Multifrontal Incomplete Factorization for Indefinite and Complex Symmetric Systems

Yong Qu and Jacob Fish
Departments of Civil, Mechanical and Aerospace Engineering
Rensselaer Polytechnic Institute, Troy, NY 12180

ABSTRACT

A new class of preconditioners based on the adaptive threshold incomplete multifrontal factorization for indefinite and complex symmetric systems is developed. Numerical experiments consisting of the 3D Helmholtz equations, fluid-structure interaction and localization problems demonstrate the excellent performance of the preconditioner.

Keywords: multifrontal, incomplete factorization, indefinite, complex, adaptive

1.0 Introduction

The primary goal of the manuscript is to construct a black-box preconditioner with efficient memory-system performance and automated/adaptive selection of algorithmic parameters. Attention is restricted to the incomplete factorization precondtioner which has been found to be very effective on a wide variety of problems [3][10]. We first briefly survey various incomplete factorization methods followed by a preliminary discussion on how to improve the memory-system performance and how to adaptively select the algorithmic parameters.

Consider the following linear system of equations

$$Ax = b \tag{1}$$

where $A \in C^{N \times N}$ is a general symmetric matrix and $A = LL^T - E$ is an incomplete factorization. The preconditioned linear system may be represented as

$$(L^{-1}AL^{-T})(L^T x) = L^{-1}b \tag{2}$$

The ultimate goal of the preconditioning is to make the preconditioned system, $L^{-1}AL^{-T}$, converge faster than the original system. In other words, the eigenvalue spectrum of the iteration matrix, $I_N - L^{-1}AL^{-T}$, should be designed to be densely clustered around *0*.

Since the pioneering work of Meijerink and van der Vorst [5]on the incomplete factorization, most of the attention focussed on selecting the best set of entries to drop in the factorization process. The dropping strategies include dropping fill-ins based on numerical value, dropping fill-ins based on matrix structure and the combination of these two strategies [7]. It has been shown [7][10][11] that dropping by value is preferential, particularly for complex and indefinite matrices. Recently, research efforts have been focussing on block incomplete factoriza-

tion and multilevel incomplete factorization methods. In the block incomplete factorization [8], the original matrix is divided into small dense blocks based on the structure of the matrix prior to the factorization and single entries of the incomplete factorization are replaced with dense blocks. The blocking technique can also be used with pivoting to avoid zero or nearly zero pivots [3][7]. In [16], Saad and Zhang introduced a multi-elimination incomplete factorization which exploits the existence of independent sets of unknowns and performs incomplete factorization on several levels of reduced systems. A family of multigraph incomplete factorization algorithms was presented in [15] by Bank and Smith. Their method recursively coarsens the graph of the sparse matrix by eliminating vertices using a graph model similar to Gaussian elimination and obtains the corresponding incomplete factorization by allowing only a limited amount of fill-ins.

Two issues, improving memory-system performance and adaptive selection of algorithmic parameters, have a significant effect on the efficiency and robustness of the incomplete factorization based solver, but somewhat surprisingly, received very little attention in the literature. These two issues are subsequently discussed.

1.1 Improving memory-system performance

For modern machines with shared-memory vector and parallel processors, the incomplete factorization algorithms become inefficient if their memory access patterns disregard the multi-layered memory hierarchies of the machines [13]. One main factor which causes the poor performance is cache miss. Most CPUs have first-level instruction and data caches on chip and many have second-level cache(s) that are bigger but somewhat slower. Memory accesses are much faster if the data is already loaded into the first-level cache. When the program accesses data that isn't in one of the caches, it gets a cache miss. This causes a block of consecutively addressed words, including the data that was just accessed, to be loaded into the cache. This process is expensive since it may load many unwanted words into the cache, thereby spending much time on moving the data at the expense of performing floating-point operations. The other factor is the bottleneck caused by a large amount of load instructions relative to the number of floating-point operations. The load instructions will quickly stall the load/store units and leave the floating-point units underutilized.

Both factors are caused by the lack of data locality due to the structures of the matrices and the algorithms. For example, most incomplete factorization algorithms for sparse symmetric matrices are based on Crout-Doolittle factorization which factorizes the matrices row-by-row (or column-by-column). For each row (column), one needs to loop over all previous rows (column) of the factor which have contributions on the current row (column). The updates are computed and assembled before moving onto the next row (column). It is clear that this algorithm has a poor data reuse since it may need to reload the same row (column) many times. Also the sparse matrix data structures used in these algorithms normally need pointer direction in their inner most loops which may cause a large amount of load instructions.

The problem is not very serious for the incomplete factorizations of very sparse and positive definite matrices because the total number of the float-point operations is much smaller compared to the complete factorization. However, the inefficiency becomes significant for complex or indefinite matrices which may require a large number of fill-ins and for dense matrices

(e.g. the matrices generated by the *p*-version finite element method with a high order *p*) which may require much more floating-point operations than the sparse matrices.

The techniques to overcome the inefficiency caused by the lack of data locality include:

- Develop better renumbering schemes to reduce the total number of floating-point operations.
- Keep frequently accessed data together and access data sequentially. Store and access frequently used data in flat, sequential data structures and avoid pointer indirection.
- Pack the data to force a big increase in the size of the data. In other words, block techniques should be used to reduce the number of load instructions
- Unroll the inner most loop to improve the data reuse.
- Use dense matrix operations, such as level 2 and level 3 BLAS primitives [14], as much as possible in the innermost loops.

To demonstrate the effects of these techniques, we consider the following sparse matrix multiplication which is used in the sparse matrix factorization.

$$B = vv^T \tag{3}$$

where $v \in \Re^N$ is a sparse vector with m nonzeros and $B \in \Re^{N \times N}$ is a sparse matrix.

The following four algorithms are designed to compare the performance of different techniques. All elements of v are stored in an array with size N. All nonzeros of B are stored in a dense array with size m^2. The indices of nonzeros in v are stored in an integer array I.

Algorithm (1) Sparse matrix multiplication with index loading

```
for i = 1: m
    for j = 1: m
        B[(i − 1) × m + j] = v[I[i]] × v[I[j]]
    endfor
endfor
```

Algorithm (2) Sparse matrix multiplication without index loading

```
Save all nonzeros of v into a dense array u
    for i = 1: m
        u[i] = v[I[i]]
    endfor
for i = 1: m
    for j = 1: m
        B[(i − 1) × m + j] = u[i] × u[j]
    endfor
endfor
```

Algorithm (3) Sparse matrix multiplication with BLAS primitive

Save all nonzeros of v into a dense array u
for i = 1: m
$u[i] = v[I[i]]$
endfor
Call BLAS 2 primitive DGMM to perform the matrix multiplication

Algorithm (4) Sparse matrix multiplication with BLAS primitive and blocking

Save all nonzeros of v into a dense array u
for i = 1: m
$u[i] = v[I[i]]$
endfor
*Divide u into n blocks. Each block has the length k. (The length of the last block is m - (n-1)*k)*
for i = 1: n
for j = 1: n
Call BLAS 2 primitive DGMM to perform the matrix multiplication between the ith block and the jth block.
endfor
endfor

Table 1 summarizes the results of the sparse matrix multiplication with m = 1,000 and N = 10,000. All programs are optimized using level -O3 [6] and run on a Sun Ultra 60 machine.

Algorithm	Block Size	CPU time (sec.)
1		0.919
2		0.671
3		0.564
4	10	0.530
	20	0.246
	50	0.230
	100	0.265
	200	0.267

TABLE 1. Comparisons of Solver CPU times

Several observation can be made from Table 1 :

- All techniques improve the performance of sparse matrix multiplication.
- The best technique is to use the dense matrix operations, such as the BLAS primitives, in the innermost loops.
- The performance of the BLAS primitives also depends on the size of the dense matrix. The blocking technique can further improve the performance.

Multifrontal methods [9] address this problem by reorganizing the factorization algorithms to use the dense matrix operations, such as the level 2 and level 3 BLAS primitives, in the innermost loops. These dense matrix operations can be optimized for each architecture to account

for the memory hierarchy, and to provide a transportable way to achieve high efficiency on diverse modern machines. In [12], Campbell and Davis presented a class of multifrontal incomplete factorization preconditioners, but their dropping strategy was based on fill-in positions, which makes it inefficient for indefinite and complex systems. In Section 2.0 , we will describe how to tailor the state-of-the-art complete multifrontal factorization method [21] to obtain an incomplete version of the Multifrontal factorization with dropping strategies based on the numerical value.

1.2 Adaptive Parameter Selection

For general symmetric systems, it is not feasible to *a prior* estimate the optimal algorithmic parameters of the incomplete factorization, such as the maximum number of allowable fill-ins, the dropping tolerance *etc*. The optimal parameters depend on many factors such as eigenvalue spectrum (positive definite, indefinite or complex), problem size and sparsity.

