall do a series of experiments to see how the iterative methods and direct method would scale with the smoothness.

We choose to work on the smooth matrix A defined as:

$$A_{ij} = \begin{cases} c \times n & i = j \\ n \times \cos(k|i-j|\pi) & i \neq j \end{cases} \tag{41}$$

Here, k is used to control the smoothness, a larger k would result in more high frequency components, which would then result in less smooth matrices and increase the HSS complexity of the HSS representation. n is the dimension of the matrix. n here is specified as 2000; that is the matrices are of size 2000×2000. c controls diagonal dominance; we choose the value of c be 2. A series of experiments with different k is performed on the HSS CG method, the HSS direct solution method and the direct solution method from LAPACK.

From Figure 5, we can see that the solution methods based on the HSS representation are preferred when the system matrix is non-smooth; it is obvious that the direct solution method does not scale well with the increasing value of k, while the CPU time needed by HSS CG method increases

TABLE I
HSS PRECONDITIONERS: CONSTRUCTION AND SOLUTION

Preconditioner	**Construction**	**Storage needed**	**Inverse solution**	**Remarks**
Block Jacobi	without effort	not needed	direct inverse	Only suitable for diagonal dominant matrix
Block SSOR	without effort	not needed	HSS forward and backward substitution [2], [11]	simple double sided preconditioner
Fast model reduction	Model reduction [12]	needed	Fast HSS direct Solvers [1], [2], [3]	advanced, high cost
Fast model reduction with LU factorization	Model reduction and HSS LU [12], [2]	needed	HSS forward and backward substitution [2], [11]	advanced double sided preconditioner, high cost

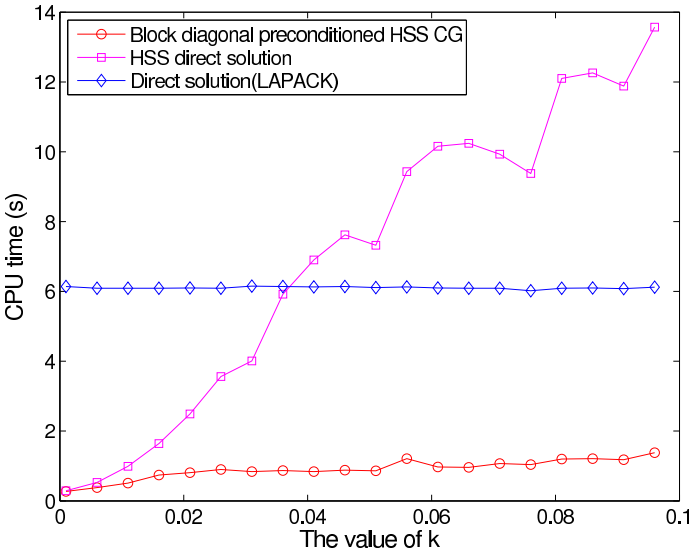

Fig. 5. Numerical experiment with solvers on 2000×2000 system matrices: the CPU time needed to solve system matrices of fixed dimension with different smoothness

smoothly with the value of k.

After the above comparison, it is safe to conclude that HSS iterative method should be preferred over direct HSS solution method, if the HSS complexity of the HSS representation is not small compared to the dimension of the matrix. Or equivalently, the iterative method should be preferred when the matrix is not very smooth. However if the matrix is completely not smooth, the solution methods based on HSS representation described in this paper are not recommended.

VII. CONCLUSIONS

We studied the limitation of direct HSS solution method. A general strategy to combine the HSS representation and its algorithms with iterative solution algorithms has been given. With this strategy, any iterative algorithm can be easily combined with the HSS representations. We implemented and tested a number of iterative solution algorithms based on HSS representations. All these numerical experiments suggest that when the off-diagonal blocks of the system matrix are not so smooth, the iterative algorithms based HSS representations exceed its direct counterpart in CPU time and memory usage. We also proposed and implemented a number of preconditioners based on HSS representation to improve the convergence of the iterative methods.

REFERENCES

[1] S. Chandrasekaran, M. Gu, and W. Lyons, "A fast and stable adaptive solver for hierachically semi-separable representations," *unpublished document*, April 2004.
[2] Z. Sheng, P. Dewilde, and S. Chandrasekaran, "Algorithms to solve hierarchically semi-separable systems," *Operator Theory: Advances and Applications*, 2006, accepted.
[3] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, "A fast solver for hss representation via sparse matrices," August 2005.
[4] A. George and J. W. H. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Rev.*, vol. 31, pp. 1–19, 1989.
[5] W. F. Tinney and J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," *J. Proc. IEEE*, vol. 55, pp. 1801–1809, April 1967.
[6] A. George, "Nested dissection of a regular finite element mesh," *SIAM J. Numer. Anal*, vol. 10, pp. 345–363, 1973.
[7] T. Bui and C. Jones, "A heuristic for reducing fill in sparse matrix factorization," *in Proc.Parallel Processing for Scientific Computing, SIAM, Philadelphia*, pp. 445–452, 1993.
[8] C. Ashcraft and J. W. H. Liu, "A partition improvement algorithm for generalized nested dissection," *Tech. Rep. BCSTECH-94-020, Boeing Computer Services, Seattle, WA*, 1994.
[9] B. Hendrickson and E. Rothberg, "Improving the run time and quality of nested dissection ordering," *SIAM J. SCI. COMPUT.*, vol. 20, pp. 468–489, 1998.
[10] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals, "A fast solver for hss representations via sparse matrices," in *Technical Report*. Delft University of Technology, August 2005.
[11] W. Lyons, "Fast algorithms with applications to pdes," *PhD thesis*, June 2005.
[12] Z. Sheng, "Hierarchically semi-separable representation and its applications," *Master thesis*, 2006.
[13] J. Hickey, *Introduction to the Objective Caml Programming Language*, October 2005.
[14] W. Lyons and S. Chandrasekaran, *Camlfloat Tutorial*, 2004.
[15] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the solution of Linear Systems: Building Blocks for Iterative Methods*. Philadelphina, PA: SIAM, 1994.