ILUS: An Incomplete LU Preconditioner in Sparse Skyline Format*

Edmond Chow and Yousef Saad

Department of Computer Science, and

Minnesota Supercomputer Institute

University of Minnesota

Minneapolis, MN 55455

April 1995

Abstract

Incomplete LU factorizations are among the most effective preconditioners for solving general large, sparse linear systems arising from practical engineering problems. This paper shows how an ILU factorization may be easily computed in sparse skyline storage format, as opposed to traditional row-by-row schemes. This organization of the factorization has many advantages, including its amenability when the original matrix is in skyline format, the ability to dynamically monitor the stability of the factorization, and the fact that factorizations may be produced with symmetric structure. Numerical results are presented for Galerkin Finite Element matrices arising from the standard square lid-driven cavity problem.

Key words. incomplete LU preconditioning, skyline format, stability, approximate inverse, lid-driven cavity

1   Introduction

The cost-effectiveness of iterative methods over direct methods for solving large-scale, sparse linear systems is now commonly accepted. The challenge remains, however, in finding general-purpose preconditioners that make the methods closer to the robustness of direct methods. A key to this robustness is the flexibility of the preconditioner to adapt itself to the difficulty of the problem, either during the construction of the preconditioner, or during the iterative phase of the solution. In this paper, we attempt to extend the capability of incomplete LU factorizations, which in our experience, are among the most reliable preconditioners to date.

In the fully-coupled solution of the incompressible Navier-Stokes equations, for example, the inclusion of the continuity condition makes the linearized system matrix indefinite. In these cases, an ILU factorization may produce L and U factors such that the norm of $(LU)^{-1}$ is very large. The long recurrences associated with solving with these factors are unstable [3, 9], producing solutions with extremely large components. A sign of this severely poor preconditioning is the erratic behavior of the iterative method, for example, divergence of the iterations due to large numerical errors. As an example to illustrate the seriousness of the stability problem, we chose Example 7 from the FIDAP fluid dynamics analysis package [10]. This example models natural convection, and the fully-coupled matrix for the first step of the first nonlinear iteration has order 1633 and 46626 nonzeros. Table 1 shows that the norm bound of $(LU)^{-1}$ calculated with $\|(LU)^{-1}e\|_\infty$ (e is the vector of all ones) for a set of incomplete LU factors increases dramatically as the allowed number of nonzeros (*lfil* per L and U factor) is reduced. Another typical feature is that the norm bound decreases again for small *lfil*. The GMRES iterative procedure [24] preconditioned with these factorizations could not succeed in solving the linear system. The linear system we chose is a striking example because it can be solved without preconditioning.

lfil	10	20	30	40	50	60	70	80	90	100
$\log_{10}\|(LU)^{-1}e\|_\infty$	132	174	203	175	277	359	231	31	27	22

Table 1: Estimate of $\|(LU)^{-1}\|_\infty$ from incomplete LU factors for EX07.

It is usually possible to produce a usable ILU factorization by allowing enough fill-in. Starting with ILU(0) or IC(0), where the nonzero pattern of the factorization is the same as that of the original matrix A, fill-in may be introduced by level-of-fill [13, 17, 25] or by threshold [18]. These indeed have been very successful for many fluid flow problems, for example, see [4]. For indefinite matrices, however, techniques based solely on level-of-fill may be inappropriate, because they ignore the numerical values. Threshold methods, on the other hand, are much more expensive, and it is difficult to determine their storage

requirements beforehand. A middle-ground between these two approaches is ILUT [21], which uses a threshold for dropping fill-ins, and an additional threshold that limits the number of fill-ins per row in the factors L and U. Careful implementation of the sparse SAXPY and numerical dropping operations makes ILUT efficient. A simple variant called ILUTP performs partial pivoting by columns and is often less prone to instability for indefinite problems. Table 1 was calculated using the ILUT factors.

