Computing 80, 169–188 (2007)
DOI 10.1007/s00607-007-0224-4
Printed in The Netherlands

**Computing**

# An algebraic approach for $\mathcal{H}$-matrix preconditioners

**S. Oliveira**, and **F. Yang**, Iowa City

Received February 14, 2006; revised December 28, 2006
Published online: April 10, 2007
© Springer-Verlag 2007

**Abstract**

Hierarchical matrices ($\mathcal{H}$-matrices) approximate matrices in a data-sparse way, and the approximate arithmetic for $\mathcal{H}$-matrices is almost optimal. In this paper we present an algebraic approach for constructing $\mathcal{H}$-matrices which combines multilevel clustering methods with $\mathcal{H}$-matrix arithmetic to compute the $\mathcal{H}$-inverse, $\mathcal{H}$-LU, and the $\mathcal{H}$-Cholesky factors of a matrix. Then the $\mathcal{H}$-inverse, $\mathcal{H}$-LU or $\mathcal{H}$-Cholesky factors can be used as preconditioners in iterative methods to solve systems of linear equations. The numerical results show that this method is efficient and greatly speeds up convergence compared to other approaches, such as JOR or AMG, for solving some large, sparse linear systems, and is comparable to other $\mathcal{H}$-matrix constructions based on Nested Dissection.

*AMS Subject Classifications:* 65F10.

*Keywords:* Preconditioners, multilevel methods, hierarchical matrices.

## 1. Introduction

The basic concept of hierarchical-matrices ($\mathcal{H}$-matrices) was introduced by Hackbusch [4]. Following this, much work has been done on the theory and applications of $\mathcal{H}$-matrices [2], [1], [6], [7], [3]. The basic idea of $\mathcal{H}$-matrix representation is to use a block cluster tree $T_{I\times I}$ to store a multilevel block partitioning of a matrix. The nodes of $T_{I\times I}$ represent Cartesian products of the corresponding row and column index sets of the matrix blocks. The leaves of $T_{I\times I}$ represent the smallest blocks that are not partitioned further. These blocks are either rank $k$ approximations or full matrices. $\mathcal{H}$-matrices are very suitable for describing certain sparse matrices arising from partial differential equations or full matrices arising from integral operators. The advantage of $\mathcal{H}$-matrix representation is that it can reduce the required storage, as well as the computational complexity of approximate $\mathcal{H}$-matrix operations, such as matrix-vector multiplication, matrix-matrix multiplication, matrix addition and inversion, to almost linear complexity ($O(n \log^{\alpha} n)$) [2], [3].

The classic methods to construct $\mathcal{H}$-matrices rely on the geometric information related to the problems, and approximation is used to represent the blocks by rank $k$ matrices [3]. A new method to build $\mathcal{H}$-matrices is proposed in [7] which uses the graph of the matrix and Nested Dissection. Its advantage is that the $\mathcal{H}$-matrices constructed in this way are very suitable for $\mathcal{H}$-LU decomposition, which keeps large off-diagonal blocks zero.

We present an algebraic method to build $\mathcal{H}$-matrices, which uses the information of a matrix itself and multilevel clustering methods based on Heavy Edge Matching (HEM). In this way a sparse matrix can be reordered and represented exactly in the format of $\mathcal{H}$-matrices. We can compute not only the approximate $\mathcal{H}$-LU and $\mathcal{H}$-Cholesky factors but also $\mathcal{H}$-inverses by using the $\mathcal{H}$-matrix arithmetic. They can then be used as preconditioners in iterative methods, such as GMRES. The experimental results show that our method provides an effective method for $\mathcal{H}$ construction and is competitive with the method in [7].

This paper is organized as follows. In the next section, we first review multilevel clustering methods and then show the way to apply them to the index set partitioning. The process to construct an $\mathcal{H}$-matrix using multilevel clustering methods is discussed in Sect. 3. Section 4 is a brief introduction to $\mathcal{H}$-matrices and $\mathcal{H}$-matrix arithmetic. Finally in Sect. 5, we present the numerical results of applying these methods to several problems and compare them with other methods and preconditioners.

## 2. Multilevel clustering methods

To build a $\mathcal{H}$-matrix for a sparse matrix, we first need to build a cluster tree $T_I$ over the matrix index set $I$. $T_I$ describes the partitioning over $I$ from the finest to the coarsest level. The way to build $T_I$ is based on multilevel clustering methods which are widely used in graph partitioning.

The basic idea of multilevel clustering is: starting from the finest graph, build clusters over its nodes, then build a coarse graph by merging the nodes in the same cluster as one coarse node, and continue this coarsening process on the coarse graphs until the graph obtained is small enough.

### *2.1. Coarsening process*

The finest graph $G_0$ for the coarsening process is constructed from a matrix. Given a sparse matrix $M$, its corresponding weighted undirected graph $G_0 = (V(G_0), E(G_0))$ is defined as follows: $V(G_0)$, the set of nodes in $G_0$, corresponds to the matrix index set $I$; node $i \in V(G_0)$ represents an index $i \in I$; there is an edge $e_{ij} \in E(G_0)$ if and only if matrix entry $m_{ij} \neq 0$; and the edge weight $w_{ij} = |m_{ij}|$.

An algorithm based on Heavy Edge Matching (HEM) [5] is used to build clusters over the nodes in $G_i = (V(G_i), E(G_i))$ and construct a coarser graph $G_{i+1} = (V(G_{i+1}), E(G_{i+1}))$.

The basic idea of HEM is: randomly pick up an unmatched node; find another unmatched node such that these two nodes are connected by an edge with the heaviest weight and mark them as matched; repeat this process until all nodes are matched. Pseudocode is shown in Fig. 1. As we continue marking nodes as matched, the

```
match(s, V) /* find match for s over unmatched nodes V */
  if ( $w_{sp} = 0$ for all $p \in V$ )  then return $\{s\}$
  else $t \leftarrow \arg\max_{p \in V} w_{sp}$;  return $\{s, t\}$
end match

while $V \neq \emptyset$
  pick $s \in V$;  $C_k \leftarrow match(s, V)$
  $V \leftarrow V \backslash C_k$;  $k \leftarrow k + 1$
end while
```

**Fig. 1.** Basic heavy-edge matching algorithm

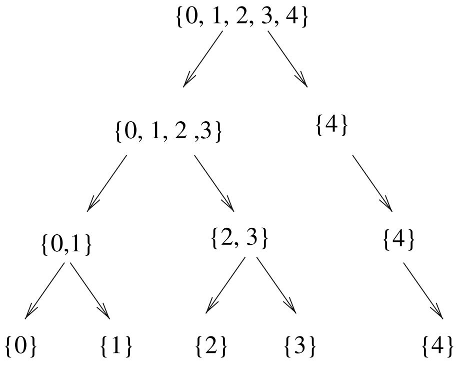


**Fig. 2.** An example of unbalanced cluster tree $T_I$

unmatched nodes will have higher chance to stay isolated, i.e., they have no unmatched neighbors. If a node or its corresponding cluster remains isolated as we continue the coarsening process, then the corresponding cluster tree $T_I$, as discussed in Sect. 2.2 may be unbalanced, as shown in Fig 2.

To make $T_I$ more balanced, we need a less random approach. Instead we divide the set of nodes $V$ into two groups $V_1$ and $V_2$. The nodes in $V_1$ have a higher priority than nodes in $V_2$. This modifies the matching algorithm in Fig. 1 to the algorithm in Fig. 3.

Priority goes to the nodes in a coarsened graph that consist of a single node at the previous (finer) level: thus if $\#C_k^{(i)} = 1$ then node $k$ is put into the higher priority group $V_1$ at level $i + 1$. This usually avoids the problem of isolated nodes causing unbalanced trees.

