SIAM J. SCI. COMPUT.
Vol. 22, No. 4, pp. 1333–1353

PRECONDITIONING HIGHLY INDEFINITE AND NONSYMMETRIC MATRICES*

MICHELE BENZI†, JOHN C. HAWS‡, AND MIROSLAV TŮMA§

Abstract. Standard preconditioners, like incomplete factorizations, perform well when the coefficient matrix is diagonally dominant, but often fail on general sparse matrices. We experiment with nonsymmetric permutations and scalings aimed at placing large entries on the diagonal in the context of preconditioning for general sparse matrices. The permutations and scalings are those developed by Olschowka and Neumaier [*Linear Algebra Appl.*, 240 (1996), pp. 131–151] and by Duff and Koster [*SIAM J. Matrix Anal. Appl.*, 20 (1999), pp. 889–901; Tech. report Ral-Tr-99-030, Rutherford Appleton Laboratory, Chilton, UK, 1999]. We target highly indefinite, nonsymmetric problems that cause difficulties for preconditioned iterative solvers. Our numerical experiments indicate that the reliability and performance of preconditioned iterative solvers are greatly enhanced by such preprocessing.

Key words. linear systems of equations, sparse matrices, maximum transversal, nonsymmetric permutations, row and column scalings, preconditioned iterative methods, incomplete LU factorization, sparse approximate inverses

AMS subject classifications. Primary, 65F10, 65N22, 65F50; Secondary, 15A06

PII. S1064827599361308

1. Introduction.

1.1. Motivation and focus. We consider the solution of sparse linear systems $Ax = b$, where A is a general sparse $n \times n$ nonsingular matrix, by preconditioned Krylov subspace methods [25], [39]. For a *general* sparse matrix we mean a matrix that has no special properties, such as symmetry, positive definiteness, diagonal dominance, etc. In particular, we focus on matrices that are highly unstructured, nonsymmetric (structurally as well as numerically), and indefinite; i.e., the eigenvalues of A can lie anywhere in the complex plane. Such matrices arise frequently in the simulation of chemical engineering processes, in economic modeling, in management science, in the analysis of circuits and power system networks, and elsewhere. These problems are very different from the ones arising from the numerical solution of elliptic partial differential equations (PDEs) and can cause serious difficulties for standard iterative methods and preconditioners.

There have been a few attempts to use preconditioned Krylov subspace methods in these contexts, but in general the results have been far from satisfactory. For

*Received by the editors September 17, 1999; accepted for publication (in revised form) June 7, 2000; published electronically October 25, 2000. A preliminary version of this paper appeared as Technical Report LA-UR-99-4857, Los Alamos National Laboratory, Los Alamos, NM, June 2000.

http://www.siam.org/journals/sisc/22-4/36130.html

†Department of Mathematics and Computer Science, Emory University, Atlanta, GA 30322 (benzi@mathcs.emory.edu). The research of this author was supported in part by Department of Energy grant W-7405-ENG-36 with Los Alamos National Laboratory.

‡Department of Mathematics, North Carolina State University, Raleigh, NC 27695-8205 (jchaws@math.ncsu.edu).

§Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic (tuma@cs.cas.cz). The research of this author was supported by Grant Agency of the Czech Academy of Sciences grants 2030706 and 2030801.

example, in [11], various incomplete factorization (ILU) preconditioners and iterative solvers were tested on a set of standard problems from chemical engineering. The main conclusions of that study were that such linear systems are difficult to solve with iterative methods (as indicated by the large number of reported failures) and that realizing the potential of iterative solvers will require improvements in the reordering and/or preconditioning schemes. Similar conclusions were reached in [33] for the use of iterative methods in circuit simulations. The use of preconditioned Krylov subspace methods for the solution of sparse linear systems arising in economic modeling has been investigated in [37] and [24]. There the conclusion was that iterative solvers are often superior to direct ones. However, some of the problems could not be solved by iterative methods; see [37].

Preconditioned iterative methods work especially well when the coefficient matrix is, at least to some degree, diagonally dominant. Reliable methods also exist to handle matrices that are symmetric positive definite, or M-matrices. Matrices with these properties arise frequently from the discretization of second-order, elliptic PDEs. Standard preconditioners, such as those based on incomplete factorizations of the coefficient matrix, are usually reliable under these circumstances and typically deliver good rates of convergence. In contrast, such preconditioners are often unstable or may not even be defined when the coefficient matrix has zeros on the main diagonal and/or is highly nonsymmetric (see the discussion in section 2). Furthermore, the presence of many eigenvalues with arbitrary real part (positive, negative, and zero) causes serious difficulties for many Krylov subspace solvers. Matrices of this kind are loosely referred to as *highly indefinite*, regardless of whether or not they are symmetric. Coefficient matrices from chemical engineering, circuits, economics, etc., often exhibit a large number of zero diagonal entries and poor spectral distributions, and they represent a challenge for preconditioned Krylov subspace solvers.