The drop-tolerance and how much fill-in is required to produce an accurate and/or stable factorization is difficult to predict. However, it is possible to determine in advance whether or not a factorization will fail due to instability by estimating a norm of $(LU)^{-1}$ in some way. In this paper, we show how to construct an ILU factorization in sparse skyline format, where the lower-part of the matrix is available as sparse rows, and the upper-part is available as sparse columns. This allows the condition of the incomplete L and U factors to be estimated during the factorization, and appropriate action may be taken as required. Like ILUT, fill-in is limited by both the drop-tolerance *droptol*, and the number of nonzeros *lfil* in each row of L or column of U.

The regular skyline format, where all elements within the profile of the matrix are stored, is very commonly used in finite element computations, particularly when direct methods are employed, for example, see [14]. This new factorization is directly suited to matrices in this format.

The preconditioner, which we call ILUS, is described in the next section. In Section 3, we show the numerical results of ILUS on the standard square lid-driven cavity problem, and in Section 4, we draw some conclusions.

2 ILUS

2.1 Factorization based on bordering

ILUS is an incomplete form of LDU Gaussian elimination based on bordering [19, 22]. Let A_{k+1} be the $(k+1)$-st leading principal submatrix of A and assume we have the decomposition $A_k = L_k D_k U_k$. Then we can compute the factorization of A_{k+1} using

$$\begin{pmatrix} A_k & v_k \\ w_k & \alpha_{k+1} \end{pmatrix} = \begin{pmatrix} L_k & 0 \\ y_k & 1 \end{pmatrix} \begin{pmatrix} D_k & 0 \\ 0 & d_{k+1} \end{pmatrix} \begin{pmatrix} U_k & z_k \\ 0 & 1 \end{pmatrix}$$

in which

$$z_k = D_k^{-1} L_k^{-1} v_k \tag{1}$$

$$y_k = w_k U_k^{-1} D_k^{-1} \tag{2}$$

$$d_{k+1} = \alpha_{k+1} - y_k D_k z_k. \tag{3}$$

Thus, we obtain each row and column of the factorization by solving two unit lower triangular systems and computing a scaled dot product. However, a sparse approximate

solution to the triangular systems is required, since we do not want the preconditioner to be dense, nor expensive to compute. In addition, a data structure that accesses the strict lower part of A by rows and the strict upper part of A by columns is required. The skyline or sparse skyline format as mentioned in Section 1 is appropriate for this purpose. This format should also be used to store the resultant L and U factors. In the symmetric case, ILUS is equivalent to an incomplete Cholesky factorization; half the work can be saved, since the computation of y_k is not necessary.

2.2   Techniques for sparse approximate solutions

There are a number of ways to compute the sparse approximations required in (1) and (2). It would seem natural to solve the given triangular systems exactly, and then use some strategy to drop small elements at the end. However, not only is the triangular solve relatively expensive, in order to retain no more than a fixed number of elements, some kind of partial sorting procedure is required, the cost of which is almost always unacceptable.

2.2.1   Truncated Neumann series approximation

One technique of approximating (1) is to use the truncated Neumann series, so that

$$z_k = D_k^{-1} L_k^{-1} v_k = D_k^{-1}(I + E + E^2 + \cdots + E^p)v_k \tag{4}$$

in which $E \equiv I - L$. In fact, by analogy with ILU(p), it is interesting to note that the powers of E_k will also tend to become smaller as p increases. A close look at the structure of $E_k^p v_k$ shows that there is indeed a strong relation between this approach and ILU(p) in the symmetric case.

Note that the matrices E are never formed and that the series is evaluated with Horner's rule. However it is more important to observe here that the vector $E_k^j v_k$ should be computed in sparse–sparse mode, i.e., the fact that both E_k^j and v_k are sparse should be exploited. When multiplying a sparse matrix A by a sparse vector v, the operation can be best done by accumulating the linear combinations of the columns of A. Therefore, instead of traversing the entire sparse matrix for the matrix-vector multiplication, only the columns corresponding to nonzeros in v are traversed, thus greatly reducing the cost. We will deal with this implementation issue in Section 2.3.