Once the clusters $C_k^{(i)}$ for level $i$ are computed, the weights for the next level $i + 1$ can be computed by the formula

$$w_{kl}^{(i+1)} = \sum_{r \in C_k^{(i)}} \sum_{s \in C_l^{(i)}} w_{rs}^{(i)}. \tag{1}$$

Recursively applying the above coarsening process gives a sequence of coarse graphs $G_1, G_2, \ldots, G_h$. We end this sequence with $G_h$, when $\#V(G_h)$ is sufficiently small.

```
$priority\_match(V_1, V_2)$ /* $V_1$ has priority over $V_2$ */
  $k \leftarrow 1$
  while $V_1 \neq \emptyset$
    pick $s \in V_1$;   $C_k \leftarrow match(s, V_1 \cup V_2)$
    $V_1 \leftarrow V_1 \backslash C_k$;   $V_2 \leftarrow V_2 \backslash C_k$;   $k \leftarrow k + 1$
  end while
  while $V_2 \neq \emptyset$
    pick $s \in V_2$;   $C_k \leftarrow match(s, V_2)$
    $V_2 \leftarrow V_2 \backslash C_k$;   $k \leftarrow k + 1$
  end while
  return $(C_1, C_2, \ldots, C_{k-1})$
end $priority\_match$
```

**Fig. 3.** Modified heavy-edge matching algorithm

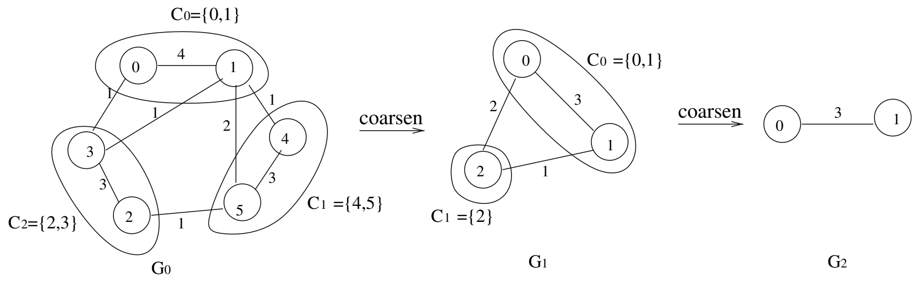


**Fig. 4.** An example of the multilevel graph coarsening process

Figure 4 illustrates the coarsening process with 2 levels for a graph defined by the following matrix:

$$\widetilde{M} = \begin{bmatrix} * & 4 & 0 & 1 & 0 & 0 \\ 4 & * & 0 & 1 & 1 & 2 \\ 0 & 0 & * & 3 & 0 & 1 \\ 1 & 1 & 3 & * & 0 & 0 \\ 0 & 1 & 0 & 0 & * & 3 \\ 0 & 2 & 1 & 0 & 3 & * \end{bmatrix}. \tag{2}$$

## *2.2. Cluster tree $T_I$ and matrix permutation*

The cluster tree $T_I$ describes the multilevel partitioning of the list of indices $I = (0, 1, 2, \ldots, n)$. Note that $(0, 1, 2) \neq (0, 2, 1)$; that is, order is significant. To build $T_I$ we first build a tree $\widetilde{T}_I$ based on the multilevel graphs $G_0, G_1, G_2, \ldots, G_h$. The nodes of $\widetilde{T}_I$ are the nodes of $G_i$, $i = 0, 1, \ldots, h$ together with a root node.

$\widetilde{T}_I$ is constructed by making the all the nodes $j \in V(G_h)$ children of the root, and make the node $k \in V(G_i)$ the parent of all the nodes $j \in C_k^{(i-1)} \subset V(G_{i-1})$. The leaves of $\widetilde{T}_I$ correspond to the indices of the original matrix $\widetilde{M}$. For the example in Fig. 4, $\widetilde{T}_I$ is shown in Fig. 5a.

We create another list $\widetilde{I}$ by listing the leaves of $\widetilde{T}_I$ from left to right. $\widetilde{I}$ and $I$ have the same index elements but their elements may be in different order because of the clustering process. So we map the indices of $\widetilde{I}$ to the indices in $I$, for our example $\widetilde{I} = (0, 1, 4, 5, 2, 3) \rightarrow I = (0, 1, 2, 3, 4, 5)$. The leaves of $\widetilde{T}_I$ are mapped to the new indices in the same way (see Fig. 5b). A permutation matrix $P$ is constructed based on this index mapping, and applied symmetrically to the original matrix $\widetilde{M}$: $M = P^T \widetilde{M} P$. $M$ is the reordered matrix.

Then we replace each node in $\widetilde{T}_I$ from the leaves to the root by a set of indices $L_j \subset I$: if a node $j$ is a leaf then $L_j = \{j\}$; else if a node $j$ of the level $i$ has children $t$ and $k$ at the level $i-1$ then $L_j^{(i)} = L_t^{(i-1)} \bigcup L_k^{(i-1)}$. This new tree is $T_I$ (shown in Fig. 5c).

$T_I$ satisfies the conditions of a cluster tree for a $\mathcal{H}$-matrix and has following properties:

(1) $T_I$ has a total of $h + 1$ levels, where $h$ is the number of levels in the HEM coarsening algorithm.
(2) The root of $T_I$ is set $I = \{0, 1, 2, \ldots, n\}$, whose elements are the indices of the permuted matrix $M$.
(3) The sets $L_k^{(i)}, k \in V(G_i)$ at each level of $T_I$ form a partition over the index set $I$.
(4) The elements inside $L_k^{(i)}$ are in increasing order.

Figure 5 shows $\widetilde{T}_I$ and $T_I$ built form the multilevel graphs in Fig. 4. The corresponding index mapping is shown in Fig. 6. The corresponding permuted matrix from Eq. (2), based on the index mapping of Fig. 6 is

$$M = \begin{bmatrix} * & 4 & 0 & 0 & 0 & 1 \\ 4 & * & 1 & 2 & 0 & 1 \\ 0 & 1 & * & 3 & 0 & 0 \\ 0 & 2 & 3 & * & 1 & 0 \\ 0 & 0 & 0 & 1 & * & 3 \\ 1 & 1 & 0 & 0 & 3 & * \end{bmatrix}. \tag{3}$$

## *2.3. Block cluster tree $T_{I \times I}$*

A block cluster tree $T_{I \times I}$ describes a multilevel block partitioning of the reordered matrix $M$. We use the cluster tree $T_I$ and the multilevel graphs $G_0, G_1, G_2, \ldots, G_h$ to build $T_{I \times I}$ from top to bottom. In the remainder of this paper, $\#A$ denotes the number of elements in the set $A$.

A constant $N_s$ is used to control the size of the smallest blocks in order to maintain the efficiency of the $\mathcal{H}$-matrix arithmetic. The basic steps for building $T_{I \times I}$ are as follows:

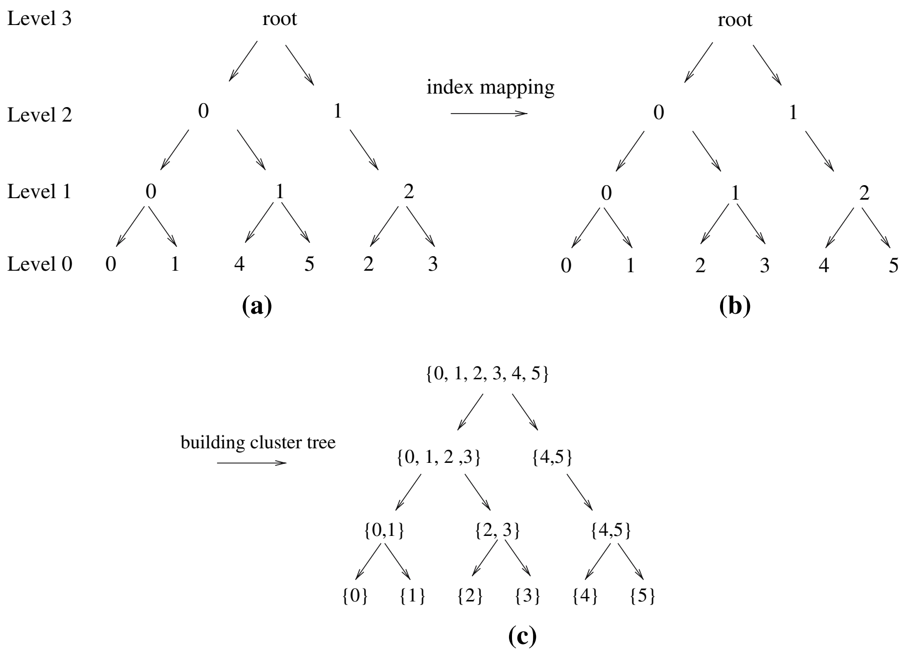


**Fig. 5.** Cluster trees for the graphs shown in Fig. 4. (**a**) $\widetilde{T}_I$ (before the index mapping), (**b**) $\widetilde{T}_I$ (after the index mapping), and (**c**) $T_I$

$$\begin{tikzcd}
\text{Old indexes} & \{0, \arrow[d] & 1, \arrow[d] & 4, \arrow[d] & 5, \arrow[d] & 2, \arrow[d] & 3\} \arrow[d] \\
\text{New indexes} & \{0, & 1, & 2, & 3, & 4, & 5\}
\end{tikzcd}$$