1.2. Contributions of the paper. In [36], Olschowka and Neumaier introduce new permutations and scaling strategies for Gaussian elimination. The goal is to preprocess the coefficient matrix so as to obtain an equivalent system with a matrix that is more diagonally dominant. This preprocessing reduces the need for partial pivoting, thereby speeding up the solution process, and has a beneficial effect on the accuracy of the computed solution. Although the focus in [36] is on dense systems, the sparse case and the case of incomplete factorizations are also briefly discussed. These and other heuristics have been further developed and efficiently implemented by Duff and Koster; see [18] and [19]. Some evidence of the usefulness of these preprocessings in connection with sparse direct solvers and for ILU preconditioning has been provided in [18] and [19]; see also [30]. Our contribution is to carry out a systematic experimental study of the use of these permutation and scaling algorithms in the context of preconditioned iterative methods applied to challenging linear systems. We consider a number of different preconditioners (diagonal, ILU, sparse approximate inverses) and the combined use of nonsymmetric permutations aimed at improving numerical stability with symmetric ones aimed at reducing fill-in in the preconditioner. Our experiments indicate that this preprocessing, and particularly the use of maximum product transversals, enables the stable computation of the preconditioners, resulting in an overall solution strategy that is both reliable and cost effective.

While we do not claim that this approach to preconditioning general sparse matrices will always work, we do hope that the results in this paper will contribute to a reassessment of the role of iterative solvers in areas where these methods had been almost written off as unreliable, such as chemical engineering. We also hope that

one-sided permutations (and related scalings) will find widespread use in the arena of preconditioned iterative solvers for highly indefinite and nonsymmetric linear systems.

The paper is organized as follows. In section 2 we briefly discuss the preconditioners used in the paper. In section 3, which is based on [19], we recall the one-sided permutations and scalings used to preprocess the matrices. The test problems used for the numerical experiments are described in section 4, and the numerical experiments themselves in section 5. Finally, in section 6 we present our conclusions.

2. ILU and approximate inverse preconditioners. In this section we briefly discuss the preconditioners used in the numerical experiments. We focus our attention on ILU-type techniques and on a sparse approximate inverse preconditioner in factorized form, AINV. These are general-purpose, algebraic preconditioners that have been used successfully to solve a wide range of problems, particularly from PDEs. For a detailed treatment of ILU preconditioning we refer the reader to [39]. For a recent survey of sparse approximate inverse preconditioners, see [6].

ILU methods compute sparse approximations to the triangular factors L and U of A. The incomplete factors are obtained by dropping nonzero entries generated in the course of the factorization process (*fill-ins*) according to some rule. Different dropping rules give rise to different ILU preconditioners. When all fill-ins are discarded and only nonzeros in positions corresponding to the nonzero entries of A are retained, the ILU(0) preconditioner [34] is obtained. This preconditioner is easy to implement and inexpensive to compute, but it is often not good enough, particularly for the kind of challenging problems considered in this paper. More powerful preconditioners can be obtained by allowing more fill-in in the incomplete factors, or by dropping fill-ins based on their value rather than position. These techniques include level-of-fills ILU, denoted ILU(k), and dual threshold ILU, denoted ILUT(tol, p). Here $k \geq 1$ is the fill level, $tol \geq 0$ is a drop tolerance, and $p \geq 0$ denotes the number of off-diagonal nonzeros that are retained in each row of the incomplete factor (usually the p largest ones among those nonzeros that are greater than tol in absolute value).

Although fairly robust in practice, ILU preconditioners often fail on general sparse matrices because of instabilities (see below). In fact, the incomplete factors may not even exist. One way to improve their robustness is by incorporating partial (column) pivoting in the incomplete factorization. In the case of ILUT, this leads to a variant, called ILUTP [39], which is sometimes successful when ILUT fails. However, even this approach often fails when applied to general sparse matrices; see [10] and subsection 5.4 below.

There are two types of instability that ILU preconditioners may suffer from. These instabilities have been discussed in detail in [10]; here we give a brief discussion only. Let $\bar{L}$ and $\bar{U}$ denote the incomplete factors, and let

$$R := \bar{L}\bar{U} - A$$

denote the residual matrix. Also, let

$$E := I - A(\bar{L}\bar{U})^{-1}$$

denote the error matrix, assuming that preconditioning is being applied on the right. Notice that $E = R(\bar{L}\bar{U})^{-1}$. Let $\|\cdot\|_F$ denote the Frobenius matrix norm. In the symmetric positive definite case, there is a good correlation between the size of $\|R\|_F$ and the rate of convergence of the preconditioned conjugate gradient method [20]. On the other hand, in the case of general sparse matrices, $\|R\|_F$ alone is not a reliable

indicator of the quality of the preconditioner; instead, both $\|R\|_F$ and $\|E\|_F$ should be taken into account. Note that $\|R\|_F$ can be interpreted as a measure of the *accuracy* of the preconditioner, regarded as an approximation of A, while $\|E\|_F$ gauges the *stability* of the approximation. In general, a large value of $\|R\|_F$ means a poor approximation and hence a poor preconditioner. This can be caused, for instance, by very small pivots encountered in the course of the incomplete factorization, and accounts for a first kind of instability. However, even if $\|R\|_F$ is small, it can happen that $\bar{L}^{-1}$ and/or $\bar{U}^{-1}$ have very large entries. Therefore $\|E\|_F = \|R(\bar{L}\bar{U})^{-1}\|_F$ could be large, in which case the preconditioned matrix will be far from the identity, and the preconditioned iteration will either stagnate or diverge, leading to a second kind of instability. We stress that the instability here is not in the incomplete factorization process, but in the incomplete factors. Unstable (more accurately, ill-conditioned) ILU factors occur frequently for matrices that are far from symmetric and lack diagonal dominance; see [22], [10], [4].