In the same vein, the computation of d_k via (3) involves the inner product of two sparse vectors. This is usually implemented by expanding one of the vectors into a full vector, and computing the inner product between a sparse vector by this full vector.

If the number of terms taken in the truncated Neumann series is large and the number of nonzeros in z_k exceeds the fill-in tolerance *lfil*, then some of the fill-ins must be dropped according to some strategy. The simplest strategy, one which tries to preserve accuracy in the factorization, is to retain the *lfil* elements of largest size. Another strategy is to

drop elements in z_k and y_k so that the resulting vectors have the same structure, i.e., the resulting preconditioner has symmetric structure. This may be a desirable property, and in addition, requires only the pattern of one triangular factor to be maintained. Yet another possible strategy will be mentioned in Section 2.4.

2.2.2   Approximate inverse techniques

A second, much cheaper approximation for (1) comes from approximate inverse techniques. Their most common application has been to independently approximate all the rows or columns of an inverse [2, 5, 7, 15, 12] or its factors [1, 16], and use it as a preconditioner. In the column case, for example, this can be done by minimizing the 2-norm of the residual,

$$\min_{x_j} \|e_j - Ax_j\|_2 \tag{5}$$

for each column j of the matrix, where e_j is the j-th coordinate vector, and where x_j is somehow constrained to be sparse. The minimization may be done in many ways, most obviously by using a QR factorization. However, since the exact minimum is not required, it may be cheaper to use a few steps of a descent-type method starting with a sparse initial guess [5].

It is useful in many circumstances to regard (5) as a general method to find a sparse approximate solution to a linear system [6]. In our context, to solve $Lz = v$ approximately, we focus on the minimization problem

$$\min_{z} \|v - Lz\|_2 \tag{6}$$

with respect to all sparse z. By constraining the nonzero pattern of z to be the same as that of v, we have a type of ILU(0) factorization. Fill-in may be introduced by calculating the residual norm decrease for each possible fill-in element and choosing the elements that give the most substantial decrease. If the QR decomposition has already been computed, the new minimization is performed with a simple update. However, it may be even more attractive to use a descent-type method, since in this case, fill-in is introduced naturally at each step.

To perform the minimization in (6), we use a small number of GMRES or Minimal Residual steps, and drop elements in the solution at the end of the steps according to *droptol*, and make sure the allowed fill-in *lfil* is not exceeded. Dropping may be applied at the end of each step if the number of steps is large, to reduce the cost of the method. Dropping may also be applied to the Krylov basis vectors if necessary, in which case a flexible version of GMRES [20] should be used.

Note that if no dropping is applied, the residual norm $\|v - Lz\|_2$ is guaranteed to decrease. However, this is no longer true whenever elements are dropped in the solution vector. This may have a detrimental effect on the result if many iterations are used, but can be avoided by dropping instead in the residual search direction. Minimization

in this direction guarantees that there is always a residual norm reduction. To control fill-in, entries in the search direction are retained if they correspond to nonzero entries already existing in the solution, or if they have large magnitude. In the Minimal Residual variation described by Algorithm 2.1 below, fill-in is introduced one at a time in Step 3. This makes the method more expensive than using GMRES and dropping at the end, but it is also possible to introduce fill-in more than one at a time. Like for the truncated Neumann series, all computations are performed in sparse–sparse mode.

ALGORITHM **2.1 Sparse approximate solution to** $Ax = b$

> 1.   *Starting with some initial guess x, $r := b - Ax$*
> 2.   *While x has fewer than* lfil *nonzeros*
> 3.   *Choose d to be r with the same pattern as x;*
>   *If $nnz(x) <$ lfil then add one entry which is the*
>   *largest remaining entry in absolute value*
> 4. $q := Ad$
> 5. $\alpha := \frac{(r,q)}{(q,q)}$
> 6. $r := r - \alpha q$
> 7. $x := x + \alpha d$
> 8.   *End do*

2.3   Companion structure

In the truncated Neumann series and approximate inverse techniques above, it is necessary, as mentioned, to perform matrix-vector multiplications in sparse–sparse mode, by accumulating linear combinations of the columns of A. Unfortunately, the unit lower triangular matrices L_k and U_k^T are stored by rows, making them inconducive for this operation. The data structure for storing the triangular matrices must be augmented by a linked-list companion structure, which points to the entries in the matrix column by column. We describe this structure now.