**Fig. 6.** Index mapping built for $T_I$

(1) The root of $T_{I\times I} = I \times I$ since $I$ is the root of $T_I$.
(2) If $L_r^{(i)} \times L_s^{(i)}$ is a node of $T_{I\times I}$, $r$ is connected to $s$ in $G_i$, and $\#L_r^{(i)}$, $\#L_s^{(i)} > N_s$, then the children of $L_r^{(i)} \times L_s^{(i)}$ are $L_v^{(i-1)} \times L_w^{(i-1)}$ where $L_v^{(i-1)}$ is a child of $L_r^{(i)}$, and $L_w^{(i-1)}$ is a child of $L_s^{(i)}$ in $T_I$.
(3) If $L_r^{(i)} \times L_s^{(i)}$ is a node of $T_{I\times I}$, $r$ is *not* connected to $s$ in $G_i$, and $\#L_r^{(i)}$, $\#L_s^{(i)} > N_s$, then $L_r^{(i)} \times L_s^{(i)}$ is a rank-$k$ leaf node in $T_{I\times I}$.
(4) If $L_r^{(i)} \times L_s^{(i)}$ is a node of $T_{I\times I}$, and $\#L_r^{(i)} \leq N_s$ or $\#L_s^{(i)} \leq N_s$, then $L_r^{(i)} \times L_s^{(i)}$ is a dense leaf node in $T_{I\times I}$.

This process is illustrated in Fig. 9, which is based on the multilevel graphs in Fig. 7 and the cluster tree $T_I$ in Fig. 8. Notice that in our example we used $N_s = 1$ for the minimal block size. However the process is similar for other values of $N_s$ except that we stop sooner (higher) in the tree.

The difference between our method and the classic methods for building $T_{I\times I}$ [2, 1, 3] is that they use geometric conditions (admissibility conditions) to determine whether or not a block $L_i^{(t)} \times L_j^{(t)}$ in $T_{I\times I}$ will be partitioned further. If a block

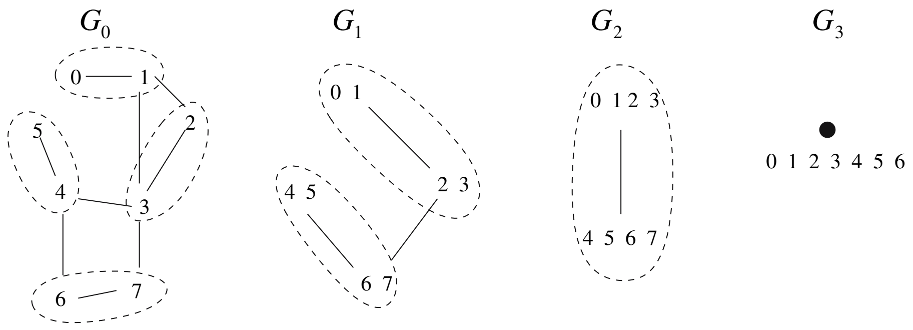


**Fig. 7.** Graphs for constructing the block cluster tree

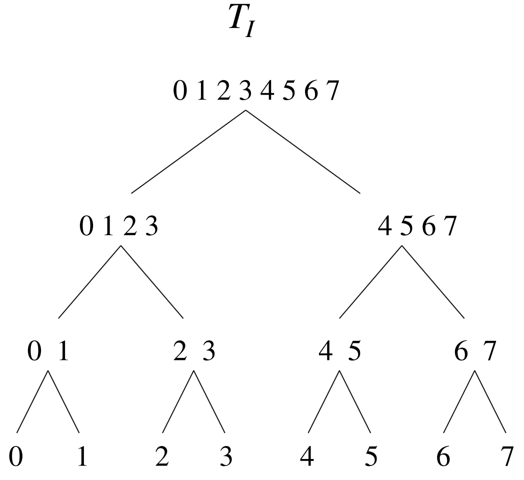


**Fig. 8.** The cluster tree $T_I$

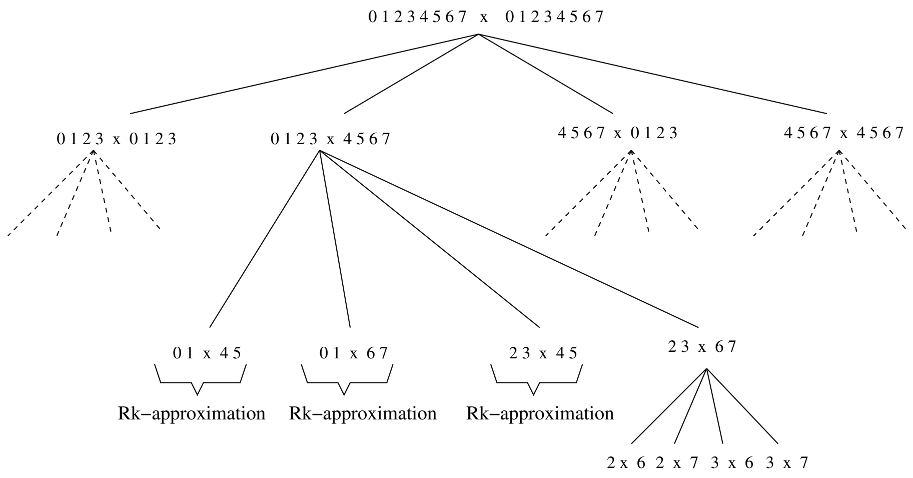


**Fig. 9.** The block cluster tree $T_{I\times I}$

is admissible, then it can be approximated by an Rk-matrix. Each internal node has exactly two children. In our algebraic method we use the edge weights from the coarse graphs to decide how to build $T_{I\times I}$. Particularly, if $w_{kt}^{(i)} = 0$ the block $L_k^{(i)} \times L_t^{(i)}$ is represented in the format of an Rk-matrix; $w_{kt}^{(i)} \neq 0$ implies that the block $L_k^{(i)} \times L_t^{(i)}$ is partitioned into smaller blocks at the next level. In this way the original sparse matrix can be expressed exactly as an $\mathcal{H}$-matrix and no approximation is needed. The number of children for the internal nodes can be either 1 or 2. More about this is discussed in the next section.

## *2.4. $\mathcal{H}$-matrices*

A block partition tree $T_{I\times I}$, is a format to represent and store a matrix. The $\mathcal{H}$-matrix representation has the same tree structure as $T_{I\times I}$, and the matrix entries are actually stored in its leaves. If a leaf is an Rk-matrix, then its corresponding block is stored as an Rk-matrix (or rank-$k$) format; otherwise it is a full matrix.

We can recursively define the $\mathcal{H}$ matrices induced by $T_{I\times I}$ as follows:

For $r, s \in V(G_i)$, connected in $G_i$ and $\#L_r^{(i)}$, $\#L_s^{(i)} > N_s$, define an $\mathcal{H}_{r,s}$ as

$$\mathcal{H}_{r,s} = \left\{ \left[ H_{v,w} \mid v \in C_r^{(i)},\ w \in C_s^{(i)} \right] \mid H_{v,w} \in \mathcal{H}_{v,w} \right\}.$$

For $r, s \in V(G_i)$, but not connected in $G_i$, and $\#L_r^{(i)}$, $\#L_s^{(i)} > N_s$, define an $\mathcal{H}_{r,s}$ to be a Rk-matrix. For $r, s \in V(G_i)$ and $\#L_r^{(i)}$, $\#L_s^{(i)} \leq N_s$ define $\mathcal{H}_{r,s}$ matrix to be an ordinary dense $\#L_r^{(i)} \times \#L_s^{(i)}$ matrix.

As an example, the (hierarchical) structure of an $\mathcal{H}$-matrix with $N_s = 1$ based on the block cluster tree of Fig. 9 is shown in Fig. 10.