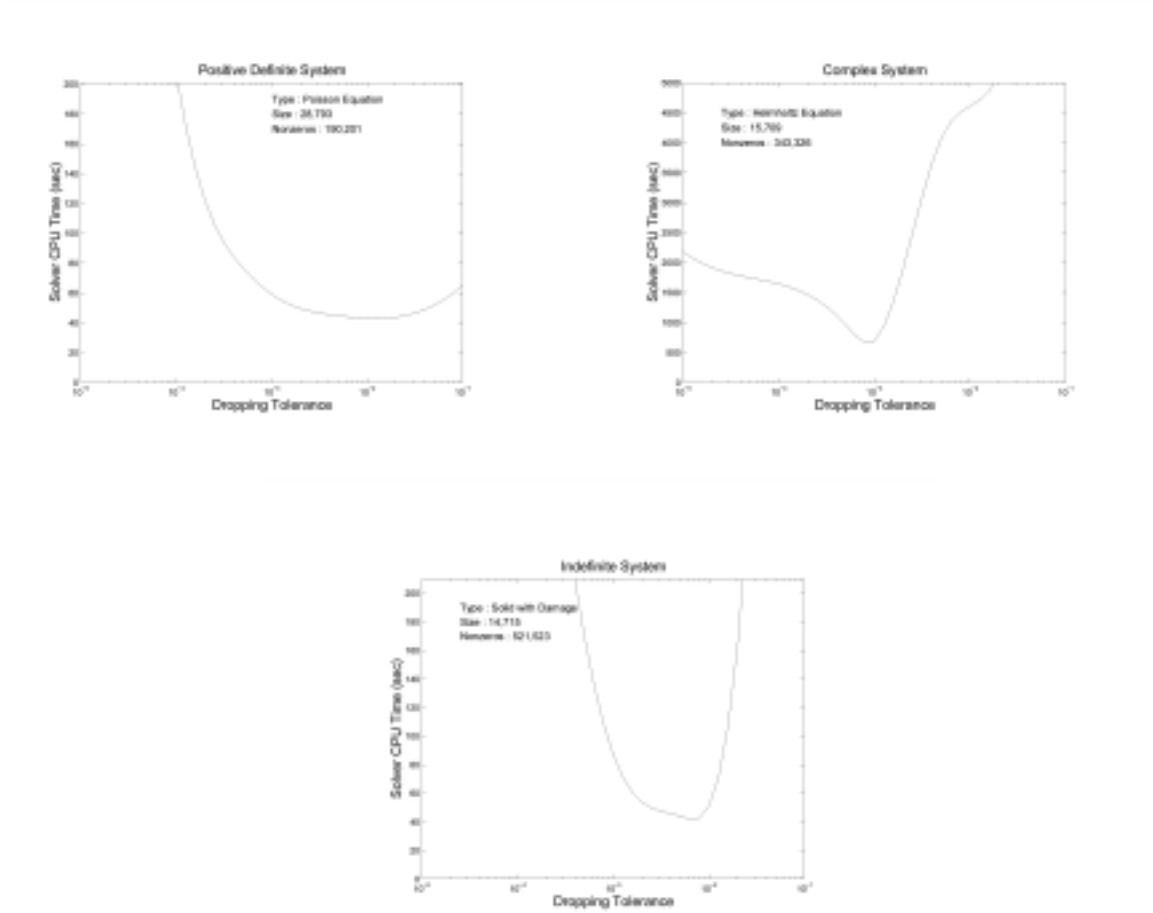


FIGURE 1. Solver CPU time vs. Dropping Tolerance

As an illustration, a threshold incomplete LDL^T factorization preconditioner [3] coupled with the QMR method is employed to solve three different types of problems, a positive definite

problem (Poisson equation), a complex problem (Helmholtz equation) and an indefinite problem (Solid with damage zone). The most important parameter investigated is the dropping tolerance. Figure 1 shows the computational cost versus the dropping tolerance for all three problems. It can be seen that these problems have a different behavior with respect to the parameter selection. Complex and indefinite problems are more sensitive to the dropping tolerance than the positive definite problem. It is not feasible to *a prior* estimate the optimal value of the parameter without further knowledge of the matrix properties. In [2][3], we proposed a predictor-corrector based framework to adaptively select algorithmic parameters for the incomplete factorization preconditioners. In this paper, we will extend this framework to the threshold multifrontal incomplete factorization. Details are described in Section 2.0 .

The paper is organized as follows. Section 2.0 details the threshold multifrontal incomplete factorization. Numerical experiments on a sequence of examples involving the Helmholtz equation, the fluid-structure interaction problem and the shear banding problem are presented in Section 3.0 . A brief discussion on future research directions concludes the manuscript.

2.0 Multifrontal Incomplete Factorization Precondtioner

In this section, we introduce a new class of preconditoners based on the threshold incomplete multifrontal factorization. We first describe the threshold incomplete multifrontal Cholesky factorization with separate analyze and factorize phases. This algorithm can be used for positive definite and complex symmetric systems. Details of the algorithm are given in Section 2.1 This is followed by a description of how to adaptively select parameters and how to extend it to incomplete LDL^T factorization for indefinite (real) systems in Section 2.2 and Section 2.3 . In Section 2.4 , we comment on an out-of-core version of the multifrontal incomplete factorization.

2.1 Threshold incomplete multifrontal Cholesky factorization

The proposed threshold incomplete multifrontal Cholesky factorization algorithm is derived from the state-of-the-art complete multifrontal Cholesky factorization algorithm developed in [21]. The main features of the complete multifrontal Cholesky factorization are: (i) It has separate analyze and factorize phases; (ii) Several renumbering schemes are considered in an attempt to find the best renumbering scheme; (iii) In the analyze phase, adjacent rows with the same nonzero structures after symbolic factorization are merged in order to exploit the use of dense matrix operations in the factorize phase; (iv) A two-end stack is used to collect and assemble contribution matrices during factorization.

The threshold incomplete multifrontal Cholesky factorization algorithm is in principle similar to the complete multifrontal Cholesky factorization. A dropping scheme based on the numerical value is inserted into factorization process to reduce the number of fill-ins in the factor. At the same time, the use of dense matrix operations is maintained in the inner most loops, which is vital for computational efficiency. An outline of this algorithm is given below.

Algorithm (5): Multifrontal Incomplete Factorization

A: Initialization

B: Analyze Phase: Create a sparse symmetric matrix and multiple frontals

B1: Find the best renumbering schemes

B2: Build an elimination tree and assign weight at each node

B3: Post-reorder the nodes according to the weights by traversing the tree depth-first

B4: Recompute the symbolic factorization

B5: Rebuild the elimination tree

B6: Collapse it to a supernodal elimination tree

C: Factorize Phase: Perform numerical incomplete factorization

C1: Estimate and allocate enough memory for the collection-assembly stack

C2: Scale the original matrix

C3: Loop over all supernodes (or fronts)

C31: Fill the current front with entries from the scaled original matrix

C32: Merge in the updates from its children

C33: Drop some entries according to their numerical values

C34: Factorize the current front and save the factor

C35: Compute the update of the current front

C3: End of the loop

C4: Back scale the factor

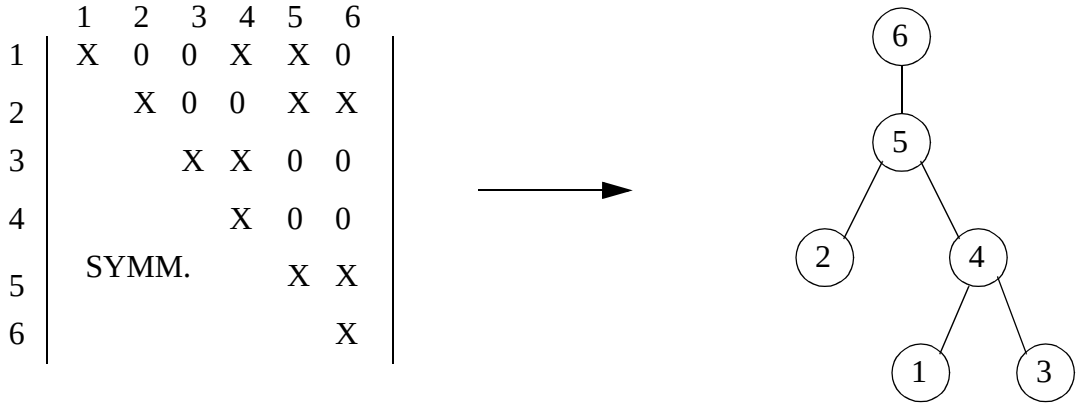
2.1.1 Step B1: selecting the best renumbering scheme

Many investigators have studied the effects of reordering on the performance of the incomplete factorization. For positive definite systems it has been shown that the renumbering schemes, such as Reverse Cuthill-McKee (RCM), Minimum Degree (MMD) and its variants (GenMMD, AMDbar and METIS [18][19]), have little effect on the convergence of the iterative methods [17]. Based on our numerical experiments, we have observed that indefinite systems and complex systems are highly sensitive to reordering [3]. RCM seems to be superior for sparse weakly indefinite or positive definite systems while MMD and its variants have been found to be better choices for highly indefinite or complex systems which require high amount of fill-ins and for dense or mildly sparse matrices which require high amount of float-

point operations. In those systems, the algorithm will attemt three renumbering schemes, GenMMD, AMDbar and METIS [18][19], and choose the scheme which generates the minimum fill-ins.

2.1.2 Step B2-B6: creating multiple fronts

After an appropriate renumbering scheme is selected, an elimination tree is constructed based on the results of the symbolic factorization. The elimination tree is constructed in such a way that each parent node of the tree is ready for factorization if and only if all its children have been factorized and all corresponding contribution matrices have been computed. From the elimination tree the factorization sequence is selected to minimize the memory usage. Each node in the elimination tree is assigned the weight representing the depth of the subtree rooted at that node. Then the tree is post-reordered so that that for each node all its children are sorted in an ascending order of weight. After the reordering, the elimination tree is rebuilt by recomputing the symbolic factorization. Greater efficiency during the following numerical factorization can be achieved if the size of the nodes and the number of rows they represent are large. The nodes are collapsed to supernodes by combining the adjacent nodes if the corresponding rows have the same nonzero pattern. We will refer to those supernodes as fronts. Figure 2 shows an example of this procedure.