Suppose the matrix L is generated row by row and stored in compressed sparse row (CSR) format [23], using the arrays A(NNZ), JA(NNZ), IA(N+1), where NNZ is the number of nonzeros in the matrix, and N is the order of the matrix. (The sparse skyline format is this structure combined with the matrix U stored in compressed sparse column (CSC) format, with the diagonal stored separately; alternatively, a single A and JA may be used.) The linked-list companion structure requires three additional integer arrays: JSTART(N), LINK(NNZ), JR(NNZ). The arrays A, JA, LINK and JR are parallel arrays, i.e., the i-th element of each array refer to the same nonzero entry. JSTART is an array of integer pointers into the parallel arrays, pointing to the first nonzero entry in each column. JR stores the row index of this entry, and LINK stores the pointer to the next entry in the

column. A value of zero in `LINK` indicates that there is no next entry. When a new row is generated in L, the new nonzeros are added to the beginning of the linked lists.

The companion structure nearly doubles the storage required for the preconditioner. At least some of this storage needs to be allocated later, if GMRES, for example, is to be used. If memory usage is critical, then the lower triangular matrices may be stored directly in column format, at the cost of memory reallocation when necessary.

2.4   Estimating stability

A key advantage of the ILUS factorization is the ease with which the stability of its intermediate L_k and U_k factors may be determined. Since we eventually solve with L and U separately, it is reasonable to estimate $\|L_k^{-1}\|$ and $\|U_k^{-1}\|$ separately. In fact, we do not actually need a very good estimate of the norm, only a rough indication of its size, and whether or not it is growing rapidly as the factorization is progressing. We have found that for the lower triangular factor, the infinity norm bound $\|L_k^{-1}e\|_\infty$, where e is a vector of all ones, is effective for this purpose. In addition, by estimating the norm of the factors separately, the solution and norm of $L_{k+1}^{-1}e$ may be updated easily from $L_k^{-1}e$ with one sparse SAXPY operation. Unfortunately, this is not true for the upper triangular factor. In this case, we estimate the infinity norm for its transpose. Other more complicated condition estimates are possible [11], but we have not found them to be necessary for our purpose.

An interesting way to determine how badly L_k and U_k are conditioned is to examine the residual norm reduction in the approximate inverse iteration. If the residual was reduced by very little, this may indicate that the factors are poorly conditioned. However, this measure is not usually monotone with k like the norm bound above, and is thus difficult to utilize.

When instability has been detected, for example when a norm estimate exceeds some stable norm limit, the ILUS factorization code exits and indicates that the solver should switch to another preconditioner, or restart ILUS with more allowed fill-in. This kind of behavior can save much computation time, especially when dealing with new and large matrices.

Instead of exiting, it is tempting to increase the allowed fill-in and continue with the current row. This must be accompanied by an increase in the norm limit, since increasing the number of allowed fill-ins does not guarantee that the norm estimate will decrease, nor that the norm limit will not again be exceeded very soon. Unfortunately, our experiments clearly show that by the time even mild instability has been detected, the factorization is already too inaccurate to support continuing the factorization with more fill-in.

Another tempting step is to stop the current factorization and proceed with the remaining submatrix. This is a technique of blocking the matrix, while extracting an incomplete factorization for the diagonal blocks. We experimented with this idea and found that a major problem is that many w_k and v_k can be zero after the blocking. When combined

with an α_{k+1} that is also zero, this leads to singular preconditioner. Complicated reorder-
ing would be necessary to solve this problem. Note that this problem may also occur
without blocking, but is far less likely.