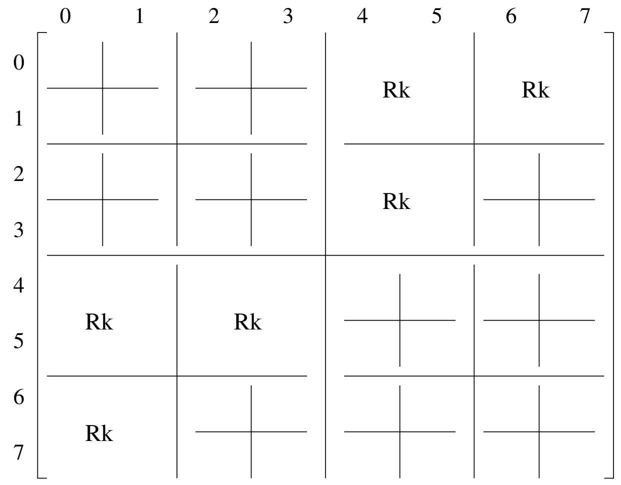


**Fig. 10.** The hierarchical structure of the $\mathcal{H}$-matrix with the block cluster tree in Fig. 9

## 3. $\mathcal{H}$-matrix arithmetic

Because of the special tree structure of a $\mathcal{H}$-matrix, the $\mathcal{H}$-matrix arithmetic is defined for $\mathcal{H}$-matrices. Its computation complexity, which depends on the tree structures of $\mathcal{H}$-matrices, is analyzed in [2], [6], [3]. In this section, we will give a brief review of some basic operations that are defined by $\mathcal{H}$-matrix arithmetic.

### *3.1. Properties of rank k-matrices*

The basic building blocks of $\mathcal{H}$-matrices are low rank matrices (called Rk-matrices). A $m \times n$ matrix $M$ is called an Rk-matrix if its rank $\leq k$ and is represented in the form of matrix product:

$$M = AB^T, \tag{4}$$

where $M$ is $m \times n$, $A$ is $m \times k$ and $B$ is $n \times k$.

If $k$ is much smaller than $m$ and $n$, by representing $M$ in Rk-matrix format we can save storage and reduce the computational complexity of the operators such as matrix-vector multiplication and matrix-matrix multiplication.

The computational complexity of Rk-matrix-Rk-matrix addition can also be reduced to $O(k^2(n+m)+k^3)$ by using the truncated singular value decomposition [2], which approximates the exact sum by an Rk-matrix.

Let $M_1 = A_1B_1^T$, $M_2 = A_2B_2^T$ be Rk-matrices. The truncated SVD which truncates $M = M_1 + M_2$ to an Rk-matrix $\widetilde{M}$ is defined as shown in Fig. 11.

### *3.2. Hierarchical truncation*

Sometimes it is necessary to approximate an $\mathcal{H}$-matrix by an Rk-matrix; this is needed occasionally for $\mathcal{H}$-matrix-matrix multiplication as described in Sect. 3.5.

If the given $\mathcal{H}$-matrix is already an Rk-matrix, there is nothing to be done. If the given $\mathcal{H}$-matrix is a full matrix, then we apply the above SVD algorithm for approximating general matrices by Rk-matrices. Otherwise, we first apply hierarchical truncation recursively to the blocks of the $\mathcal{H}$-matrix, as shown in Fig. 12. Then we need to truncate the $2 \times 2$ block matrix (with Rk-matrix blocks) to a single Rk-matrix. Since the $2 \times 2$ block matrix is a rank-$4k$ matrix, we can reduce the problem to finding the SVD of a single $4k \times 4k$ matrix. This provides the final truncation shown in Fig. 12.

```
Compute QR factorization $[A_1, A_2] = Q_A R_A$.   ($R_A$ is $k \times k$)
Compute QR factorization $[B_1, B_2] = Q_B R_B$.   ($R_B$ is $k \times k$)
Compute SVD $R_A R_B^T = U \Sigma V^T$.
$\widetilde{U} \leftarrow [U_1, \dots, U_k]$;  $\widetilde{V} \leftarrow [V_1, \dots, V_k]$
$\widetilde{A} \leftarrow Q_A \widetilde{U}$diag$(\Sigma_{11}, \dots, \Sigma_{kk})$;   $\widetilde{B} \leftarrow Q_B \widetilde{V}$
$M = \widetilde{A}\widetilde{B}^T$
```

**Fig. 11.** Pseudocode for approximate sum of Rk-matrices

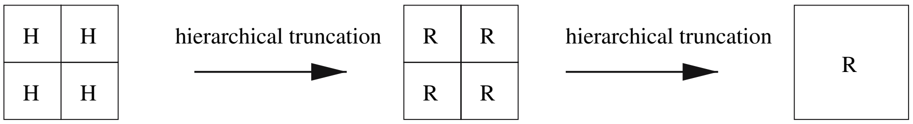


**Fig. 12.** The process of the hierarchical multiplication and truncation

### *3.3. $\mathcal{H}$-matrix-vector multiplication*

The product $Hv$ of an $\mathcal{H}$-matrix times a vector is computed recursively by the multiplication of the subblocks of a $\mathcal{H}$-matrix with the corresponding segments of $v$: if $H$ is a full matrix then use standard matrix-vector multiplication; if $H$ is an Rk-matrix then Rk-matrix-vector multiplication is called; else if $H$ has children, then for each child $H_i$ use $\mathcal{H}$-matrix vector multiplication. Note that this is an exact operation.

### *3.4. $\mathcal{H}$-matrix addition*

$\mathcal{H}$-matrix addition is defined by adding the corresponding subblocks of two $\mathcal{H}$-matrices with the same block partition tree $T_{I\times I}$ and its result is also a $\mathcal{H}$-matrix with the same tree structure: if the subblocks are full matrices then use exact matrix addition to add them and generate a block in the full matrix format; otherwise if they are Rk-matrices then use truncated Rk-matrix addition defined in the Sect. 3.1 to truncate the sum to an Rk-matrix. $\mathcal{H}$-matrix subtraction can be defined in a similar way.

### *3.5. $\mathcal{H}$-matrix multiplication*

The multiplication of two $\mathcal{H}$-matrices ($H \leftarrow H +_{\mathcal{H}} H_1 *_{\mathcal{H}} H_2$) ($H_1$ with $T_{I\times J}$ and $H_2$ with $T_{J\times K}$) gives a $\mathcal{H}$-matrix $H$ with $T_{I\times K}$. So depending on the cluster tree structures there can be four cases:

(1) $H_1$, $H_2$ and $H$ all have subblocks. Then multiplication is done recursively on the subblocks.
(2) If $H$ has subblocks but $H_1$ or $H_2$ does not, then the product, which is an Rk-matrix or full matrix, is partitioned and added to $H$.
(3) If $H$ is a full matrix, then the product is added to the target directly.
(4) If $H$ is an Rk-matrix, then the hierarchical multiplication and truncation is called to get a product in Rk-matrix format, as described in Sect. 3.2.

### *3.6. $\mathcal{H}$-matrix inversion*

Let $+_{\mathcal{H}}$ and $*_{\mathcal{H}}$ be $\mathcal{H}$-matrix addition and multiplication as defined in Sect. 3.4 and 3.5. $\mathcal{H}$-matrix inversion is based on the block Gauss-Jordan elimination; the exact matrix operators are replaced by the $\mathcal{H}$-matrix arithmetic which computes the approximate inverse in a cheaper way. Let $H = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}$ be a $2 \times 2$ block on the top level and $inv_{\mathcal{H}}$ denote $\mathcal{H}$-matrix inversion. The process of $\mathcal{H}$-matrix inversion is defined recursively as

$$inv_{\mathcal{H}}(H) = \begin{bmatrix} H'_{11} & H'_{12} \\ H'_{21} & H'_{22} \end{bmatrix}, \quad \text{where} \tag{5}$$

$$\begin{aligned}
S &= H_{22} -_{\mathcal{H}} H_{21} *_{\mathcal{H}} inv_{\mathcal{H}}(H_{11}) *_{\mathcal{H}} H_{21}, \\
H'_{11} &= inv_{\mathcal{H}}(H_{11}) +_{\mathcal{H}} inv_{\mathcal{H}}(H_{11}) *_{\mathcal{H}} H_{12} *_{\mathcal{H}} inv_{\mathcal{H}}(S) *_{\mathcal{H}} H_{21} *_{\mathcal{H}} inv_{\mathcal{H}}(H_{11}), \\
H'_{12} &= -inv_{\mathcal{H}}(H_{11}) *_{\mathcal{H}} H_{12} inv_{\mathcal{H}}(S), \\
H'_{21} &= -inv_{\mathcal{H}}(S) *_{\mathcal{H}} H_{21} *_{\mathcal{H}} inv_{\mathcal{H}}(H_{11}), \\
H'_{22} &= inv_{\mathcal{H}}(S).
\end{aligned} \tag{6}$$

## *3.7. $\mathcal{H}$-matrix LU factorization and Cholesky factorization*

We also can factor a $\mathcal{H}$-matrix by $\mathcal{H}$-LU factorization or $\mathcal{H}$-Cholesky factorization, which generates approximate factors in the $\mathcal{H}$-matrix format.

