INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 25, 739–748 (1997)

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, U.S.A.

SUMMARY

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. © 1997 John Wiley & Sons, Ltd.

Int. J. Numer. Meth. Fluids, **25**: 739–748 (1997).

No. of Figures: 2. No. of Tables: 2. No. of References: 28.

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 factors L and U such that the norm of $(LU)^{-1}$ is very large. The long recurrences associated with solving with these factors are unstable,[1,2] producing solutions with extremely large components. A sign of this severely poor preconditioning is the erratic behaviour of

* Correspondence to: E. Chow, Department of Computer Science and Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN 55455, U.S.A.

Contract grant sponsor: National Science Foundation; Contract grant number: NSF/CCR-9214116
Contract grant sponsor: NASA; Contract grant number: NAG2-904

CCC 0271–2091/97/070739–10 $17.50 *Received 24 April 1995*
© 1997 John Wiley & Sons, Ltd. *Revised 17 October 1996*

the iterative method, e.g. 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.[3] This example models natural convection and the matrix for the first step of the first non-linear iteration has order 1633 and 46,626 non-zeros. Table I 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 non-zeros (*lfil* per row per L- and U-factor) is reduced. Another typical feature is that the norm bound decreases again for small *lfil*. The GMRES iterative procedure[4] 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.

It is usually possible to produce a usable ILU factorization by allowing enough fill-in. Starting with ILU(0) or IC(0), where the non-zero pattern of the factorization is the same as that of the original matrix A, fill-in may be introduced by level-of-fill[5–7] or by threshold.[8] These indeed have been very successful for many fluid flow problems (see e.g. Reference 9). 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,[10] 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 I was calculated using the ILUT factors.

The drop tolerance and how much fill-in is required to produce an accurate and/or stable factorization are difficult to predict. However, it is possible to determine in advance whether or not a factorization will fail owing 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 factors L and U 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 non-zeros *lfil* in each row of L or column of U. The latter parameter allows the maximum storage for the preconditioner to be known beforehand.

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 (see e.g. Reference 11). This new factorization is directed suited to matrices in this format.

The preconditioner, which we call ILUS, is described in the next section. ILUS defines the procedure for computing an incomplete factorization by threshold for non-symmetric matrices in sparse skyline format. This is valuable if the original matrix is stored in skyline format, since conversion to row- or column-based data structures to compute an incomplete factorization is expensive and often requires a copy of the matrix. 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.

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

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

2. ILUS

2.1. *Factorization based on bordering*

ILUS is an incomplete form of LDU Gaussian elimination based on bordering.[12,13] Let A_{k+1} be the $(k+1)$th 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 factors L and U. 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, but also, 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. The first idea that comes to mind for approximating (1) is to use the truncated Neumann series, so that

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

in which $E_k \equiv I - L_k$. 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_k 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 non-zeros 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 and this full vector.

If the number of terms taken in the truncated Neumann series is large and the number of non-zeros 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[14–18] or its factors[19,20] and use it as a preconditioner. In the column case, for example, this can be done by minimizing the two-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 jth co-ordinate vector and 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.[15]

It is useful in many circumstances to regard (5) as a general method to find a sparse approximate solution to a linear system.[21] 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 non-zero 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, 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[22] 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 non-zero entries already existing in the solution or if they have large magnitude. In the minimal residual variation described by Algorithm 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. As for the truncated Neumann series, all computations are performed in sparse–sparse mode.