For general sparse matrices, it is frequently the case that both kinds of instability occur simultaneously, with a crippling effect on the quality of the preconditioner. In general, the *complete* LU factorization of A may not even be defined without pivoting, or it may be unstable. This can happen, for example, when there are zero or small entries on the main diagonal. As long as A is nonsingular it has (in exact arithmetic) an LU factorization with pivoting, but this is not true for *incomplete* factorizations [38]. On the other hand, ILU factorizations that are both accurate and stable are possible for diagonally dominant matrices.

Because of these and other limitations of ILU-type preconditioners, alternative preconditioning techniques based on sparse approximate inverses have been intensively developed in the past few years. The AINV algorithm [3], [5], based on an incomplete biconjugation process, has been shown to be one of the most effective techniques in this class. Here the preconditioner is the product of two triangular matrices, which are sparse approximations to the inverses of the L and U factors of A. Sparsity is preserved by using a drop tolerance. The approximate inverse factors are computed directly from A; no knowledge of the factors L or U is needed. Factorized approximate inverse preconditioners share one drawback with ILU-type techniques: The construction phase is guaranteed to be breakdown-free only for special classes of matrices. Sufficient conditions are that A be symmetric positive definite or an H-matrix; see [29], [3], [28], [2]. For general sparse matrices, instabilities due to very small or zero pivots can occur during the construction of the preconditioner, with disastrous effects. This is perfectly analogous to the instability of the first kind for ILU. Notice that, in contrast to ILU, the instability of the second kind—unstable ILU solves—is not an issue here.

Like for ILU, the performance of approximate inverse preconditioners in factorized form is sensitive to the ordering of the matrix. For structurally symmetric (or nearly so) matrices having a stable AINV preconditioner, it was shown in [9] and [7] that symmetric reorderings that reduce fill-in in the inverse factors, like minimum degree or (generalized) nested dissection, can be used to improve the performance of the preconditioner. However, these symmetric reorderings alone are of little use for general sparse matrices, as in this case the AINV preconditioner may not even be defined. As we shall see, the stability and effectiveness of AINV can be dramatically improved by nonsymmetric permutations and scalings aimed at placing large entries on the main diagonal.

3. Overview of algorithms for finding maximum transversals. In this section we give a summary of algorithms for determining permutations of a matrix that place entries of large absolute value on the main diagonal. The codes used to

perform the permutations were taken from MC64, a set of Fortran routines that will be included in a forthcoming release of the Harwell Subroutine Library. Further details on the algorithms and implementations are provided in [19].

We examine three approaches for permuting large entries to the diagonal of matrices. First, we discuss methods for permuting a matrix so that the diagonal contains a maximum number of nonzeros; this method is referred to as finding a *maximum transversal* or *maximum matching*. Second, we discuss a method that permutes a matrix so that the product of the absolute value of the entries on the diagonal is maximized. Third, we discuss a variant of the second method, where the matrix is permuted so that the absolute value of the smallest entry on the diagonal is maximized.

Finally, we include a discussion on how to scale the entries of a matrix so that its diagonal entries are 1 in absolute value, and its off-diagonal entries are all less than or equal to 1 in absolute value.

In our experiments we found that, in general, the best results in terms of both preconditioner fill and convergence rates were obtained when matrices were permuted so as to maximize the product of the absolute value of the entries on the diagonal, and scaled so that the diagonal entries are 1 in absolute value, and off-diagonal entries are all less than or equal to 1 in absolute value. Also note that the timing behaviors mentioned in the discussions of the algorithms below are worst-case scenarios. The implementations in MC64 are efficient and the preprocessing phase is very fast, even for relatively dense matrices.

3.1. Finding a maximum transversal. Let $A = (a_{ij})$ be a general $n \times n$ matrix. Let $\mathcal{M}$ denote a set of at most n ordered index pairs $(i, j), 1 \le i, j \le n$, in which each row index i and each column index j appears at most once. $\mathcal{M}$ is called a *transversal* or *matching*. When $\mathcal{M}$ has maximum cardinality (n for structurally nonsingular matrices), $\mathcal{M}$ is called a *maximum transversal* or *maximum matching*.