The complexity of $\mathcal{H}$-LU with respect to $T_{I\times I}$ is $O(n \log^2 n\, k^2)$ [6]. Note that [6] also shows the recursive way to compute $\mathcal{H}$-LU factors step by step. An approximate $\mathcal{H}$-matrix triangular solve algorithm can be developed recursively provided all diagonal leaf blocks are nonsingular full matrices. An upper triangular $\mathcal{H}$-matrix $U$ is either an upper triangular full matrix (if it is a leaf node), or has the recursive structure

$$U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \tag{7}$$

with $U_{11}$ and $U_{22}$ upper triangular $\mathcal{H}$-matrices. A lower triangular $\mathcal{H}$-matrix is a matrix whose transpose is an upper triangular $\mathcal{H}$-matrix. Solving (approximately) $UX = B$ for $X$ where $B$ is a consistently structured $\mathcal{H}$-matrix can then be done recursively using blockwise back-substitution. Then the $\mathcal{H}$-LU factorization of

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix} \tag{8}$$

is computed by the pseudocode shown in Fig. 13.

Using triangular solves, $\mathcal{H}$-Cholesky factorization can also be defined in a similar way.

The $\mathcal{H}$-matrix inverse, $\mathcal{H}$-LU factors and $\mathcal{H}$-Cholesky factors can be used as preconditioners in iterative methods. Since the complexity of $\mathcal{H}$-matrix inversion involves large constants and $\mathcal{H}$-LU and $\mathcal{H}$-Cholesky are cheaper to perform, instead of using a $\mathcal{H}$-inverse, we use $\mathcal{H}$-LU or $\mathcal{H}$-Cholesky factors as preconditioners in iterative methods to solve systems of linear equations.

```
$\mathcal{H}$-LU-factor($A$)
  if $A$ is full
    return $L, U$ where $A = LU$ (ordinary $LU$ factorization)
  end if
  $[L_{11}, U_{11}] \leftarrow$ $\mathcal{H}$-LU-factor($A_{11}$)
  $\mathcal{H}$-triangular solve $A_{12} = L_{11}U_{12}$ for $U_{12}$
  $\mathcal{H}$-triangular solve $A_{21} = L_{21}U_{11}$ for $L_{21}$
  $[L_{22}, U_{22}] \leftarrow$ $\mathcal{H}$-LU-factor($A_{22} -_{\mathcal{H}} L_{21} *_{\mathcal{H}} U_{12}$)
  return $\begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix}, \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}$
end $\mathcal{H}$-LU-factor
```

**Fig. 13.** $\mathcal{H}$-matrix LU factorization

## 4. Experimental results

In this section, we show the experimental results of applying our algebraic $\mathcal{H}$-matrix construction approach combined with $\mathcal{H}$-matrix arithmetic to solve two kinds of systems of linear equations, both arising from two different discretization methods to construct approximate solutions to partial differential equations.

To show the performance of our approach, which is based on HEM, we compare it with the domain decomposition clustering algorithm using Nested Dissection (ND) [7]. We also compared $\mathcal{H}$-preconditioners obtained by the algebraic approaches with other preconditioners when used with GMRES. The preconditioners are: $\mathcal{H}$-inverse based on HEM (HEM-$\mathcal{H}$-INV), $\mathcal{H}$-LU factors based on HEM (HEM-$\mathcal{H}$-LU), $\mathcal{H}$-Cholesky factors based on HEM (HEM-$\mathcal{H}$-CH), $\mathcal{H}$-LU factors based on ND (ND-$\mathcal{H}$-LU), $\mathcal{H}$-Cholesky factors based on ND (ND-$\mathcal{H}$-CH), as well as Jacobi Over-Relaxation (JOR) and smoothed aggregation AMG [8].

GMRES did not converge for the $\mathcal{H}$-inverse based on Nested Dissection (ND-$\mathcal{H}$-INV) ($k = 4$) used as a preconditioner for our test problems. Increasing the rank of the Rk matrices to $k = 8$ or higher did result in convergence; however, the cost of this approach greatly exceeded the cost of using the H-inverse based on HEM.

The iteration stops when the original residual was reduced by the factor of $10^{-12}$ as measured by the 2-norm. The convergence rate $a$, defined as the average decreasing speed of residuals in each iteration, can be obtained by solving the equation: $a^t = 10^{-12}$, where $t$ is the number of iterations.

To compute $\mathcal{H}$-LU and $\mathcal{H}$-Cholesky factors, we used $\mathcal{H}$-matrix arithmetic with adaptive ranks: the rank of each Rk-matrix block $M_{L_i \times L_j}$ approximating a matrix $A$ in a $\mathcal{H}$-matrix satisfies $\mathrm{rank}(M_{L_i \times L_j}) = \min\{ k \mid \Sigma_k \le \alpha \Sigma_1 \}$, where $\Sigma_i$ is $i$th largest singular value of $A$, and $\alpha$ is a parameter to control the accuracy. In this paper we choose $\alpha = 0.0625$.

But to compute $\mathcal{H}$-inverses, we used the $\mathcal{H}$-matrix arithmetic with fixed ranks: the rank of each Rk-matrix block $M_{L_i \times L_j}$ is less than $k$, since the fixed rank $\mathcal{H}$-matrix arithmetic had better speed and convergence rates than the adaptive rank. Here we set $k = 4$.

We also set the sizes of all the leaf blocks in a $\mathcal{H}$-matrix to be smaller than $N_s = 40$.

The Meschach library [10] was used for the data structures and functions related to full matrices and vectors.

### *4.1. $\mathcal{H}$-matrix preconditioners for finite element method*

The first problem is to solve a system

$$Kb = f, \tag{9}$$

where $K$ is a stiffness matrix. The experiments in this section were carried out on a dual processor computer with 64-bit Athlon 4200++ CPU's and 3GB of memory.

$K$ is constructed by using the grid generator of Persson and Strang [9] and applying the piecewise linear finite element method for $\nabla^2 u = f$ in $\Omega$ and $u(x, y) = e^{2x} \cos(2y) + x^3 - 3xy^2$ on $\partial\Omega$. Note that $K$ is sparse, symmetric, and positive definite. Figure 14 shows an example of a mesh with the element size $h_0 \approx 0.2$, and the distribution of the nonzero entries in $K$ based on the mesh.

In our test, we chose $h_0 \approx 0.020$, 0.015, 0.012, 0.010, 0.007, and 0.005. The corresponding number of unknowns are: $n = 8753$, 15697, 24657, 35632, 73131, and 143834, respectively.

Figure 15 shows the time required for HEM and ND to build a cluster tree and to build the corresponding $\mathcal{H}$-matrix over $K$. Figure 16 compares the time taken to compute $\mathcal{H}$-LU factors, $\mathcal{H}$-Cholesky factors and $\mathcal{H}$-inverses based on the $\mathcal{H}$-matrices built in the previous stage. Figure 17 compares their corresponding memory storage. Figure 18 shows the time taken by preconditioned GMRES for the given preconditioners.