Algorithm 1. Sparse approximate solution to $Ax = b$

```
1. Starting with some initial guess $x$, $r := b - Ax$
2. While $x$ has fewer than $lfil$ non-zeros
3.    Choose $d$ to be $r$ with the same pattern as $x$;
      If $nnz(r) < lfil$ then add one entry which is the largest remaining entry in absolute value
4.    $q := Ad$
5.    $\alpha := (r, q)/(q, q)$
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) and IA(N + 1), where NNZ is the number of non-zeros 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) and JR(NNZ). The arrays A, JA, LINK and JR are parallel arrays, i.e. the ith element of each array refers to the same non-zero entry. JSTART is an array of integer pointers into the parallel arrays, pointing to the first non-zero 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 non-zeros 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 factors L_k and U_k 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, the solution and norm of $L_{k+1}^{-1}e$ may be updated easily: the last component of $L_{k+1}^{-1}e$ can be determined with one sparse SAXPY operation. Unfortunately, this cannot be done for the upper triangular factor. In this case we estimate the infinity-norm of its transpose. Other more complicated condition estimates are possible,[24–26] 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, e.g. 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 behaviour 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. We performed several experiments and, unfortunately, they showed that by the time even mild instability has been detected, the factorization was 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 a singular preconditioner. Complicated reordering would be necessary to solve this problem. Note that this problem may also occur without blocking, but it is far less likely.

It is also possible that a weighted dropping scheme for the elements in z_k could be used, e.g. 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.

ILUS in its simplest form does not circumvent instability any better than ILUT, for example. Techniques for augmenting the diagonal elements to enhance stability are possible for various forms of incomplete factorization (see e.g. Reference 25, p. 196). The ILUS algorithm may be summarized as follows.

Algorithm 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 Section 1. 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.

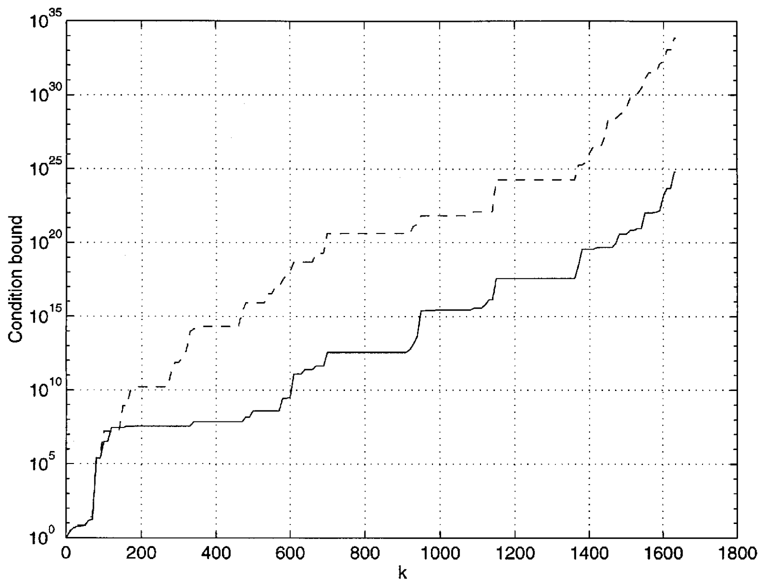

Figure 1. Growth in $\|L_k^{-1}e\|_\infty$ (full line) and $\|U_k^{-T}e\|_\infty$ (broken line) for EX07

We tested ILUS by using it as a preconditioner for GMRES for solving the linear systems arising from the square lid-driven cavity problem.[27] The problem is modelled by the incompressible Navier–Stokes equations

$$Re(\mathbf{u}\cdot\nabla\mathbf{u}) = -\nabla p + \nabla^2\mathbf{u}, \tag{7}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{8}$$

over the unit square, where **u** denotes the velocity variables, p denotes the pressure variables and Re is the Reynolds number. The boundary conditions are $\mathbf{u} = (1,\ 0)^{\mathrm{T}}$ on the top edge of the square and $\mathbf{u} = (0,\ 0)^{\mathrm{T}}$ on the other three sides and the corners. The reference pressure specified at the bottom left corner is zero.