In the case where $|\mathcal{M}| = n$, then $\mathcal{M}$ defines an $n \times n$ permutation matrix $Q = (q_{ij})$, where

$$\begin{cases} q_{ji} = 1 & \text{for } (i, j) \in \mathcal{M}, \\ q_{ji} = 0 & \text{otherwise}, \end{cases}$$

and thus AQ and QA are matrices with the transversal entries on the diagonal. Because our code is row oriented, we limit our discussion to row permutations, i.e., permutations of the form QA.

One of the options in MC64 uses the algorithm MC21 implemented by Duff [14], [15]. MC21 is a depth-first search algorithm with look-ahead; for a sparse matrix with τ nonzero entries, the algorithm has worst-case complexity of $\mathcal{O}(n\tau)$ but in practice exhibits $\mathcal{O}(n + \tau)$ behavior.

The use of such a reordering strategy is fundamental as the first step of permuting sparse reducible matrices to block triangular form. We mention that permuting to a zero-free diagonal has been examined before as a reordering strategy for preconditioning; see, for instance, [11] and [5]. Clearly, such a permutation will prove beneficial when, under the original ordering, there are zeros lying on the diagonal. However, in our experiments, we encountered few cases where finding a maximum transversal provided more benefit than finding a maximum product transversal.

3.2. Finding a maximum product transversal. In this subsection, we discuss permuting a matrix so that the product of the absolute value of the entries along

the diagonal is maximized. That is, we look for a permutation σ that maximizes

$$\prod_{i=1}^{n} |a_{\sigma(i)i}|. \tag{3.1}$$

This strategy, combined with the scalings mentioned below, was introduced in [36] for pivoting in dense Gaussian elimination.

First, this maximization problem is translated into a minimization problem. Let $a_i = \max_j |a_{ij}|$ denote the maximum value in row i of the matrix A. Define the matrix $C = (c_{ij})$ by

$$c_{ij} = \begin{cases} \log a_i - \log |a_{ij}| & \text{for } a_{ij} \neq 0, \\ \infty & \text{otherwise.} \end{cases}$$

Then maximizing (3.1) is equivalent to minimizing

$$\sum_{i=1}^{n} c_{\sigma(i)i}. \tag{3.2}$$

Minimizing the sum (3.2) is equivalent to finding a minimum weight perfect matching. In combinatorial optimization, this is known as the bipartite weighted matching problem, or linear assignment problem. MC64 uses a sparse variant of the bipartite weighted matching algorithm introduced in [36]. Fundamental to finding a minimum weight perfect matching is finding a shortest augmenting path, which in turn relies on a sparse variant of Dijkstra's algorithm [13]. For a full $n \times n$ matrix, the assignment problem can be solved in $\mathcal{O}(n^3)$ time; for a sparse matrix, the problem can be solved in $\mathcal{O}(n\tau \log n)$ time.

An important aspect of applying the bipartite weighted matching algorithm to the matrix C is the following result [21]: A perfect matching $\mathcal{M}$ has minimum weight if and only if there exist vectors $u = (u_i)$ and $v = (v_i)$, each of length n, such that

$$\begin{cases} u_i + v_j = c_{ij} & \text{for } (i,j) \in \mathcal{M}, \\ u_i + v_j \leq c_{ij} & \text{for } (i,j) \notin \mathcal{M}. \end{cases} \tag{3.3}$$

These vectors u and v are used in scaling the permuted matrix (see below).

3.3. Algorithm for finding a bottleneck transversal. In this subsection we consider a simple variation of the method discussed in the previous subsection. Here we are interested in finding a permutation of A such that the smallest absolute value on the diagonal is maximized. That is, we look for a permutation σ that maximizes

$$\min_{1 \leq i \leq n} |a_{\sigma(i)i}|, \tag{3.4}$$

and define the matrix $C = (c_{ij})$ by

$$c_{ij} = \begin{cases} a_i - |a_{ij}| & \text{for } a_{ij} \neq 0, \\ \infty & \text{otherwise.} \end{cases}$$

Then maximizing (3.4) is equivalent to minimizing

$$\max_{1 \leq i \leq n} c_{\sigma(i)i}. \tag{3.5}$$

Thus the maximum product transversal algorithm can be applied here, with only minor modifications, such as to replace the sum operation (3.2) by the max operation (3.5).

Note that this method regards only the smallest entry on the diagonal of the permuted matrix. For example, as mentioned in [19], consider a matrix having a row containing only one nonzero entry whose absolute value is the smallest in the matrix. Then the bottleneck transversal algorithm may return a transversal with small values on the diagonal. MC64 takes a somewhat different approach to avoid this problem; see [18], [19] for details.

Scaling the matrix prior to finding a bottleneck transversal also alleviates this problem. Ultimately, though, we note that we found few examples where bottleneck transversal permutations proved superior to maximum product transversal permutations.