Original 6x6 matrix

Elimination tree

Node	Weight	Reordering
1	0	2
2	0	4
3	0	1
4	1	3
5	2	5
6	3	6

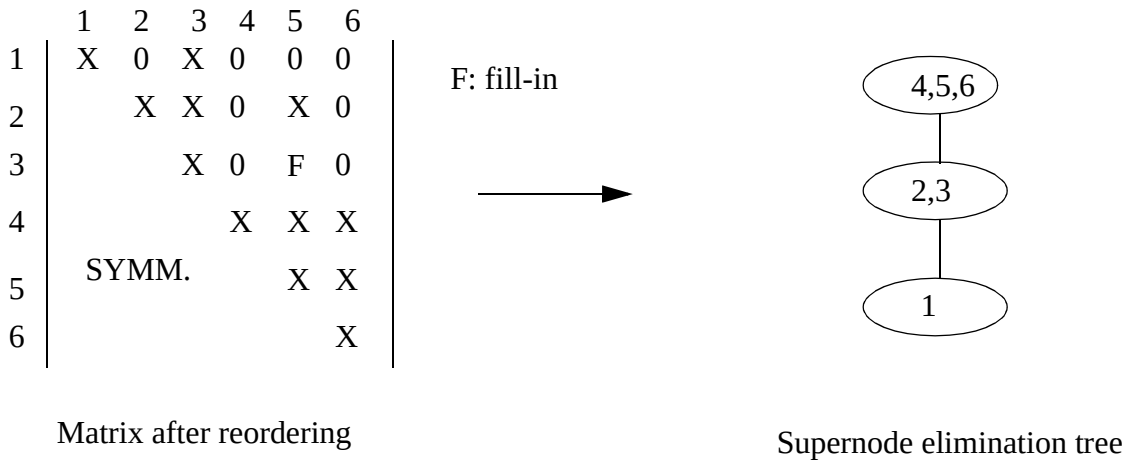


FIGURE 2. Multifrontal creation

2.1.3 Scaling

The objective of scaling is to improve the conditioning of the original matrix. For the incomplete factorization by threshold, scaling also allows to use the same drop tolerance for all rows. For positive definite systems, the magnitude of diagonal entries of the original matrix is typically used for scaling. For indefinite systems or complex systems, on the other hand, an iterative approach (see *Algorithm 6*) is proposed to compute the diagonal scaling matrix P which makes the ∞-norm of each row of the scaled system $P^{-1}KP^{-1}$ to be bounded by $[1-\mu, 1+\mu]$ where μ is a small positive number. The iterative process is required to preserve the symmetry of the preconditioner.

Algorithm (6): Scaling for Indefinite Systems

Set $P = I_n$, *flag = 0, V=0.*
Do while flag = 0
  flag = 1; $T = I_n$, *V=0*
  Loop over all nonzeros in matrix K
    For each nonzero K_{ij}
      if $|K_{ij}/P_{ii}/P_{jj}| > V_i$, *then* $V_i = |K_{ij}/P_{ii}/P_{jj}|$.
      if $\sqrt{|K_{ij}/P_{ii}/P_{jj}|} > T_{ii}$, *then* $T_{ii} = \sqrt{|K_{ij}/P_{ii}/P_{jj}|}$
      if $\sqrt{|K_{ij}/P_{ii}/P_{jj}|} > T_{jj}$, *then* $T_{jj} = \sqrt{|K_{ij}/P_{ii}/P_{jj}|}$
  End of loop
  Loop over all elements of the vector V
    if $V_i > 1+\mu$ *or* $V_i < 1-\mu$, *then flag =0.*
  End of loop
  P=PT
Enddo

2.1.4 The collection-assembly stack

During the analyze phase, a supernode elimination tree is constructed for the following numerical factorization. A two-end collection-assembly stack is allocated to collect children's contribution matrices and assemble them to their parent's frontal matrix and contribution matrix. Figure 3 shows an example of the frontal matrix and the contribution matrix of a particular front. Since for each front its contribution matrix is assembled once, one stack is sufficient for collection and assembly of the contribution matrices. To avoid the moving of data within the memory, a two-end stack is designed to push or pop the contribution matrices at both ends. In other words, the end is switched when the factorization process moves one level higher within the elimination tree. This process is illustrated in Figure 4.

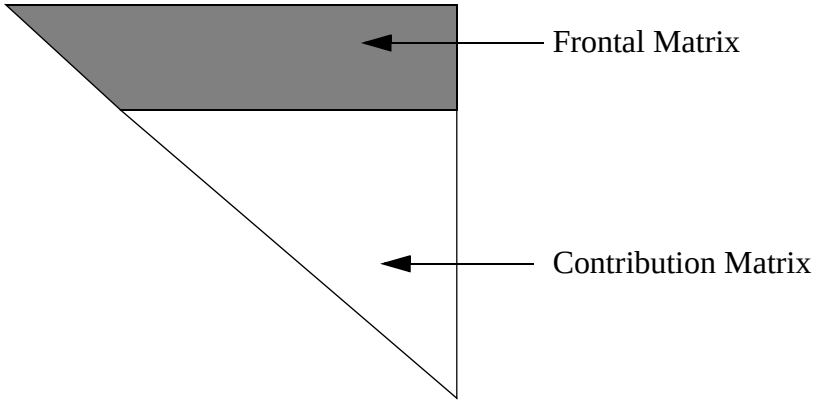


FIGURE 3. Frontal matrix and contribution matrix

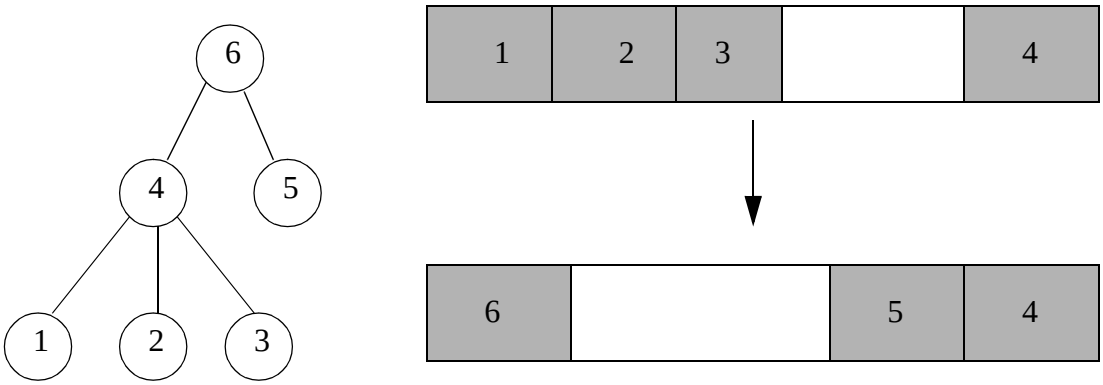


FIGURE 4. Elimination tree and two-end collection-assembly stack

2.1.5 Step C3: Numerical Incomplete Factorization

Before updating and factorizing each front, it should be filled in with entries from the scaled original matrix and the contribution matrices of its children, using the elimination tree and the

two-end collection-assembly stack. The incomplete Cholesky factorization scheme can be constructed by utilizing the dropping scheme within the complete Cholesky factorization process. Algorithm 7 gives the incomplete Cholesky factorization using regular threshold dropping schemes. Note that with *if*-statements in the inner most loops for dropping small entries, it is not possible to exploit dense matrix operations such as the level 2 and level 3 primitives of BLAS. In the following algorithms, we denote $|x|$ as the magnitude of x.

Algorithm (7): *m*-step Incomplete Cholesky Factorization

1. Update and factorize the frontal matrix

for $i = 1 \ldots m$ *Do:*
  $S = 0$
  for $k = 1 \ldots i\text{-}1$ *Do:*
    if $a_{ki} \neq 0$
      $S = S + a_{ki} \times a_{ki}$
    endif
  endfor
  $a_{ii} = \sqrt{a_{ii} - S}$
  for $j = i+1 \ldots n$ *Do*
    $S = 0$
    for $k = 1 \ldots i\text{-}1$ *Do:*
      if $a_{ki} \neq 0$ *and* $a_{kj} \neq 0$
        $S = S + a_{kj} \times a_{ki}$
      endif
    endfor
    $a_{ij} = (a_{ij} - S) / a_{ii}$
  endfor
  apply the dropping rule to a_{ij} *where* $i + 1 \leq j \leq n$
    Scheme (i): The largest lfil nonzeros of a_{ij} *are kept*
    Scheme (ii): If $|a_{ij}| \leq droptol$, $a_{ij} = 0$
endfor

2. Update the contribution matrix

for $i = m+1 \ldots n$ *Do:*
  $S = 0$
  for $j = 1 \ldots n$ *Do:*
    for $k = 1 \ldots m$ *Do:*

      if $a_{ki} \neq 0$ *and* $a_{kj} \neq 0$
        $S = S + a_{kj} \times a_{ki}$
      endif

endfor
endfor
$a_{ij} = a_{ij} - S$
endfor

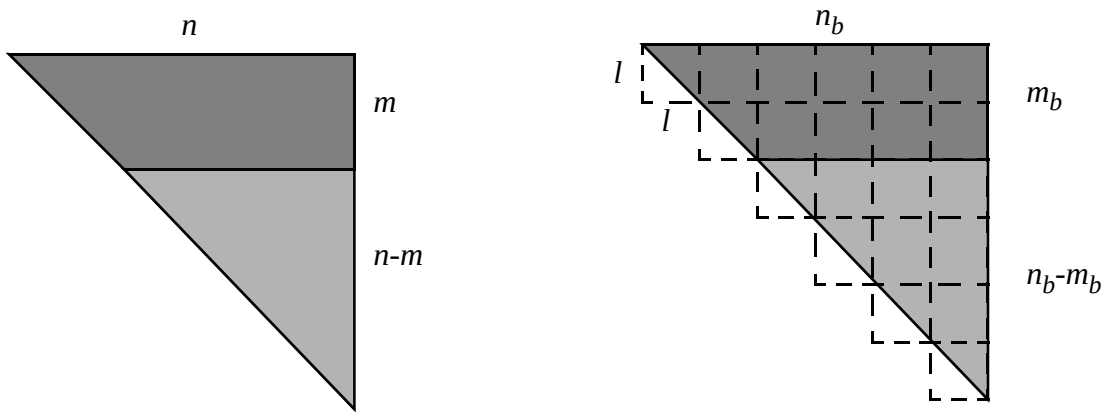


FIGURE 5. Block structures of the frontal matrix

In order to effectively utilize the level 2 and level 3 primitives of BLAS in the inner most loops, we first divide the whole triangular matrix into small block matrices. To simplify the description, we assume all block matrices have the same dimensions, *l* by *l*, as shown in Figure 5. For a general case, we need to treat the block matrices on the boundary separately since they may have different dimensions.