The Galerkin finite element method was used to discretize the problem, using rectangular elements with biquadratic basis functions for velocities and linear discontinuous basis functions for pressure. A mesh of 40×40 elements was used, leading to matrices of size $n = 17{,}922$ and having $nnz = 567{,}467$ non-zero 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. The matrices have symmetric structure; only the matrix for Reynolds number zero is symmetric. A zero initial guess is used for GMRES with 20 Krylov basis vectors. The linear iterations are stopped when the initial residual is reduced by 10^{-7}, a much stronger tolerance than is normally used within non-linear iterations.

Rows in the Jacobian matrix express either continuity or conservation of momentum. The momentum equations contain the product of the velocity gradient and the Reynolds number. At high Reynolds numbers the velocity gradient will be large and the magnitude of the momentum equations is much larger than that of the continuity equations. This difference in scales causes poor behaviour in threshold incomplete factorizations. Thus we scaled the linear systems so that each row has unit two-norm and then scaled again so that each column has unit two-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 II 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 was used to compute the ILUS factorization. This 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 non-zeros, while the frontal solver produced an upper triangular factor with 4,327,550 non-zeros and a lower triangular factor with 4,424,959 non-zeros. 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. The advantage of iterative methods for problems of this size is storage rather than time. For larger problems, iterative methods may also be advantageous in terms of time.

Figure 2 illustrates the increase in the condition norm bound for the L-factor at Reynolds number zero on a small (20×20) 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. The unscaled problem could not be solved with GMRES and this preconditioner.

4. CONCLUSIONS

The bordered form of incomplete factorization by threshold 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 been 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.[28] Whereas 'preordering' the matrix does not cause any more difficulties with ILUS

Table II. Test results for driven-cavity problems

		CPU time (s)		
Re	GMRES 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

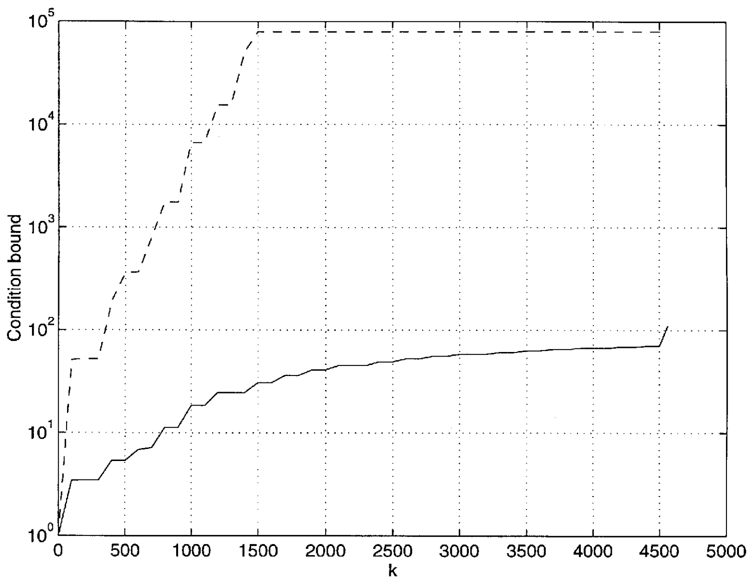

Figure 2. Growth in condition norm bounds at $Re = 0$, scaled (full line) and unscaled (broken line)

than with other ILU factorizations, one disadvantage of ILUS is that it cannot easily accommodate '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.

ACKNOWLEDGEMENT

The authors wish to thank Andrew Chapman (Minnesota Supercomputer Institute) for the use of his lid-driven cavity code and for his generous help during this work. The first author also wishes to express his thanks to Michael Heroux for many helpful discussions and for inviting him to spend some time at Cray Research where this work was first begun. Computer facilities for this research were provided by both the Minnesota Supercomputer Institute and Cray Research, Inc.

This work was supported in part by the National Science Foundation under grant NSF/CCR-9214116 and in part by NASA under grant NAG2-904.

REFERENCES

1. A. M. Bruaset, A. Tveito and R. Winther, 'On the stability of relaxed incomplete LU factorizations', *Math. Comput.*, **54**, 701–719 (1990).
2. H. C. Elman, 'A stability analysis of incomplete LU factorizations', *Math. Comput.*, **47**, 191–217 (1986).