Finally, we would like to mention that a weighted dropping scheme for the elements
in z_k may be used, for example by using D_k^{-1} or $L_k^{-1}e$ as inverse weights. This allows the
stability of the factors to be controlled in some sense. In one extreme, if all elements in
z_k must be dropped, this corresponds to breaking the triangular solve recurrence, just as
blocking does above.

The ILUS algorithm may be summarized as follows.

ALGORITHM 2.2 **ILUS**

    1.   Set $D_1 = a_{11}, L_1 = U_1 = 1$
    2.   For $k = 1,\ldots,n-1$
    3.   Compute a sparse $z_k \approx D_k^{-1} L_k^{-1} v_k$
    4.   Compute a sparse $y_k \approx w_k U_k^{-1} D_k^{-1}$
    5.   Compute $d_{k+1} := \alpha_{k+1} - y_k D_k z_k$
    6.   Estimate $\|L^{-1}\|$ and $\|U^{-1}\|$ and exit if either exceeds some limit
    7.   End do

3   Numerical Results

We first wish to illustrate how easily ILUS detects instability in the FIDAP Example
problem described in the Introduction. Figure 1 illustrates the growth in the condition
norm bounds as the factorization progresses. In typical operation, the factorization would
have been aborted when the bounds exceeded some level.

We tested ILUS by using it as a preconditioner for GMRES for solving the linearized
systems arising from the Galerkin Finite Element discretization of the square lid-driven
cavity problem. Rectangular elements were used, with biquadratic basis functions for
velocities, and linear discontinuous basis functions for pressure. A mesh of 40 by 40
elements was used, leading to matrices of size $n = 17922$ and having $nnz =$567,467
nonzero entries. The degrees of freedom were numbered element by element. The solution
at Reynolds number 1000 was obtained by solving a sequence of problems with Reynolds
number ramped in increments of 100, and the results are shown for the first Jacobian
system of the Newton iterations at each Reynolds number. A zero initial guess is used for
GMRES with 20 Krylov basis vectors. The linear iterations is stopped when the initial
residual is reduced by 10^{-7}, a much stronger tolerance than is normally used within
nonlinear iterations.

The scalings of the matrices are poor, since each matrix contains different types of
equations and variables. Thus, we scaled the systems so that each row and then each
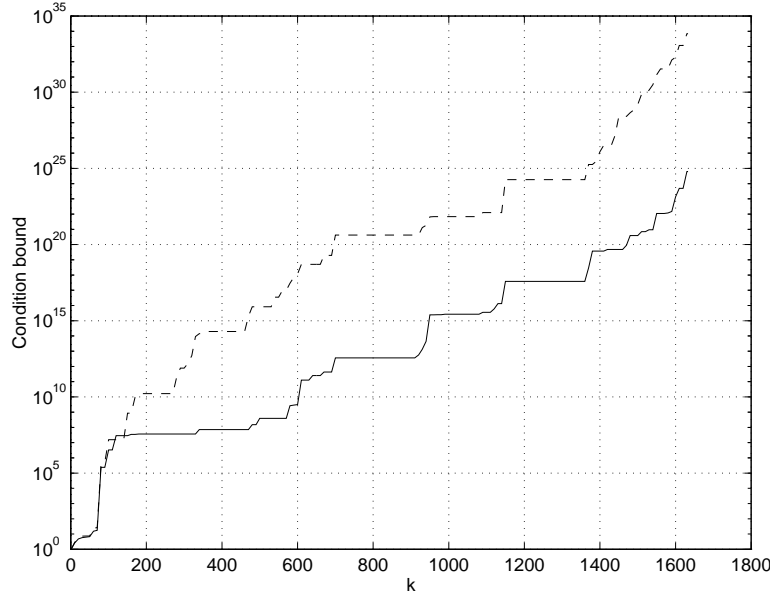
Figure 1: Growth in $\|L_k^{-1}e\|_\infty$ (solid line) and $\|U_k^{-T}e\|_\infty$ (dashed line) for EX07.

column have unit 2-norm. To assure ourselves that no accuracy was lost with this procedure, we examined the residual norm reduction computed with the unscaled matrix and also the relative error norm using a solution computed by a direct frontal method with pivoting. In all cases these were approximately 10^{-7}.