As shown in Algorithm 2, it is the dropping scheme based on a single entry that precludes the utilization of the dense matrix operations. Apparently, we need to develop a new dropping scheme which drops a group of entries at a time. In the following, we will introduce three types of block dropping schemes: block-based threshold dropping scheme, column-based threshold dropping scheme, and segment-based threshold dropping scheme.

Algorithm (8): Block-based threshold dropping scheme:

For matrix $A \in C^{l \times l}$
if all entries of A satisfies $|a_{ij}| \le droptol$ *then*
drop A
else
keep A
endif

Algorithm (9): Column-based threshold dropping scheme:

For matrix $A \in C^{l \times l}$
loop over all columns of A, $A_{(j)}$

if all entries of column $A_{(j)}$ satisfies $|a_{ij}| \le droptol$ then
drop column $A_{(j)}$
else
keep column $A_{(j)}$
endif
endloop
if all columns of A are dropped then
drop A
endif

Algorithm (10): Segment-based threshold dropping scheme:

As shown in Figure 6, all columns of matrix *A* are divided into equal size segments and a binary number is assigned to each column according to its dropping pattern.

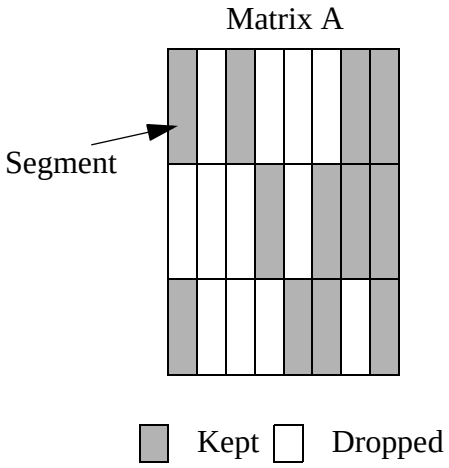
For matrix $A \in C^{l \times l}$
loop over all columns of A, $A_{(j)}$
set binary number $B_{(j)}$ to be zero
loop over all segments, $A_{(k)(j)}$, of column $A_{(j)}$
if all entries of segment $A_{(k)(j)}$ satisfies $|a_{ij}| \le droptol$ then
drop segment $A_{(i)(j)}$
set the last bit of $B_{(j)}$ to be 0 and left-shift $B_{(j)}$ one bit
else
keep column $A_{(j)}$
set the last bit of $B_{(j)}$ to be 1 and left-shift $B_{(j)}$ one bit
endif
endloop
if all segments of $A_{(j)}$ are dropped then
drop $A_{(j)}$
endif
endloop
if all columns of A are dropped then
drop A
endif

With a binary number assigned to each column, it is possible to compute the inner product of two columns using BLAS dense vector inner products in the inner most loop.

In order to compute the inner product of two columns, $A_{(j)}$ and $A_{(k)}$, we first compute the logical AND operation of the two binary numbers, $B_{(j)}$ and $B_{(k)}$, assigned to the two columns.

$C_{(j)(k)}$ = AND($B_{(j)}$, $B_{(k)}$)
temp = 0
Loop over all bits of $C_{(j)(k)}$

Find all blocks of bit 1, the consecutive bits of 1, within $C_{(j)(k)}$.
For each block, compute the inner-product of the corresponding segments of $A_{(j)}$ and $A_{(k)}$.
Add the result to temp.
endloop
temp is the final result of the inner product.



Column	Assigned Binary Number
1	100
2	000
3	100
4	010
5	001
6	011
7	110
8	111

FIGURE 6. Segment-based threshold dropping scheme

As shown in Algorithm 3, for the block-based threshold dropping scheme, it is possible to take full advantage of the dense matrix operations by using dense matrix multiplications in the inner most loops. However, since the basic unit of this dropping scheme is a block, it prevents from dropping more nonzeros and makes the incomplete factorization process expensive.

The column-based dropping scheme is a compromise between the efficient utilization of dense matrix operations and the accuracy of the preconditioner. The basic unit of this dropping scheme is a column, which is much smaller than the block. In the inner most loops, dense vector inner products are used instead of the dense matrix multiplications. In other words, we settle for less than optimal use of BLAS in the interest of keeping the memory usage and the total number of float-point operations low.

The segment-based dropping scheme is more aggressive than the column-based dropping scheme. It intends to drop more nonzeros by using a segment, which is smaller than the column, as the basic unit for dropping. This scheme is very useful if the size of the column is very large.

Algorithm (11): m-step Block Incomplete Cholesky Factorization

1. Update and factorize the frontal matrix

for $i = 1... m_b$ Do:

$S = 0$
for $k = 1 \ldots i-1$ *Do:*
Block-based dropping scheme:
if A_{ki} *is kept*
$S = S + A_{ki}^T A_{ki}$
endif
Column-based dropping scheme:
if A_{ki} *is kept*
for $p = 1 \ldots l$
for $q = p \ldots l$
if $(A_{ki})_{(p)}$ *is kept and* $(A_{ki})_{(q)}$ *is kept*
$S_{pq} = S_{pq} + (A_{ki})_{(p)}^T (A_{ki})_{(q)}$
endif
endfor
endfor
endif
Segment-based dropping scheme:
if A_{ki} *is kept*
for $p = 1 \ldots l$
for $q = p \ldots l$
if $(A_{ki})_{(p)}$ *is kept and* $(A_{ki})_{(q)}$ *is kept*
Compute the inner product of $(A_{ki})_{(p)}$ *and* $(A_{ki})_{(q)}$
using Algorithm 10.
Add the result to S_{pq}
endif
endfor
endfor
endif
endfor
$A_{ii} = A_{ii} - S$
Compute l-step Cholesky factorization of A_{ii} *and replace the corresponding first l rows of* A_{ii} *with the factorization results*
for $j = i+1 \ldots n_b$ *Do*
$S = 0$
for $k = 1 \ldots i-1$ *Do:*
Block-based dropping scheme:
if A_{ki} *is kept and* A_{kj} *is kept*
$S = S + A_{ki}^T A_{kj}$
endif

Column-based dropping scheme:

if A_{ki} *is kept and* A_{kj} *is kept*

for $p = 1 \ldots l$

for $q = 1 \ldots l$

if $(A_{ki})_{(p)}$ *is kept and* $(A_{kj})_{(q)}$ *is kept*

$$S_{pq} = S_{pq} + (A_{ki})_{(p)}^{T}(A_{kj})_{(q)}$$

endif

endfor

endfor

endif

Segment-based dropping scheme:

if A_{ki} *is kept and* A_{kj} *is kept*

for $p = 1 \ldots l$

for $q = 1 \ldots l$

if $(A_{ki})_{(p)}$ *is kept and* $(A_{kj})_{(q)}$ *is kept*

Compute the inner product of $(A_{ki})_{(p)}$ *and* $(A_{ki})_{(q)}$

using Algorithm 10.

Add the result to S_{pq}

endif

endfor

endfor

endif

endfor

$A_{ij} = A_{ij} - S$

$A_{ij} = (A_{ii})^{-1} A_{ij}$

endfor

apply the dropping rule to A_{ij} *where* $i + 1 \le j \le n$

endfor

2. Update the contribution matrix

for $i = m_b + 1 \ldots n_b$ *Do:*

for $j = i \ldots n_b$ *Do*

$S = 0$

for $k = i \ldots m_b$ *Do:*

Block-based dropping scheme:

if A_{ki} *is kept and* A_{kj} *is kept*

$$S = S + A_{ki}^{T} A_{kj}$$

endif

Column-based dropping scheme:

if A_{ki} *is kept and* A_{kj} *is kept*
for $p = 1 \ldots l$
for $q = 1 \ldots l$
if $(A_{ki})_{(p)}$ *is kept and* $(A_{kj})_{(q)}$ *is kept*
$S_{pq} = S_{pq} + (A_{ki})^T_{(p)}(A_{kj})_{(q)}$
endif
endfor
endfor
endif

Segment-based dropping scheme:

if A_{ki} *is kept and* A_{kj} *is kept*
for $p = 1 \ldots l$
for $q = 1 \ldots l$
if $(A_{ki})_{(p)}$ *is kept and* $(A_{kj})_{(q)}$ *is kept*
Compute the inner product of $(A_{ki})_{(p)}$ *and* $(A_{ki})_{(q)}$
using Algorithm 10.
Add the result to S_{pq}
endif
endfor
endfor
endif
endfor
endfor
$A_{ij} = A_{ij} - S$
endfor

2.2 Multifrontal LDL^T incomplete factorization

The multifrontal LDL^T incomplete factorization is needed for solving indefinite and close-to-singular positive definite systems. The multifrontal LDL^T incomplete factorization uses the same data structures, such as fronts, elimination trees, the two-end stacks *etc*. Since it is unnecessary to keep ones on the diagonal of L^T, the diagonals of all front matrices can be used to keep the diagonal, D, and the off-diagonals to keep the upper triangular matrix, L^T.

The two major differences between the multifrontal Cholesky incomplete factorization and the multifrontal LDL^T incomplete factorization are the factorization of the frontal matrices and the updating process of the contribution matrices. The following is the algorithm for the multifrontal LDL^T incomplete factorization with the block-based dropping scheme. For other dropping schemes, the algorithms are similar. In the following algorithm, the diagonal of the block matrix A_{kk} is denoted as D_k.