3. M. Engleman, *FIDAP: Examples Manual, Revision 6.0*, Fluid Dynamics International, Evanston, IL, 1991.
4. Y. Saad and M. Schultz, 'GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems', *SIAM J. Sci. Stat. Comput.*, **7**, 856–869 (1986).
5. I. Gustafsson, 'A class of first order factorization methods', *BIT*, **18**, 142–156 (1978).
6. 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**, 148–162 (1977).
7. J. W. Watts III, 'A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation', *Soc. Petrol Eng. J.*, **21**, 345–353 (1981).
8. N. Munksgaard, 'Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients', *ACM Trans. Math. Softw.*, **6**, 206–219 (1980).

9. P. Chin, E. F. D'Azevedo, P. A. Forsyth and W.-P. Tang, 'Preconditiioned conjugate gradient methods for the incompressible Navier–Stokes equations', *Int. j. numer. methods fluids*, **15**, 273–295 (1992).
10. Y. Saad, 'ILUT: a dual threshold incomplete LU factorization', *Numer. Linear Algebra Appl.*, **1**, 387–402 (1994).
11. Y. Hasbani and M. Engleman, 'Out-of-core solution of linear equations with non-symmetric coefficient matrix', *Comput. Fluids*, **7**, 13–31 (1979).
12. J. M. Ortega, *Introduction to Parallel and Vector Solution of Linear Systems*, Plenum, New York, 1988.
13. Y. Saad, 'Preconditioned Krylov subspace methods for CFD applications', *Proc. Int. Workshop on Solution Techniques for Large-Scale CFD Problems*, Montreal, September 1994, pp. 179–185.
14. M. W. Benson and P. O. Frederickson, 'Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems', *Utilitas Math.*, **22**, 127–140 (1982).
15. E. Chow and Y. Saad, 'Approximate inverse preconditioners via sparse–sparse iterations', *SIAM J. Sci. Comput.*, in press.
16. J. D. F. Cosgrove, J. C. Díaz and A. Griewank, 'Approximate inverse preconditioning for sparse linear systems', *Int. J. Comput. Math.*, **44**, 91–110 (1992).
17. M. Grote and T. Huckle, 'Parallel preconditioning with sparse approximate inverses', *SIAM J. Sci. Comput.*, in press.
18. 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, SIAM, Philadelphia, PA, 1993, pp. 519–523.
19. O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, 1994.
20. L. Yu Kolotilina and A. Yu. Yeremin, 'Factorized sparse approximate inverse preconditionings I. Theory'. *SIAM J. Matrix Anal. Appl.*, **14**, 45–58 (1993).
21. E. Chow and Y. Saad, 'Approximate inverse techniques for block-partitioned matrices', *SIAM J. Sci. Comput.*, in press.
22. Y. Saad, 'A flexible inner–outer preconditioned GMRES algorithm', *SIAM J. Sci. Comput.*, **14**, 461–469 (1993).
23. Y. Saad, 'SPARSKIT: a basic tool kit for sparse matrix computations', *Tech. Rep. 90-20*, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.
24. C. H. Bischof, J. G. Lewis and D. J. Pierce, 'Incremental condition estimation for sparse matrices', *SIAM J. Matrix Anal. Appl.*, **11**, 644–659 (1990).
25. I. S. Duff, R. G. Grimes and J. G. Lewis, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1989.
26. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd edn, Johns Hopkins University Press, Baltimore, MD, 1989.
27. U. Ghia, K. N. Ghia and C. T. Shin, 'High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method', *J. Comput. Phys.*, **48**, 387–411 (1982).
28. L. C. Dutto, 'The effect of ordering on preconditioned GMRES algorithms, for solving the compressible Navier–Stokes equations', *Int. j. numer. methods eng.*, **36**, 457–497 (1993).