3.4. Scaling. It is often beneficial to scale the matrices using the parameters u_i and v_j from (3.3) obtained in the weighted matching algorithm. To this end, define diagonal matrices D_1 and D_2 by

$$\begin{aligned} D_1 &= \operatorname{diag}(d_1^1, d_2^1, \ldots, d_n^1), \quad d_j^1 = \exp(v_j)/a_j, \\ D_2 &= \operatorname{diag}(d_1^2, d_2^2, \ldots, d_n^2), \quad d_i^2 = \exp(u_i). \end{aligned} \tag{3.6}$$

Then QD_1AD_2 is a matrix whose diagonal entries are 1 in absolute value, and whose off-diagonal entries are all less than or equal to 1, in absolute value. Such a matrix is referred to as an I-matrix in [36]. By simply changing the sign of rows (or columns) having a diagonal entry equal to -1 we obtain a matrix with unit diagonal. The eigenvalues of such a matrix lie in discs centered at 1, with radii equal to the sum of the absolute values of the off-diagonal entries in the corresponding row. Therefore, the less the matrix deviates from a diagonally dominant matrix, the more clustered the eigenvalues will be, around 1. In the ideal case, all the discs of such a matrix will have radii less than 1, and the matrix will be strictly rowwise diagonally dominant. In turn, this guarantees that ILU and AINV preconditioners will be well defined. It is also a favorable situation for Krylov subspace methods since, in particular, all the eigenvalues will have positive real part.

4. Description of test problems. In this section we describe the matrices that were used in the numerical experiments. Most of these matrices are available in the public domain [17], [12], [35]. They are representative of problems from a variety of applications, but they have in common the fact that they are difficult to solve with iterative methods. The matrices are listed in Table 1 below, together with some basic information. In Table 9 we report the estimate for the condition number returned by MATLAB (except for the last three problems, which are too large).

The first eight matrices are from chemical engineering and represent simulations of different chemical processes. While they are not large, these matrices are interesting because of the challenge they pose to iterative solvers. They are highly unstructured, very far from being structurally symmetric, with most of the diagonal entries equal to zero. In addition, they are highly indefinite and tend to be very ill conditioned. Matrices similar to these have been used to test ILU-type preconditioners in [11]. In that paper, MC21 was used to obtain a zero-free diagonal: The results were poor.

TABLE 1
Test problem information.

Application area: Chemical engineering			
Matrix	Description	Order	Nonzeros
WEST0655	Sixteen stage column section	655	2854
WEST0989	Seven stage column section	989	3537
WEST1505	Eleven stage column section	1505	5445
WEST2021	Fifteen stage column section	2021	7353
LHR01	Light hydrocarbon recovery	1477	18592
LHR02	Light hydrocarbon recovery	2954	37206
BAYER09	Chemical process simulation	3083	21216
BAYER10	Chemical process simulation	13436	94926
Application area: Economic models and linear programming			
Matrix	Description	Order	Nonzeros
MAHINDAS	Economic model of Victoria	1258	7682
ORANI678	Economic model of Australia	2529	90158
BP200	Simplex method basis matrix	822	3802
GEMAT11	Power flow in 2400 bus system —initial simplex method basis	4929	33185
GEMAT12	Power flow in 2400 bus system —basis after 100 iterations	4929	33111
Application area: Circuit modeling			
Matrix	Description	Order	Nonzeros
WATSON4a	Jacobian at step 4, 1 row added	468	2870
WATSON5a	Jacobian at step 4, 1 row added	1854	10848
CIRCUIT3	Jacobian from nonlinear DAE system	12127	48137
Application area: PDE problems			
Matrix	Description	Order	Nonzeros
SHERMAN2	Thermal simulation with steam injection	1080	23094
LNS3937	Linearized Navier–Stokes equations	3937	25407
UTM5940	Plasma physics, tokamak modeling	5940	83842
SLIDE	Solid deformation model (ALE3D)	20191	1192535
TWO-DOM	Solid deformation model (ALE3D)	22200	1188152
VENKAT25	2D unstructured Euler solver, time step=25	62424	1717792

The next five matrices come from the general area of mathematical economics and management. These matrices are also highly unstructured, nonsymmetric, indefinite with most diagonal entries equal to zero, and cause difficulties for preconditioned iterative methods.

The next three matrices arise in circuit design. The first two problems are described in [33] and are available from the Matrix Market [35]. The original matrices WATSON4 and WATSON5 are rectangular; as in [33], we appended one row at the bottom of these matrices to make them square (and we changed the names to WATSON4a and WATSON5a). The row vector used was $e_n^T = (0, \ldots, 0, 1)$. All diagonal entries are nonzero. The third circuit matrix was kindly provided by Wim Bomhof of Utrecht University; see [8]. This matrix has some zero diagonal entries. All three matrices are very sparse, and they exhibit a good deal of structural symmetry.

Finally, we included six matrices from the discretization of PDEs. Of these, SHERMAN2 does not present any difficulty for ILU preconditioning, but it has been reported by several researchers as being quite challenging for sparse approximate in-

verse preconditioners; see, e.g., [1], [6], [26], and [27]. Matrix LNS3937 has a zero diagonal block corresponding to the divergence constraint in the Navier–Stokes equations and is challenging for both ILU and approximate inverse techniques; see, respectively, [10] and [1]. Zero diagonal blocks induced by constraints also occur in the unstructured finite element matrices SLIDE and TWO-DOM, which were kindly provided by Ivan Otero (Lawrence Livermore National Laboratory). Matrix UTM5940 is fairly difficult to solve with ILU-type methods [10] and is even more so by sparse approximate inverse techniques. Matrix VENKAT25 was included because it is difficult for AINV. These last two matrices have no zero diagonal entries.