Table 2 reports the number of GMRES iterations required for each linear system, and the computation time on one processor of a Cray C90 supercomputer in 64-bit arithmetic. The computation time is divided into the time to compute the ILUS preconditioner, and the time required by the GMRES iterations. The approximate inverse procedure also used GMRES, starting with the right-hand-side of the linear systems as the initial guess. Three iterations were used, without dropping between steps, the parameter *lfil* was set at 40, and no drop tolerance was used.

For comparison with direct methods, at Reynolds number 1000, ILUS produced a preconditioner with 1,403,370 total nonzeros, while the frontal solver produced an upper triangular factor with 4,327,550 nonzeros, and a lower triangular factor with 4,424,959 nonzeros. Although the direct solver is much faster for these problems, the storage requirement for the iterative solver is much less, even including the companion structure.

Figure 2 illustrates the increase in the condition norm bound for the L factor at Reynolds number 0, on a small 20 by 20 mesh ($n = 4562$, *lfil* = 30, all other parameters the same). For comparison, the growth of this bound is illustrated for the same, but unscaled matrix. This small, but unscaled problem could not be solved with GMRES and this preconditioner.

	GMRES	CPU time (s)		
Re.	iterations	precon	solve	total
0	68	135.3	7.1	142.4
100	93	131.9	9.4	141.3
200	209	133.2	21.4	154.6
300	189	130.7	19.1	149.8
400	145	132.3	14.8	147.2
500	222	130.5	22.5	153.0
600	235	132.8	24.0	156.9
700	258	131.5	26.1	157.6
800	147	134.1	15.0	149.1
900	264	132.8	26.8	159.6
1000	391	135.4	39.9	175.4

Table 2: Test results for the driven cavity problems.

4   Conclusions

An alternative form of incomplete factorization has been described and tested. This form has many desirable properties, such as its amenability to the skyline format, the ease with which stability may be monitored, and the possibility of constructing a preconditioner with symmetric structure. Various stability issues have also discussed, such as systematically checking some estimate of the norm of $(LU)^{-1}$, furthering some understanding of what is required for robust preconditioners. As for any complete or incomplete factorization, the ordering of the matrix plays an important role [8]. Whereas 'preordering' the matrix does not cause any more difficulties with ILUS than with other ILU factorizations, one disadvantage of ILUS is that it cannot easily accomodate 'dynamic' orderings, i.e., orderings that can be generated as the factorization progresses. For example, partial column or row pivoting would be difficult to implement.
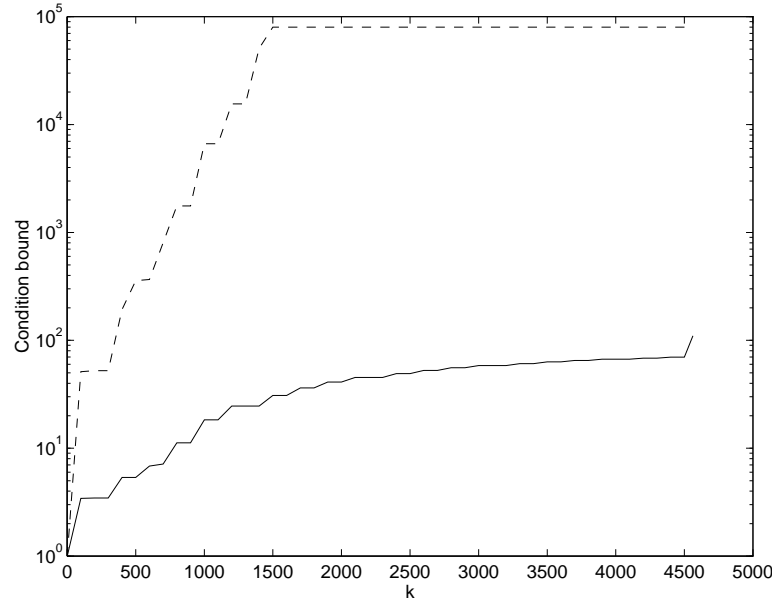
Figure 2: Growth in condition norm bounds at Re. 0, Scaled (solid line), Unscaled (dashed line).