Algorithm (12): m-step Block Incomplete LDL^T Factorization

1. Update and factorize the front matrix

for $i = 1 \ldots m_b$ *Do:*
  $S = 0$
  for $k = 1 \ldots i-1$ *Do:*
    Block-based dropping scheme:
      if A_{ki} *is kept*
        $S = S + A_{ki}^T D_k A_{ki}$
      endif
  endfor
  $A_{ii} = A_{ii} - S$
  Compute l-step LDL^T *factorization of* A_{ii} *and replace the corresponding first l rows of* A_{ii} *with the factorization results*
  for $j = i+1 \ldots n_b$ *Do*
    $S = 0$
    for $k = 1 \ldots i-1$ *Do:*
      Block-based dropping scheme:
        if A_{ki} *is kept and* A_{kj} *is kept*
          $S = S + A_{ki}^T D_k A_{kj}$
        endif
    endfor
    $A_{ij} = A_{ij} - S$
    $A_{ij} = (A_{ii})^{-1} A_{ij}$
    endfor
  apply the dropping rule to A_{ij} *where* $i + 1 \le j \le n$
endfor

2. Update the contribution matrix

for $i = m_b+1 \ldots n_b$ *Do:*
  for $j = i \ldots n_b$ *Do*
    $S = 0$
    for $k = i \ldots m_b$ *Do:*
      Block-based dropping scheme:
        if A_{ki} *is kept and* A_{kj} *is kept*
          $S = S + A_{ki}^T D_k A_{kj}$
        endif
    endfor
  endfor
  $A_{ij} = A_{ij} - S$

endfor

2.3 Adaptive estimation of parameters by global-basis two-level method

The multifrontal incomplete factorization can be used as a general-purpose single-level precondtioner for Krylov subspace methods such as CG, GMRES, QMR *etc*. However, its robustness and efficiency can be further improved if it is incorporated into the framework of the global-basis two-level method as a smoother[2][3].

The global-basis two-level method is a methodology designated to enhance the performance of a single- or multi-level iterative methods by constructing an additional coarse level aimed at eliminating the components of error which have not been removed by the iteration matrix R of a single- or multi-level iterative method of choice. The matrix R is given as

$$R = I_N - M^{-1}A \in C^{N \times N} \tag{4}$$

where $M \in C^{N \times N}$ denotes the smoothing preconditioner.

Convergence studies in [2] showed that for weakly and highly indefinite systems a successful iterative method should possess a coarse model able to accurately reproduce the space of algebraically “smooth” modes which span the spectrum of highest eigenmodes of the smoothing iteration matrix R.

The adaptive global-basis method starts by constructing a *tentative* multifrontal incomplete factorization as a smoother. The *tentative* multifrontal incomplete factorization employs a small number of fill-ins in attempt to estimate the maximum eigenvalue of the smoothing iteration matrix R. If the smoother is found to be satisfactory, the coarse model is formed to include the algebraically “smooth” modes. If the number of required “algebraically” smooth modes exceeds the user prescribed limit, the quality of the smoother is improved. On the other hand, if the *tentative* incomplete factorization smoother is found to be not satisfactory, i.e., a very large coarse model is required, the new tolerance setting is heuristically determined on the basis of the magnitude of the maximum eigenvalue of R.

Numerical experiments have indicated that it is more efficient to improve the quality of the multifrontal incomple factor than to construct the coarse level. The reason is that a multifrontal incomplete factorization with a large dropping tolerance converges very slow. As the number of fill-ins is increased the iterative method converges very fast with the spectrum radius of the smoothing iteration matrix R rapidly decreasing.

As an illustration, we consider a fluid-structure interaction problem [21] with 1,436 degrees-of-freedom. Figure 7 and Figure 8 show the number of the algebraically “smooth” modes and the iterations versus the quality of the smoother for the multifrontal incomplete factorization smoother (*ICMultifrontal*) and the scalar version of incomplete factorization (*ICScalar*) [3], respectively. The quality of the smoother is measured in terms of the percentage of total nonzeros it keeps. Figure 7 and Figure 8 also show the computational cost of the incomplete factorization and the total solver CPU time, respectively. It can be seen that the number of the algebraically “smooth” modes rapidly decreases as the number of fill-ins is increased. On the

other hand, for the same number of nonzeros the scalar version of the incomplete factorization has fewer algebraically "smooth" modes and results in less number of iterations, but the CPU time is significantly higher.

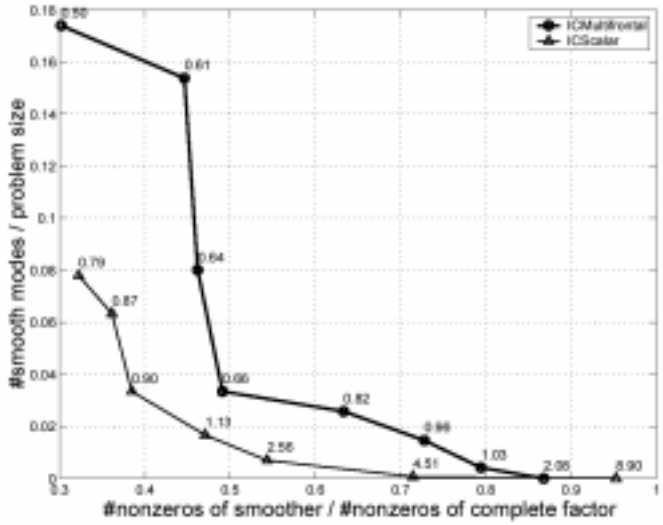


FIGURE 7. Percentage of "smooth" modes vs. percentage of the nonzeros in the incomplete factor

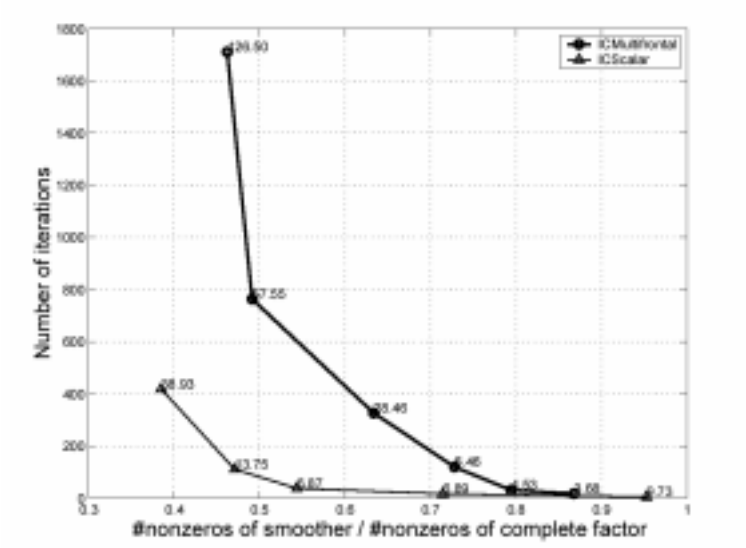


FIGURE 8. Number of iterations vs. percentage of the nonzeros in the incomplete factor

Although it may not be necessary to construct the global-basis coarse model for the multifrontal incomplete factorization smoother, we exploit its framework to heuristically determine the algorithmic parameters on the basis of the magnitude of the maximum eigenvalue of the smoothing iteration matrix, $|\lambda_{max}|$. The estimation of the maximum eigenvalue normally only requires a few steps of the Lanczos procedure (typically in the range from five to ten). Besides

the dropping tolerance, the other important parameter for the multifrontal incomplete factorization is the block size. The optimal block size depends not only on the type of the hardware but also on the properties of the matrices. For example, if a matrix is positive definite and well-conditioned, the multifronal incomplete factorization with a small block size is more efficient than with a large block size since the small block size allows to drop more nonzeros without significantly deteriorating the quality of the incomplete factorization. If the tentative smoother is satisfactory and the incomplete factorization keeps only a small percentage of nonzeros, 20% - 30%, the algorithm attempts to reconstruct the incomplete factorization with a smaller block size and to estimate the maximum eigenvalue of the new smoothing iteration matrix and the sparsity of the new incomplete factorization. Schematics of the algorithm is illustrated in Figure 9.

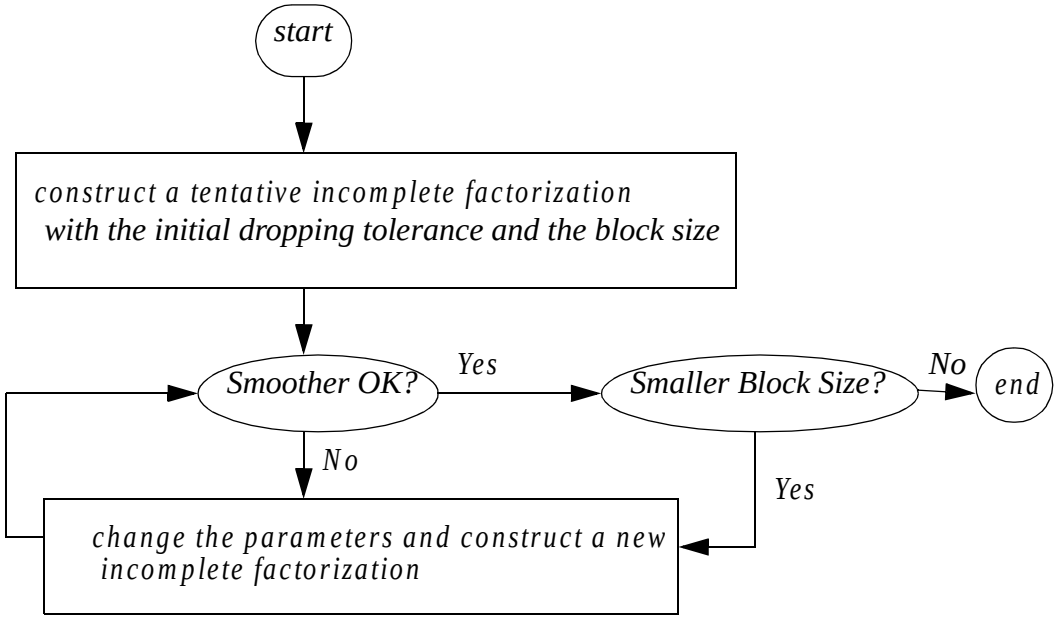


FIGURE 9. Flow chart for adaptive parameter selection

2.4 Out-of-core version of the multifrontal incomplete factorization

Any solver should have both in-core and out-of-core capabilities since it is not usually possible to keep the entire matrix and factor in core especially for large and dense matrices. To keep the memory usage low, out-of-core solution methods usually utilize the secondary memory. Only part of the matrix and factor are loaded into memory from the disk files when necessary.