These matrices are just a selection from a larger set that was used for the tests; the chosen problems are representative of the results observed. As we will see, preprocessing makes all these problems solvable by iterative methods preconditioned with ILUT and AINV, and many of them even with ILU(0) or ILU(1) preconditioning. We found, however, several problems that could not be solved by the techniques employed in this paper. In these cases, the preconditioned iteration usually converged, but to a seriously inaccurate solution. Among these matrices we mention SHYY41 (also discussed in [10]), NNC666 and NNC1374, GRAHAM1, several of the FIDAP matrices, and some of the LHR0$*$c matrices from chemical engineering, all available in [12] or [35]. We tried to solve these problems by a *direct* method, namely, Gaussian elimination with partial pivoting, but again the computed solution was affected by large errors. The same happened when we used the *complete* factors computed with the direct method as preconditioners for Krylov subspace methods—a form of iterative refinement. Not surprisingly, these matrices are severely ill conditioned. One cannot blame an algorithm for being unable to produce accurate solutions to extremely ill conditioned problems. In this paper, we only present results for problems that could be solved with some accuracy.

5. Numerical experiments. In this section we report on numerical experiments aimed at assessing the impact of the MC64 permutation and scaling routines on the robustness and performance of preconditioned Krylov subspace methods. All algorithms were implemented in Fortran77 using double precision arithmetic. The codes were compiled by `f77` with the `-O3` optimization option. Test runs were performed on a Sun Ultra 5 workstation for all test problems except the last three, for which one 250 MHz processor of an SGI Origin 2000 computer was used.

Three different Krylov subspace solvers were tried: BiCGSTAB [41], GMRES [40], and TFQMR [23]. While the three algorithms performed similarly on most problems, BiCGSTAB was found to be somewhat better than the others overall. Therefore, we will present results for BiCGSTAB only. The preconditioners used are diagonal (Jacobi) scaling, ILU(0), ILU(1), ILUT, and AINV. Besides these, we performed experiments also with ILUTP [39] and SPAI [26]. In all cases, right preconditioning was used. An artificial right-hand side b was used in the runs, so that the system $Ax = b$ had the known solution $x = (x_i)$ with $x_i = i,\ 1 \le i \le n$. Runs were performed also with other choices of b, with similar results. The initial guess was always the zero vector, $x_0 = 0$. The iteration was stopped when the ℓ_2-norm of the initial residual was reduced by at least eight orders of magnitude, or when a maximum number of iterations maxit $= \min\{n, 2000\}$ was reached.

Some comments on the accuracy of the approximate solutions corresponding to this stopping criterion are in order. As reported in Table 9, many of the test matrices are very ill conditioned, and a small residual does not necessarily guarantee a small error. Nevertheless, the stopping criterion adopted was sufficient to produce

solutions with good relative accuracy except in a few cases (BAYER09, GEMAT12, CIRCUIT3), where some of the components of the solution were affected by large errors. For these cases, reducing the stopping tolerance from 10^{-8} to 10^{-12} resulted in accurate solutions, at the expense of an increase in the number of iterations of roughly 30%. However, all the results presented in the subsequent sections correspond to the stopping tolerance 10^{-8}.

5.1. Testing reliability. Here we present results aimed at assessing the impact of the preprocessing on the reliability of preconditioned BiCGSTAB. Only iteration counts are reported. In Table 2 we report the results of runs using diagonal (Jacobi) preconditioning for the original matrix (under "orig") and for different preprocessings: the basic maximum transversal (under "mc21"), the bottleneck transversal (under "bt"), the maximum product transversal without scalings (under "mpd"), and with scalings (under "mps").

TABLE 2
Iteration count, Jacobi preconditioning.

Matrix	orig.	mc21	bt	mpd	mps
WEST0655	‡	†	†	†	†
WEST0989	‡	†	†	239	171
WEST1505	‡	†	†	536	570
WEST2021	‡	†	†	342	455
LHR01	‡	†	†	421	238
LHR02	‡	†	†	1195	1109
BAYER09	‡	†	†	†	244
BAYER10	‡	†	†	†	†
MAHINDAS	‡	†	†	348	137
ORANI678	‡	†	1133	217	196
BP200	‡	†	†	†	†
GEMAT11	‡	†	†	†	†
GEMAT12	‡	†	*bd*	†	†
WATSON4a	467	467	467	222	183
WATSON5a	†	†	†	†	†
CIRCUIT3	‡	†	†	†	†
SHERMAN2	†	†	†	466	†
LNS3937	‡	†	†	†	†
UTM5940	†	†	†	†	†
SLIDE	‡	†	†	†	†
TWO-DOM	‡	†	†	†	†
VENKAT25	†	†	†	†	†

A "‡" means that the preconditioner was not defined, due to zeros on the main diagonal. A "†" means failure to converge within the maximum number of allowed iterations. A "*bd*" denotes a breakdown in the BiCGSTAB acceleration. Notice that mc21 leaves matrices WATSON4a, WATSON5a, SHERMAN2, UTM5940, and VENKAT25 unchanged, since these matrices have no zero diagonal entries.