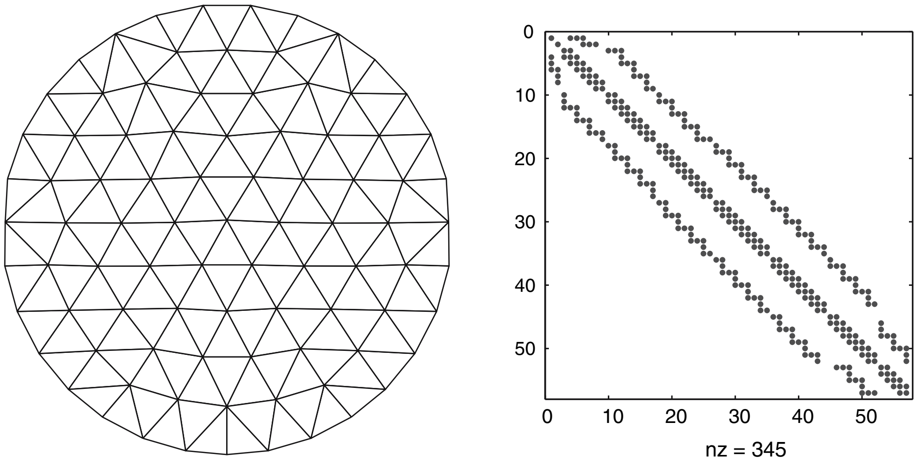
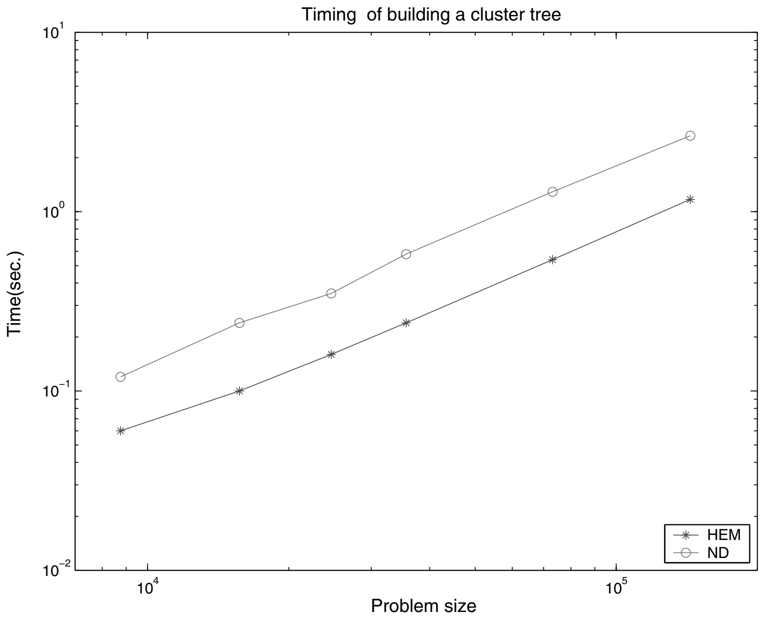


**Fig. 14.** The left side is a roughly uniform mesh on a unit circle with the element size $h_0 \approx 0.2$. The right side is the distribution of the nonzero entries (black dots) in the stiffness matrix $K$ based on the left mesh



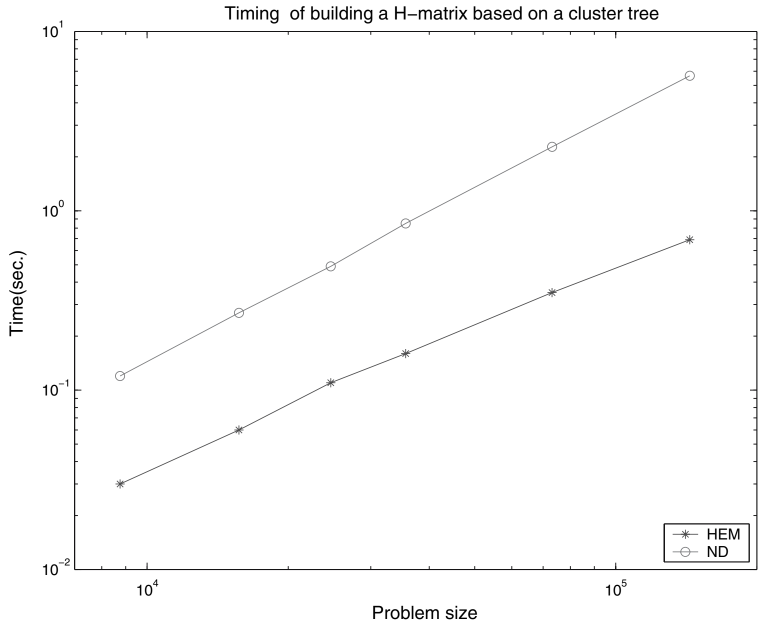


**Fig. 15.** Comparison of the times for HEM and ND to build a cluster tree $T_{I\times I}$ (left) and the time to build a H-matrix using $T_{I\times I}$ (right) for the finite element problem

Figure 19 compares the total running time (the time to construct preconditioners plus the time for preconditioned GMRES) and Fig. 20 shows the convergence rates of JOR and $\mathcal{H}$-matrix preconditioners. Based on the above figures, we can see