For the multifrontal Cholesky/LDL^T incomplete factorization, we can store the incomplete factor of each front into the disk files after its factorization. The data kept in the disk files includes the nonzeros and their positions in the compressed row format [1]. The only data which should be kept in the memory are the positions of the factors in the disk files which

allow the solver to quickly locate and load the factors during the process of the preconditioning. The same method can also be applied for storing the original matrix.

3.0 Numerical Examples and Discussion

3.1 Helmholtz equation on bounded domains

Consider Helmholtz's equation in the region enclosed between two concentric cubes of length $2(\Gamma_0)$ and $6(\Gamma_1)$. The strong form of the governing equations is given as

$$\nabla^2 u(x) + k^2 u(x) = 0 \qquad x \in \Omega \tag{5}$$

$$\frac{\partial u}{\partial n} = \frac{\partial}{\partial r}(e^{ikr}/r)\Big|_{\Gamma_0} \qquad x \in \Gamma_0 \subset \partial\Omega \tag{6}$$

$$\frac{\partial u}{\partial n} = \frac{\partial}{\partial r}(e^{ikr}/r)\Big|_{\Gamma_1} \qquad x \in \Gamma_1 \subset \partial\Omega \tag{7}$$

where n is a coordinate in a direction normal to Γ_0 and Γ_1; $\Gamma_0 \cup \Gamma_1 = \partial\Omega$ and $\Gamma_0 \cup \Gamma_1 = 0$; r is distance from the center of the cube. Equations (5)-(7) describe the acoustic pressure u of a wave in a transmitting medium.

Due to symmetry, only one-eighth of the domain is discretized. Three meshes consisting of 3,072, 23,925 and 156,009 *p*-method tetrahedral fluid elements have been considered. The basic information of the three meshes is listed in Table 2 and the coarsest discretization is shown in Figure 10. We define the sparsity of the system, as $\omega = nz(L)/(0.5n^2)$, where *nz(L)* is the number of nonzeros in the factor L, and n is the size of the system.

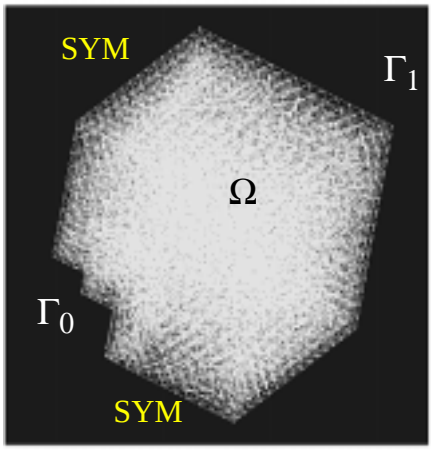


FIGURE 10. : Typical finite element mesh and boundary conditions

	Mesh #1	Mesh #2	Mesh #3
# of Elements	156,009	23,925	3,072
Average Element Size h	0.3094	0.3572	0.7733
Order of p	2	3	5
Problem Size	218,994	114,886	69,151
# of Nonzeros (Original System)	2,943,139	3,570,150	2,616,864
# of Nonzeros (Complete Factor)	149,478,256	59,752,343	28,391,676
System Sparsity ω	0.00623	0.00905	0.0119

TABLE 2. Basis information of three meshes

The resulting discrete linear system of equations, $Kx = f$, is complex symmetric. We consider three approaches for solving the discrete linear system of equations:

(i) Global-basis two-level method coupled with *QMR* method (*GlobalBasis + QMR*) [3],

(ii) Multifrontal Cholesky incomplete factorization preconditioner coupled with *QMR* method (*ICMultifrontal + QMR*),

(iii) Multifrontal Direct solver (Cholesky factorization) (*Multifrontal Direct*).

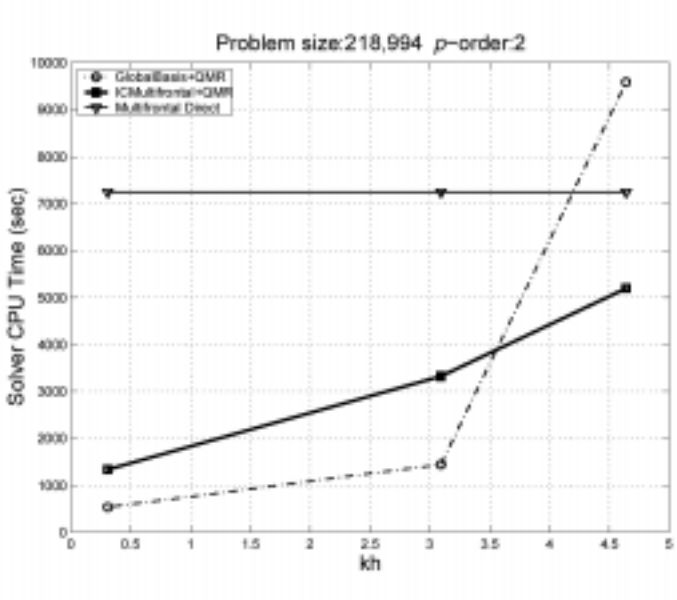


FIGURE 11. : CPU Time versus *kh* for discrete Helmholtz linear systems with 218,994 equations

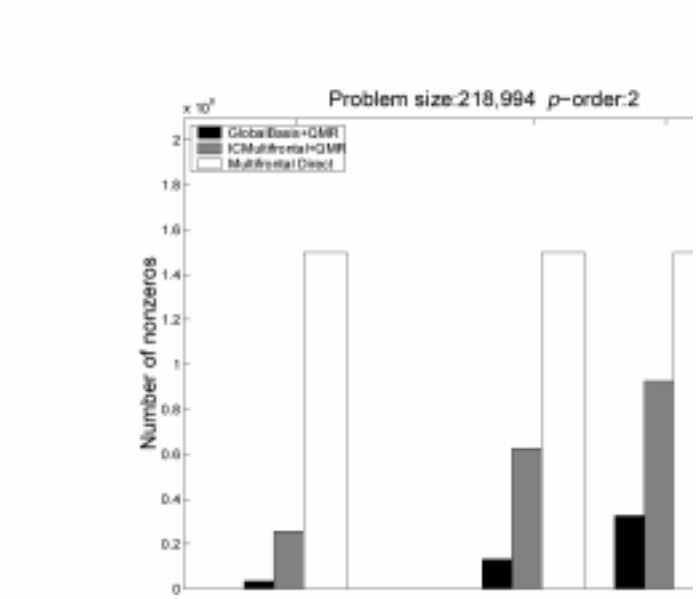


FIGURE 12. : # of nonzeros versus *kh* for discrete Helmholtz linear systems with 218,994 equations

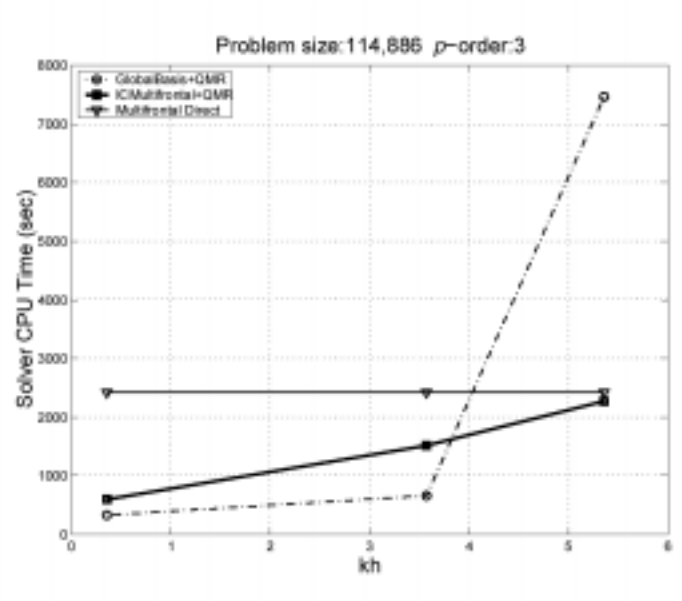


FIGURE 13. : CPU Time versus *kh* for discrete Helmholtz linear system of 114,886 equations

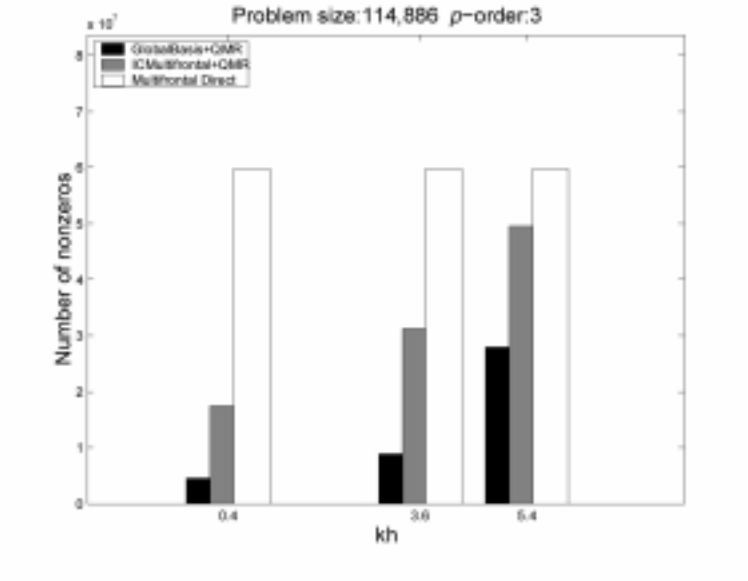


FIGURE 14. : # of nonzeros versus *kh* for discrete Helmholtz linear systems with 114,886 equations