References

[1] O. Axelsson. *Iterative Solution Methods.* Cambridge, Cambridge, 1994.

[2] M. W. Benson and P. O. Frederickson. Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems. *Utilitas Math.,* 22 (1982), pp. 127-140.

[3] A. M. Bruaset, A. Tveito and R. Winther. On the stability of relaxed incomplete LU factorizations, *Math. Comp.,* 54 (1990), pp. 701-719.

[4] P. Chin, E. F. D'Azevedo, P. A. Forsyth and W.-P. Tang. Preconditioned conjugate gradient methods for the incompressible Navier-Stokes equations, *Int. J. Numer. Methods Fluids,* 15 (1992), pp. 273-295.

[5] E. Chow and Y. Saad. Approximate inverse preconditioners for general sparse matrices, Technical Report UMSI 94/101, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, Minnesota, 1994.

[6] E. Chow and Y. Saad. Approximate inverse techniques for block-partitioned matrices, Technical Report UMSI 95/13, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, Minnesota, 1995.

[7] J. D. F. Cosgrove, J. C. Díaz and A. Griewank. Approximate inverse preconditioning for sparse linear systems. *Intl. J. Comp. Math.*, 44 (1992), pp. 91-110.

[8] L. C. Dutto. The effect of ordering on preconditioned GMRES algorithms, for solving the compressible Navier-Stokes equations, *Int. J. Numer. Methods Engrg.*, 36 (1993), pp. 457-497.

[9] H. C. Elman. A stability analysis of incomplete LU factorizations. *Math. Comp.*, 47 (1986), pp. 191-217.

[10] M. Engelman. *FIDAP: Examples Manual*, Revision 6.0. Fluid Dynamics International, Evanston, Illinois, 1991.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations*, 2nd edition. John Hopkins, Baltimore, 1989.

[12] M. Grote and H. D. Simon. Parallel preconditioning and approximate inverses on the Connection Machine. In R. F. Sincovec, D. E. Keyes, L. R. Petzold and D. A. Reed, eds., *Parallel Processing for Scientific Computing*, vol. 2, pp. 519-523. SIAM, Philadelphia, Pennsylvania, 1993.

[13] I. Gustafsson. A class of first order factorization methods, *BIT*, 18 (1978), pp. 142-156.

[14] Y. Hasbani and M. Engelman. Out-of-core solution of linear equations with non-symmetric coefficient matrix, *Computers and Fluids*, 7 (1979), pp.13-31.

[15] T. Huckle and M. Grote. A new approach to parallel preconditioning with sparse approximate inverses. Manuscript SCCM-94-03, Scientific Computing and Computational Mathematics Program, Stanford University, Stanford, California, 1994.

[16] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings I. Theory. *SIAM J. Matrix Anal. Appl.*, 14 (1993), pp. 45-58.

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

[18] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients, *ACM Trans. Math. Softw.*, 6 (1980), pp. 206-219.

[19] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum Press, New York, 1988.

[20] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14 (1993), pp. 461-469.

[21] Y. Saad. ILUT: A dual threshold incomplete LU factorization. *Num. Lin. Alg. Appl.,* 1 (1994), pp. 387-402.

[22] Y. Saad. Preconditioned Krylov subspace methods for CFD applications, *Proceedings of the International Workshop on Solution Techniques for Large-Scale CFD Problems,* Montréal, Sept. 26–28, 1994, pp. 179-195.

[23] Y. Saad. SPARSKIT: a basic tool kit for sparse matrix computations. Technical Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.

[24] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Statist. Comput.,* 7 (1986), pp. 856-869.

[25] J. W. Watts-III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation, *Soc. Pet. Eng. J.,* 21 (1981), pp. 345-353.