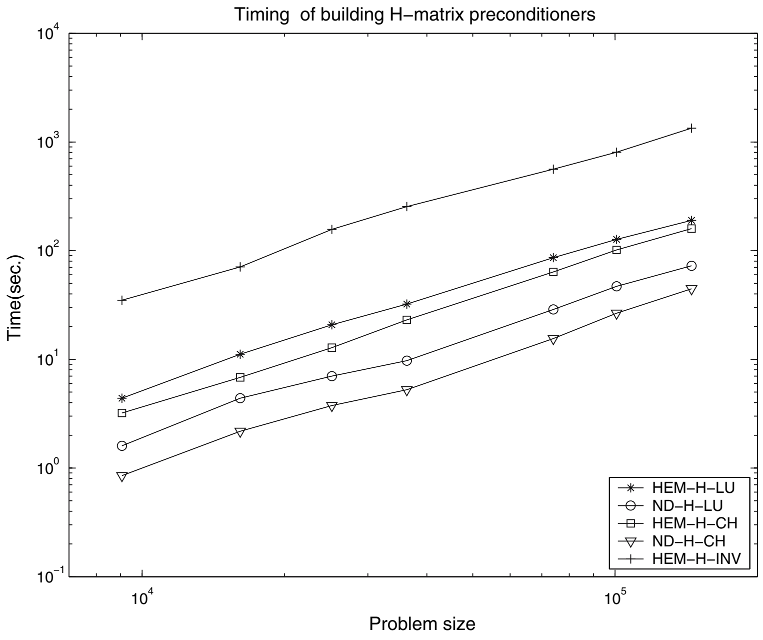


**Fig. 16.** Comparison of the times to construct $\mathcal{H}$-matrix preconditioners (FEM problem)

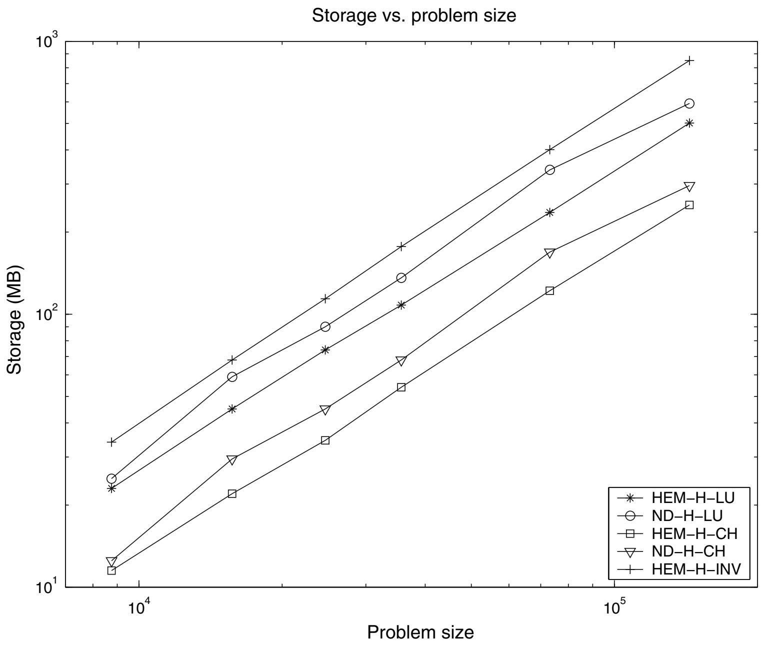


**Fig. 17.** Comparison of the storage of different $\mathcal{H}$ matrix preconditioners

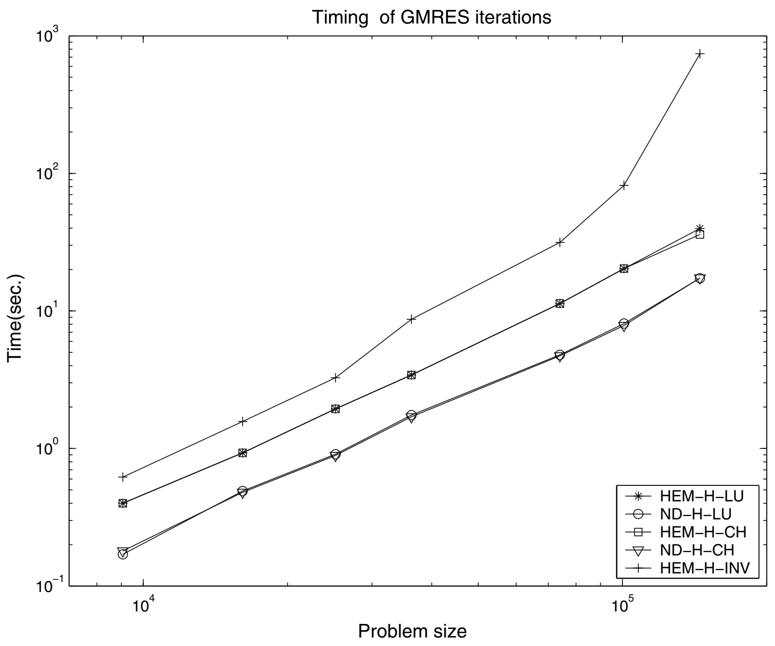


**Fig. 18.** Comparison of the times for GMRES with various preconditioners (FEM problem)

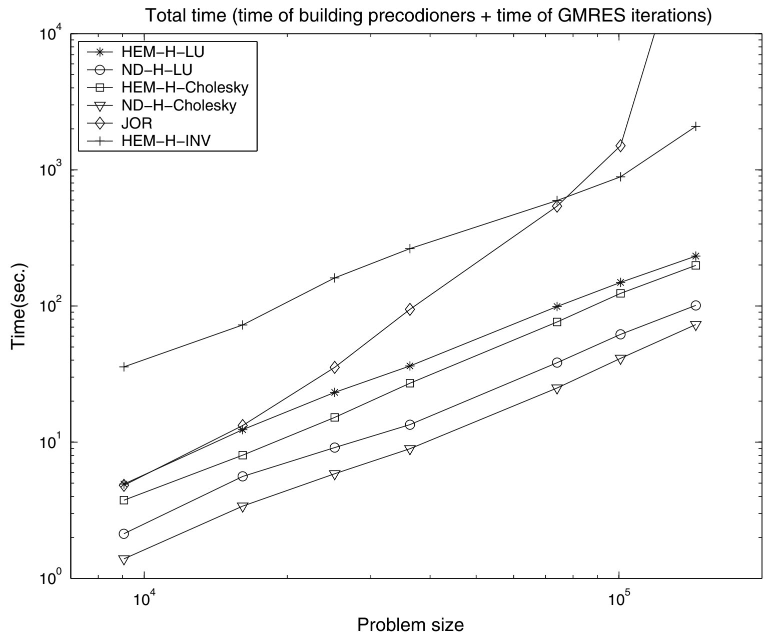


**Fig. 19.** Comparison of the total running time of GMRES with different preconditioners (FEM problem)

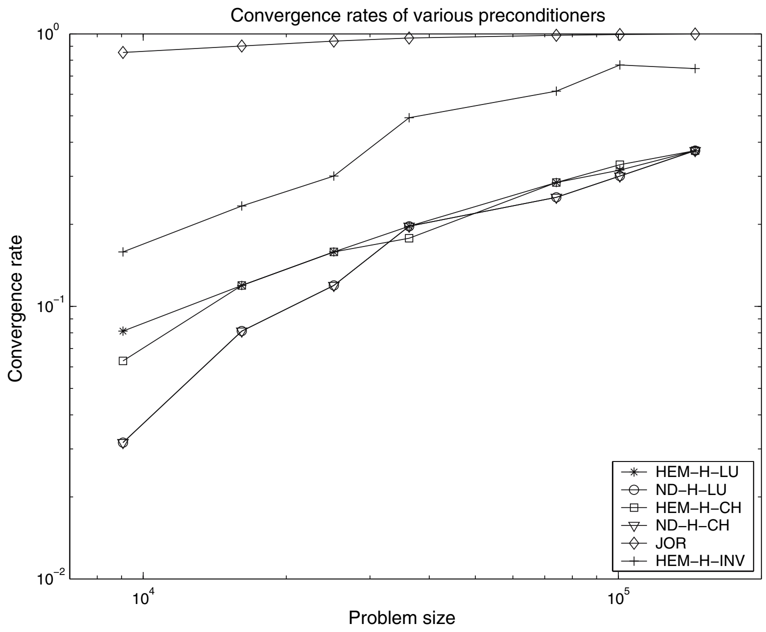


**Fig. 20.** Comparison of the convergence rates of GMRES with different preconditioners (FEM problem)

that with respect to the convergence rate, all the $\mathcal{H}$-matrix preconditioners outperform JOR; as to the total running time, $\mathcal{H}$-LU and $\mathcal{H}$-Cholesky factors are better and increase more slowly with problem size than JOR. $\mathcal{H}$-inverses outperform JOR when the problem size is bigger than $10^5$. $\mathcal{H}$-LU and $\mathcal{H}$-Cholesky factors are cheaper to compute than $\mathcal{H}$-inverses. HEM and ND are comparable with each other as to the total running time and the convergence rates.

### *4.2. $\mathcal{H}$-matrix preconditioners for saddle point systems*

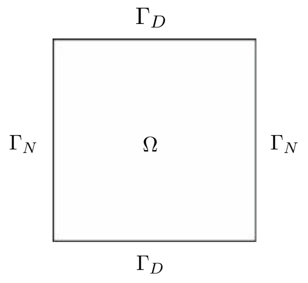
The second problem is the saddle point system, generated by meshfree discretizations to a second-order partial differential equation defined on a domain $\Omega \in \mathbf{R}^2$ [8]:

$$\begin{cases} -\nabla^2 u(\underline{x}) = f(\underline{x}),\ \underline{x} \in \Omega \\ u(\underline{x}) = g(\underline{x}),\ \underline{x} \in \Gamma_D \\ (\partial u/\partial n)(\underline{x}) = h(\underline{x}),\ \underline{x} \in \Gamma_N, \end{cases} \tag{10}$$

where $\Gamma_D \cup \Gamma_N$ is the boundary of $\Omega$ and the domain $\Omega$ is $(0, 1) \times (0, 1)$.

A Meshfree scheme, the Reproducing Kernel Particle Method (RKPM), is used to discretize the continuous problem (10). The generated meshfree linear system $Kx = F$ is

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ \lambda \end{bmatrix} = \begin{bmatrix} c \\ d \end{bmatrix}. \tag{11}$$



The stiffness matrix $K := \begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix}$ in (11) is symmetric but indefinite. But the submatrix $A$ is symmetric positive semi-definite. To avoid preconditioning an indefinite system, we solve the following equivalent system:

$$\begin{bmatrix} A & B^T \\ -B & 0 \end{bmatrix} \begin{bmatrix} u \\ \lambda \end{bmatrix} = \begin{bmatrix} c \\ -d \end{bmatrix}. \tag{12}$$

To construct preconditioners, one option is to compute the approximate $\mathcal{H}$-inverse over $K$ (HEM-$\mathcal{H}$-INV) and use it as a preconditioner. Another option is to factorize $K$. Since $K$ is indefinite we can not directly apply Cholesky factorization to it, but we can factorize $K$ in the following way:

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ L_2 & L_3 \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} L_1^T & 0 \\ L_2^T & L_3^T \end{bmatrix} = \begin{bmatrix} L_1 & 0 \\ L_2 & -L_3 \end{bmatrix} \begin{bmatrix} L_1^T & L_2^T \\ 0 & L_3^T \end{bmatrix}, \tag{13}$$

where $L = \begin{bmatrix} L_1 & 0 \\ L_2 & L_3 \end{bmatrix}$ is a lower triangular matrix and $U = \begin{bmatrix} L_1^T & L_2^T \\ 0 & L_3^T \end{bmatrix}$ is an upper triangular matrix. We can use them as preconditioners in GMRES.