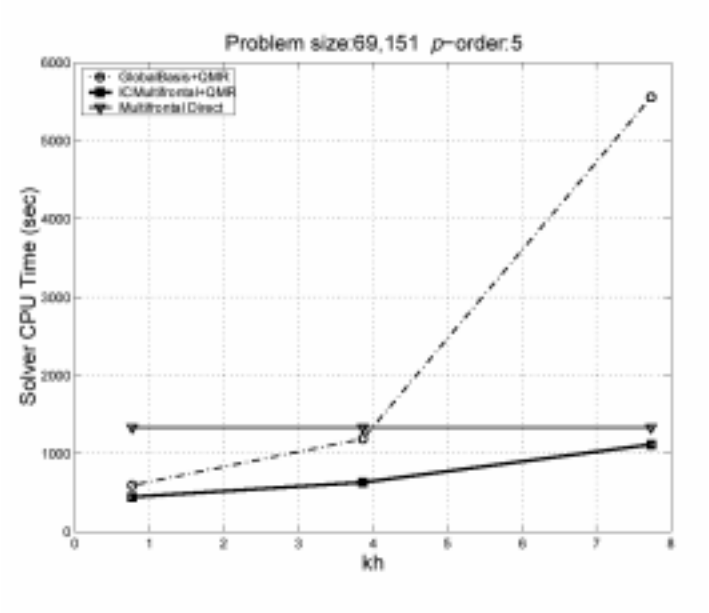


FIGURE 15. CPU Time versus *kh* for discrete Helmholtz linear system of 69,151 equations

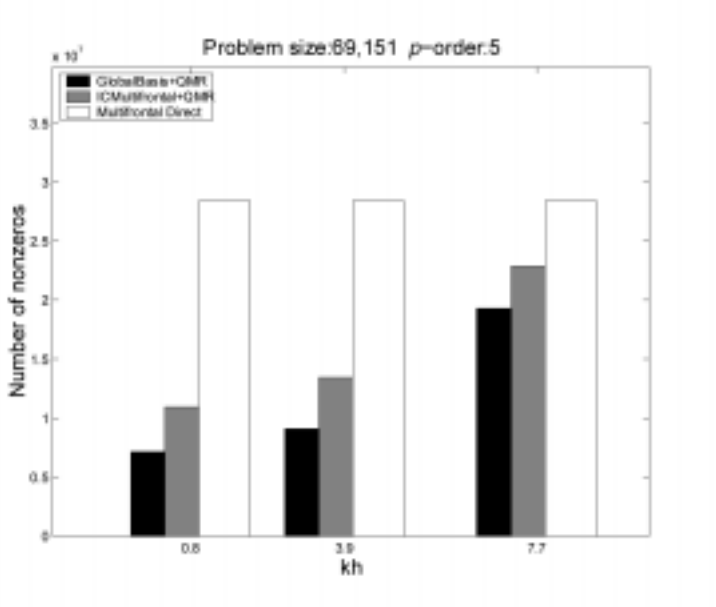


FIGURE 16. # of nonzeros versus *kh* for discrete Helmholtz linear systems with 69,151 equations

The Global-basis scheme uses the scalar version of the incomplete LDL^T factorization preconditioner as a smoother. We used $\mu = 0.1$ for adaptive scaling and $\alpha = 0.001$, $m = 4$ for pivoting. In the Global-basis algorithm, the *tentative incomplete factorization* smoother is considered satisfactory if $|\lambda_{max}| < 10$. The initial dropping tolerance for the *enhanced incomplete factorization* is determined as $droptol = min(1/(\sqrt{|\lambda_{max}|}), 0.01)$ and is reduced by factor of 5 for subsequent incomplete factorizations [3].

In the multifrontal incomplete scheme, the initial dropping tolerance for the multifrontal incomplete factorization is set as $droptol = 0.005$ and the new dropping tolerance is determined as

$$droptol_{new} = min(1/(\sqrt{|\lambda_{max}|}), droptol/5.0) \tag{8}$$

for subsequent incomplete factorizations if $|\lambda_{max}| > 1$. The initial block size is set as 32 and is reduced by factor of 2 for subsequent incomplete factorizations if necessary. We only consider the column-based dropping scheme.

Figure 11, Figure 13 and Figure 15 show the CPU time versus the product of the average element size and the wave number, *kh*, for the three meshes considered, respectively. The numbers of nonzeros in the scalar version of incomplete LDL^T factorization (Scheme i), the multifrontal incomplete factorization (Scheme ii) and the complete factorization (Scheme iii) are shown in Figure 12, Figure 14 and Figure 16, respectively.

It can be seen that for the densest system, Mesh # 3, the multifrontal incomplete scheme is the best solver over the whole spectrum of *kh*. The multifrontal incomplete scheme is also the best

solver for Mesh #1 and Mesh #2 when the real part of the system is highly indefinite ($4<kh<10$). The global-basis scheme is best suited for very sparse systems, Mesh #1 and Mesh #2 and when the real part of the system is positive definite or weakly indefinite ($kh<4$). However, for systems which require dense preconditioners, the global-basis scheme is significantly slower than the multifrontal direct solver because its algorithm and data structures do not take advantage of the fast dense matrix operations such as the level 2 and level 3 primitives of BLAS.

The multifrontal incomplete scheme has been found to be faster than of the direct solver although the CPU times are very close when the preconditioners are dense. For a given system, the incomplete multifrontal scheme keeps more nonzeros than that of the global-basis scheme. However, since it employs fast dense matrix operations in the inner most loops, it is faster than the global-basis scheme with the scalar version of the incomplete factorization even though it drops 20%-30% of total nonzeros as shown in Figure 12, Figure 14 and Figure 16.

3.2 Fluid-structure interaction problem

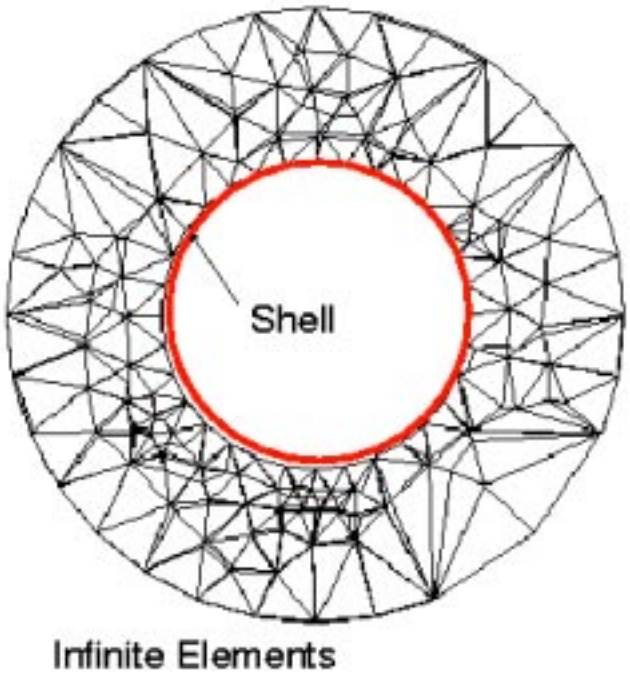


FIGURE 17. Typical Finite element mesh

	GlobalBasis + QMR	ICMultifrontal + QMR		Multifrontal Direct	
	in-core	**in-core**	**out-of-core**	**in-core**	**out-of-core**
# Nonzeros of factor	5,478,751	11,601,625		15,416,567	
Solver Total (sec.)	1516	273	278	346	349
Incomplete Factorization	283	225	227	335	338
GlobalBasis Construction	130				
Iterative Process	1103	48	51		
Forward/ backward Substitutions				11	11
# Iterations	72	8	8		

TABLE 3. Split-up Times

Figure 17 illustrates the schematics of a fluid-structure interaction problem [21]. The resulting system of equations is complex symmetric. The size of the system is 41,592 and the number of nonzeros is 3,140,344.

The results of the three solvers described in Section 3.1 are listed in Table 3 . It can be seen that the multifrontal incomplete scheme is the best solver although its multifrontal incomplete factorization only drops 25% of total nonzeros. On the other hand, the scalar version of the incomplete factorization in the global-basis scheme keeps 30% of nonzeros but the CPU time is higher than that of the multifrontal incomplete scheme. We were unable to drop more terms since the fluid-structure interaction problem is exteremely bad conditioned. Also it can be seen from Table 3 that the I/O operations are very fast making the computational costs of the in-core and the out-of-core methods to be very close.

3.3 Shear banding problem

We considered a linearized shear banding problem, illustrated in Figure 18. The cube is discretized with $32 \times 32 \times 32$ 8-node hexahedral elements totaling to 107,787 degrees-of-freedom. We assume that a shear band (softening zone) develops on the diagonal plane of two layers of elements [24]. We considered the spectrum of ratios between the stiffness inside and outside the shear band, E_{band}/E, in the range of 0.3 and -0.9.