The only problem that can be solved without any preprocessing is the small circuit matrix WATSON4a, which requires a number of iterations almost equal to the order of the matrix. While the maximum and bottleneck transversals lead to virtually no improvements, the maximum product transversals result in convergence in nine cases. In particular, six out of the eight chemical engineering problems and both economics problems can be solved using Jacobi preconditioning in connection with

the maximum product transversal and associated scalings. Notice that with mps, Jacobi preconditioning simply has the effect of changing the sign of those matrix columns for which the corresponding diagonal entry is -1.

The next three tables report the results obtained for various ILU-type preconditioners. As is well known, these preconditioners are sensitive to the ordering of the equations and unknowns. Therefore, after applying the various preprocessings, we also reorder the matrix with a symmetric permutation. Consider for instance mps preprocessing. Indicated with D_1 and D_2, respectively, the row and column scalings associated with the maximum product transversal and with Q the corresponding row permutation that permutes the scaled matrix to a zero-free diagonal form, the symmetric permutations are based on the adjacency graph of the symmetric matrix $\hat{A} + \hat{A}^T$, where $\hat{A} = QD_1AD_2$. Once the permutation matrix P has been determined, one solves the following linear system:

$$(P^T Q D_1 A D_2 P)y = P^T Q D_1 b.$$

The solution of the original linear system is then $x = D_2 P y$. The preconditioner calculation is carried out on the scaled and reordered matrix

$$\tilde{A} = P^T Q D_1 A D_2 P.$$

While the purpose of the scalings D_1, D_2 and of the one-sided permutation Q is to improve stability, the main effect of the symmetric permutation P is to increase the effectiveness of the preconditioner by making it more accurate. For structurally symmetric matrices that are numerically unsymmetric and far from diagonally dominant, it was found in [4] that the performance and robustness of ILU-type preconditioners was generally improved by the reverse Cuthill–McKee ordering [16], denoted "rcm" in the tables (see column "SO"). Thus, we used rcm as the default ordering for ILU preconditioners. Although it is not always the best ordering, rcm gave good results in a majority of cases, in good agreement with the conclusions in [4]. Whenever rcm performed poorly we switched to another symmetric ordering, like multiple minimum degree [32] (denoted "mmd") or generalized nested dissection [31] (denoted "gnd"). Occasionally, we found that no symmetric reordering was the best option (denoted "no" in the tables). In these tables, a "‡" indicates a failure due to pivot breakdown; i.e., a zero or exceedingly small pivot was encountered in the course of the incomplete factorization.

The results for ILU(0) preconditioning are reported in Table 3. Again, using just mc21 is ineffective, with the only exception of matrix ORANI678. Results for the bottleneck transversal are not much better. However, the robustness of ILU(0) is greatly improved when the maximum product transversal is used. No pivot breakdown occurs, and all but five problems can be solved. The five failures are due to very slow convergence except for CIRCUIT3, for which the ILU(0) factors are unstable. Using different symmetric reorderings for these problems did not help.

We notice that there are two matrices, SHERMAN2 and VENKAT25, which can be easily solved with ILU(0) preconditioning without any preprocessing. Indeed, using mps results in a slight deterioration in the rate of convergence. Clearly, if a given combination of preconditioner and Krylov subspace solver gives good results, the use of preprocessing is not necessary and should not be used. We further observe that in several cases, mpd (without scalings) gives better results than mps. The same phenomenon occurs also for Jacobi and ILU(1) preconditioning (see Tables 2 and 4).

TABLE 3
Iteration count, ILU(0) preconditioning.

Matrix	SO	orig.	mc21	bt	mpd	mps
WEST0655	gnd	‡	‡	‡	157	144
WEST0989	gnd	‡	‡	‡	51	60
WEST1505	mmd	‡	‡	‡	1498	903
WEST2021	mmd	‡	‡	‡	136	162
LHR01	rcm	‡	†	†	52	49
LHR02	rcm	‡	‡	‡	210	62
BAYER09	rcm	‡	‡	‡	32	42
BAYER10	rcm	‡	‡	‡	†	†
MAHINDAS	rcm	‡	‡	167	34	32
ORANI678	rcm	‡	83	50	28	21
BP200	mmd	‡	‡	126	510	603
GEMAT11	rcm	‡	†	†	153	140
GEMAT12	rcm	‡	†	†	702	838
WATSON4a	rcm	131	131	127	60	89
WATSON5a	rcm	†	†	†	†	†
CIRCUIT3	rcm	‡	‡	†	†	†
SHERMAN2	no	8	8	†	12	11
LNS3937	rcm	‡	†	†	†	†
UTM5940	no	†	†	†	†	†
SLIDE	rcm	‡	†	628	335	335
TWO-DOM	rcm	‡	†	†	419	513
VENKAT25	rcm	90	90	†	88	117

TABLE 4
Iteration count, ILU(1) preconditioning.