To approximate the submatrices $L_1$, $L_2$ and $L_3$ we use the following approach:

(1) Since $A$ is symmetric positive semi-definite and $A = L_1 L_1^T$, we approximate $A$ by an $\mathcal{H}$-matrix $A_{\mathcal{H}}$; then apply $\mathcal{H}$-Cholesky factorization to $A_{\mathcal{H}}$ and we obtain $L_1$ which is represented in the $\mathcal{H}$-matrix format.
(2) Since $B = L_1 L_2^T$, by solving an $\mathcal{H}$-matrix lower triangular system $L_1 L_2^T = B$ we can get $L_2$. To save storage and to speed up the computation, $L_2$ is represented by the Meschach sparse matrix format [10].
(3) Since $L_3 L_3^T = L_2 L_2^T$ and $L_2 L_2^T$ is relative small, we apply the ordinary Cholesky factorization to the product and get $L_3$ in the full matrix format.

We use HEM-$\mathcal{H}$-CH or ND-$\mathcal{H}$-CH to indicate the above factorization according to the method used to build the cluster tree.

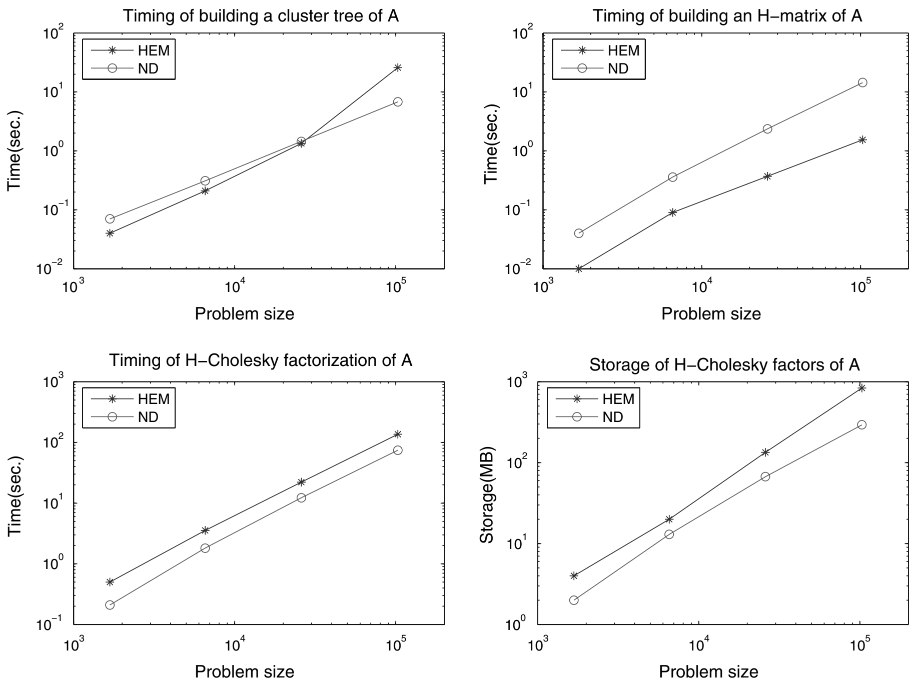


**Fig. 21.** Comparison of HEM-$\mathcal{H}$-Cholesky and ND-$\mathcal{H}$-Cholesky factorization of $A$ in the saddle point systems

In our test set, the number of the basis functions for the domain $\Omega$ is $N_\Omega = 1600, 6400, 25600, 102400$ and the corresponding number of the boundary basis functions is $N_\Gamma = 80, 160, 320, 640$. Thus the problem sizes are $n = 1680$, 6560, 25920, and 103040 respectively. The experiments in this section were carried out on a dual processor Dell computer wint Intel Xeon CPU's running at 2.4GHz.

Figure 21 compares HEM and ND based $\mathcal{H}$-Cholesky factorization of $A$. The upper left figure shows the time to build a cluster tree $T_I$; the upper right shows the time to build $A_\mathcal{H}$ based on $T_I$; the lower left shows the time to factor $A_\mathcal{H}$; and the lower right shows the required storage of $L_1$ in megabytes (MB). Based on Fig. 21, the overall performance of HEM and ND are very close to each other.

Figure 22 compares the times and convergence rates of the following preconditioners: JOR, smoothed aggregation AMG [8], $\mathcal{H}$-Cholesky based factors (13) and HEM-$\mathcal{H}$-INV. The time in the figure includes the time to build the preconditioners and the time of GMRES iterations.

As to JOR and AMG, we did not have enough memory for $n = 103,040$. Based on Fig. 22, we can see that the approach we described above to factor $K$ (13) works really well, while both HEM and ND outperform JOR and AMG with respect to the total running time and convergence rates. ND is the best among the preconditioners and HEM is comparable.

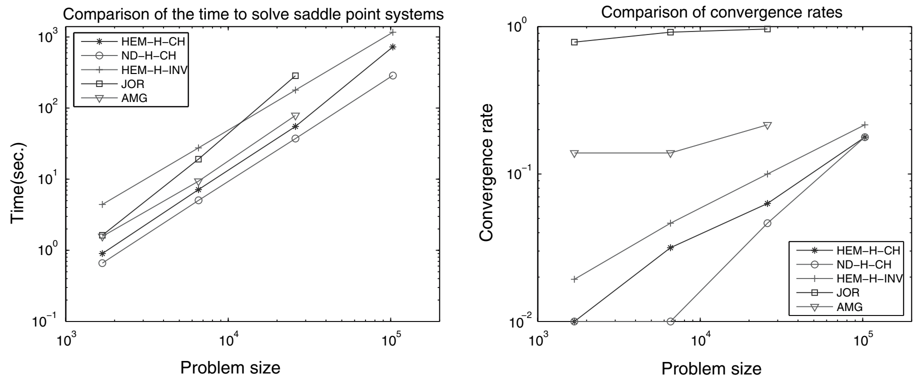


**Fig. 22.** Comparison of various preconditioners to solve the saddle point systems

## Acknowledgements

I would like to thank the referees for various suggestions regarding the presentation of the paper, and also Prof. Kendall Atkinson for proofreading the final version. This work was supported by NSF ITR grant DMS-0213305.

## References

[1] Börm, S., Grasedyck, L., Hackbusch, W.: Hierarchical matrices. Technical report, Max-Planck-Institut für Mathematik in den Naturwissenschaften, Leipzig, Germany 2003. Lecture Notes No. 21. Available online at `www.mis.mpg.de/preprints/ln/`

[2] Börm, S., Grasedyck, L., Hackbusch, W.: Introduction to hierarchical matrices with applications. EABE *27*, 403–564 (2003).

[3] Grasedyck, L., Hackbusch, W.: Construction and arithmetics of $\mathcal{H}$-matrices. Computing *70*(4), 295–334 (2003).

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

[5] Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. *20*(1), 359–392 (1999).

[6] Le Borne, S., Grasedyck, L.: $\mathcal{H}$-preconditioners in convection-dominated problems. SIAM J. Matrix Anal. Appl. *27*(4), 1172–1183, (2006).

[7] Le Borne, S., Grasedyck, L., Kriemann, R.: Domain-decomposition based $\mathcal{H}$-LU preconditioners. In: Proc. 16th Int. Conf. on Domain Decomposition Methods (New York, 2005). Springer: LNCSE 2006 (forthcoming).

[8] Leem, K. H., Oliveira, S., Stewart, D.: Algebraic multigrid (AMG) for saddle point systems from meshfree discretizations. Numer. Linear Algebra Appl. *11*(3), 293–308 (2004).

[9] Persson, P., Strang, G.: A simple mesh generator in Matlab. SIAM Rev. *46*(2) (2004).

[10] Stewart, D. E., Leyk, Z.: Meschach: Matrix computations in C. Proc. CMA, vol. 32. The Australian National University 1994.

S. Oliveira and F. Yang
Department of Computer Science
University of Iowa
14 McLean Hall
Iowa City
USA
e-mails: {oliveira, fayang}@cs.uiowa.edu