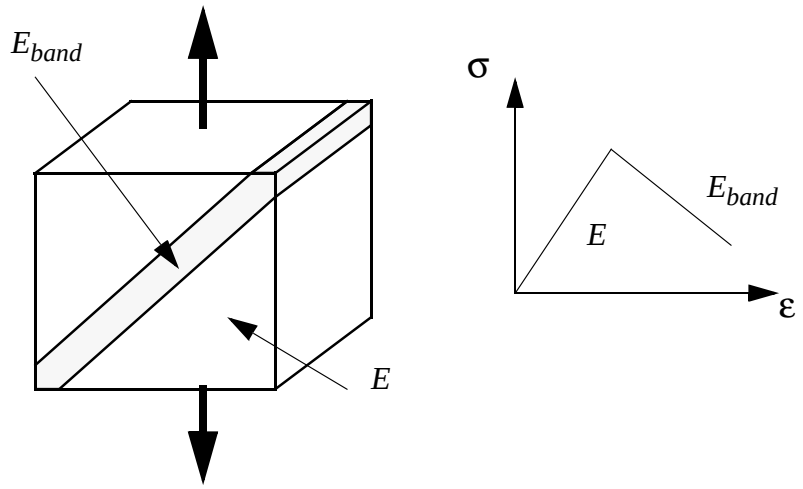


FIGURE 18. : Model for shear banding problem

We consider three approaches for the solution of discrete linear system of equations:

(i) Global-basis two-level method coupled with *QMR* method (*GlobalBasis + QMR*),

(ii) Multifrontal LDL^T incomplete factorization preconditioner coupled with *QMR* method (*ICMultifrontal + QMR*),

(iii) Multifrontal Direct solver (LDL^T factorization) (*Multifrontal Direct*).

Figure 19 and Figure 20 show the CPU time and the number of nonzeros of the preconditoners versus the measure of indefiniteness (E_{band}/E). It can been seen that the multifrontal incomplete scheme has the best overall performance among the three approaches. The behavior of the global-basis scheme is similar to that in the Helmholtz equations. It is very fast when the system is positive definite or weakly indefinite, $E_{band}/E > 0.1$, but is very slow for highly indefinite systems, $-0.9 \le E_{band}/E \le -0.1$.

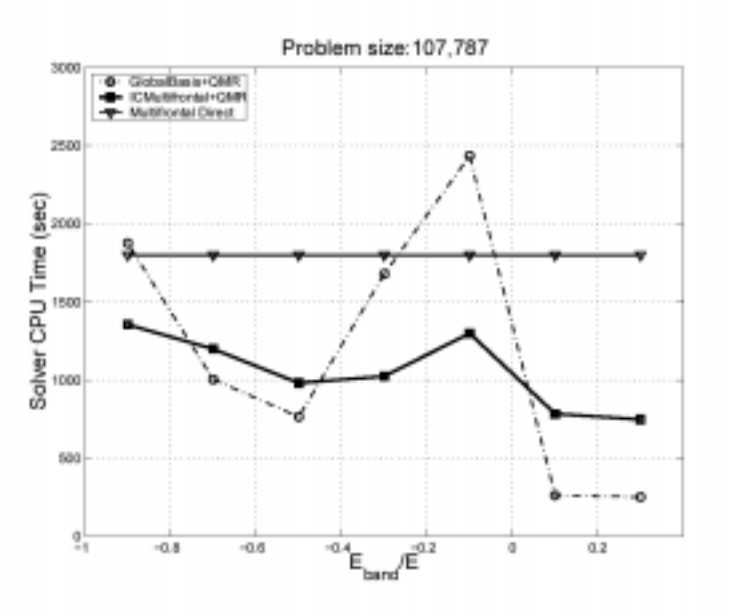


FIGURE 19. : CPU Time vs. E_{band}/E for shear banding problem with 107,787 equations

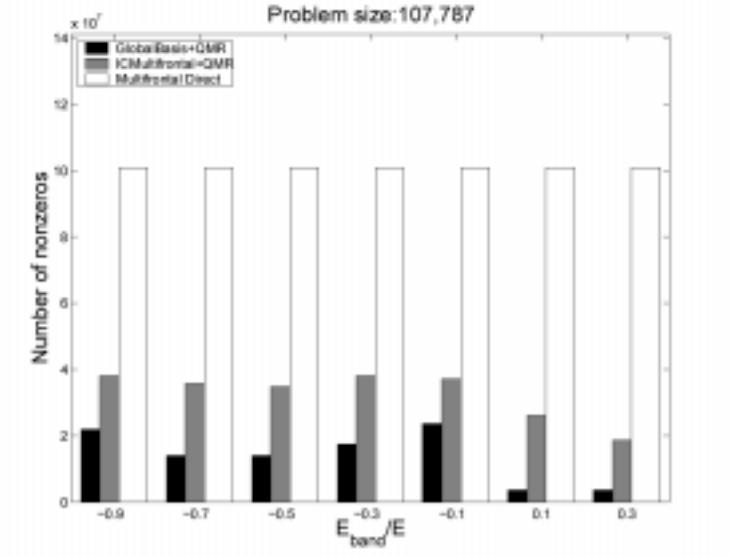


FIGURE 20. : # nonzeros vs. E_{band}/E for shear banding problem with 107,787 equations

4.0 Conclusions

This paper addresses two important issues concerning the incomplete factorization preconditioner: improving the memory-system performance and adaptive selection of algorithmic parameters. A multifrontal incomplete factorization preconditioner is developed for indefinite

and complex symmetric systems. The precondtioner keeps the dense matrix kernels in the innermost loops of its computation in order to achieve high computational efficiency. An adaptive scheme based on the global-basis method has been utilized to adaptively select the algorithmic parameters for the preconditioner. Numerical experiments conducted on the Helmholtz equations, the fluid-structure interaction problem and the shear banding problem affirm the potential of the method.

Further studies will be conducted to incorporate pivot selection into the analyze and factorize phases, to further improve the adaptive parameter selection and to extend the application of the preconditioner to nonsymmetric indefinite and complex systems.

References

1 Y. Saad, “Iterative Methods for Sparse Linear Systems,” PWS Publishing Co., Boston, 1996.

2 J. Fish and Y. Qu, “Global Basis Two-Level Method For Indefinite Systems. Part 1: Convergence studies,” *International Journal for Numerical Methods in Engineering*, Vol. 49, pp. 439-460, (2000).

3 Y. Qu and J. Fish, “Global Basis Two-Level Method For Indefinite Systems. Part 2: Computational Issues” *International Journal for Numerical Methods in Engineering*, Vol 49, pp. 461-478, (2000).

4 R. W. Freund and N. M. Nachtigal, “Software for Simplified Lanczos and QMR Algorithms,” *Applied Numerical Mathematics*, Vol. 19, pp. 319-341, (1995).

5 J. Meijerink and H. van der Vorst, “An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix,” *Math. Comput.* 31, 137 (Jan.), pp. 148-162, (1977).

6 B. Stroustrup, “The C++ Programming Language, Bjarne Stroustrup,” Addison-Wesley 1991.

7 Chow, E. and Y. Saad, “Experimental Study of ILU Preconditioners for Indefinite Matrices,” *J. Comput. and Appl. Math.* Vol. 87, pp. 387-414, (1997).

8 E. Chow and M. A. Heroux, “An object-oriented framework for block preconditioning,” *ACM Trans. Math. Softw.*, 24, pp. 159-183, (1998).

9 I.D. Duff and J.K. Reid, “The Multifrontal Solution of Indefinite Sparse Symmetric Linear Equations,” *ACM Transactions on Mathematical Software*, Vol. 9, No. 3, pp. 302-325,(1983).

10 M.T. Jones and P.E. Plassmann, “An improved incomplete factorization,” *ACM Transaction on Mathematical Software*, Vol. 21, pp.5-17,(1995)

11 J. Fish and A. Suvorov, “Automated Adaptive Multilevel Solver,” *Comp. Meth. Appl. Mech. Engng.*,Vol. 149, pp. 267-287, (1997).

12 Y.E. Campbell and T.A. Davis, “Incomplete LU Factorization: A Multifrontal Approach,” Technical Report TR-95-204, Computer and Information Sciences Depart-

ment, University of Florida, (1995)

13 Timothy A. Davis, “Block matrix methods: Taking advantage of high-performance computers,” Technical Report TR-98-024, Computer and Information Sciences Department, University of Florida, (1998)

14 E. Anderson *etc*., “LAPACK Users’ Guide Third Edition”, the Society for Industrial and Applied Mathematics (1999).

15 R. E. Bank, R. K. Smith, “The Incomplete Factorization Multigraph Algorithm,” *SIAM Journal on Scientific Computing*, Vol. 20, No. 4, pp. 1349-1364, (1999).

16 Y. Saad and J. Zhang, “Diagonal Threshold Techniques in Robust Multi-Level ILU Preconditioners for General Sparse Linear Systems,” Technical Report umsi-98-7,University of Minnesota, Minneapolis, MN, 1998.

17 M. Benzi, D. B. Szyld, and A. van Duin, “Orderings for Incomplete Factorization Preconditioning of Nonsymmetric Problems,” SIAM J. Scientific Computing 20, 3, pp. 1652--1670,(1999).

18 G. Karypis and V. Kurnar, “METIS: A Software Package for Partitioning Unstructed Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0,” University of Minnesota, Minneapolis, MN, 1998.

19 J. Harding, “Harwell Subroutine Library,” B 552, AEA Technology, C Harwell, Didcot, Oxon OX11 0RA

20 S. Toledo, “Improving the Memory-System Performance of Sparse-Matrix Vector Multiplication,” *IBM Journal of Research and Development*, Vol. 41. No.6, (1997)

21 The Naval Research Laboratory (NRL), 555 Overlook Ave. S.W. Washington, DC 20375

22 A.Bayliss, C.I.Goldstein and E.Turkel, “On accuracy conditions of the numerical computation of waves,” *Journal of Computational Physics*, Vol. 59,(1985)

23 N.N.Abboud and P.M. Pinsky, “Finite element dispersion analysis of the numerical computation of waves,” *Int. J. For Numerical Methods In Engineering*, Vol. 35, pp. 1183-1218, (1992)

24 T.Belytschko, J.Fish and B. Engelmann, “A finite element with embedded localization zones,” *Comp. Meth. Appl. Mech. Engng.*, Vol.108, (1993).