Matrix	SO	orig.	mc21	bt	mpd	mps
WEST0655	gnd	‡	‡	‡	‡	†
WEST0989	rcm	‡	‡	‡	32	38
WEST1505	rcm	‡	‡	‡	37	45
WEST2021	rcm	‡	‡	‡	68	74
LHR01	rcm	‡	‡	†	64	58
LHR02	rcm	‡	‡	‡	143	104
BAYER09	rcm	‡	‡	‡	10	14
BAYER10	rcm	‡	‡	‡	1176	1707
MAHINDAS	rcm	‡	‡	108	9	16
ORANI678	rcm	‡	‡	69	14	13
BP200	mmd	‡	58	36	94	†
GEMAT11	rcm	‡	‡	107	43	46
GEMAT12	rcm	‡	‡	†	183	55
WATSON4a	rcm	126	126	92	27	26
WATSON5a	no	†	†	†	†	932
CIRCUIT3	rcm	‡	‡	†	†	†
SHERMAN2	rcm	8	8	†	4	4
LNS3937	gnd	379	*bd*	†	264	355
UTM5940	no	151	151	†	168	119
SLIDE	mmd	‡	‡	†	226	259
TWO-DOM	mmd	‡	‡	†	151	155
VENKAT25	rcm	56	56	†	57	81

In Table 4 we report the results obtained with ILU(1) preconditioning. Again, mc21 and bt offer little benefit, whereas the use of mpd or mps allows all but three problems to be solved. With mpd we had one failure due to pivot breakdown (WEST0655), one due to slow convergence (WATSON5a), and one due to unstable ILU factors (CIRCUIT3). For mps, unstable ILU solves were the cause of all three failures. Notice that in a few cases, ILU(1) performs worse than ILU(0). However, all PDE problems can be solved with ILU(1) when either mpd or mps is used, with mpd giving somewhat better results on average.

TABLE 5
Iteration count, ILUT preconditioning.

Matrix	SO	$tol,\ p$	orig.	mc21	bt	mpd	mps
WEST0655	gnd	$10^{-1}, 5$	‡	‡	‡	65	37
WEST0989	rcm	$10^{-1}, 5$	‡	‡	470	21	9
WEST1505	mmd	$10^{-1}, 5$	‡	‡	†	513	53
WEST2021	rcm	$10^{-1}, 5$	‡	‡	‡	110	34
LHR01	rcm	$10^{-1}, 5$	‡	‡	‡	252	41
LHR02	rcm	$10^{-1}, 5$	‡	‡	‡	‡	78
BAYER09	rcm	$10^{-1}, 5$	‡	‡	†	†	14
BAYER10	mmd	$10^{-4}, 20$	‡	‡	‡	‡	65
MAHINDAS	rcm	$10^{-1}, 5$	‡	†	959	8	9
ORANI678	rcm	$10^{-1}, 5$	‡	†	62	12	14
BP200	mmd	$10^{-1}, 5$	‡	†	28	17	11
GEMAT11	rcm	$10^{-1}, 5$	‡	‡	†	1471	303
GEMAT12	rcm	$10^{-1}, 5$	‡	‡	†	354	306
WATSON4a	rcm	$10^{-1}, 5$	457	457	457	208	31
WATSON5a	rcm	$10^{-5}, 25$	6	6	6	6	6
CIRCUIT3	rcm	$10^{-4}, 20$	80	‡	516	36	90
SHERMAN2	rcm	$10^{-1}, 5$	37	37	†	8	10
LNS3937	rcm	$10^{-3}, 10$	‡	‡	†	776	24
UTM5940	no	$10^{-4}, 20$	164	164	†	302	141
SLIDE	rcm	$10^{-1}, 5$	‡	†	†	†	491
TWO-DOM	rcm	$10^{-1}, 5$	‡	†	†	†	494
VENKAT25	rcm	$10^{-2}, 5$	†	†	‡	†	98

It is already clear from these results that the reliability of preconditioned iterative solvers is greatly enhanced by the use of one-sided permutations aimed at placing large entries on the main diagonal, even when simple preconditioners like ILU(0) and ILU(1) are used. There are, however, a few hard problems for which this approach does not work. Additional robustness can be achieved by using a drop tolerance.

In Table 5 we present results for the popular dual-threshold ILUT(tol, p) preconditioner. Our strategy for the experiments was the following. We used the default parameters $tol = 10^{-1}$, $p = 5$ with rcm as the basic symmetric reordering. Whenever we found that this combination produced poor results, we switched to a different symmetric reordering until a good one was found (in the following order: mmd, gnd, no reordering). When this didn't work we changed the ILUT parameters by decreasing tol and, if necessary, increasing p. We do not claim that this leads to an optimal, or even good preconditioning strategy: Rather, the purpose of these experiments is to demonstrate that iterative solvers can be made reliable without much need for fine-tuning. We emphasize that the values $tol = 10^{-1}$ and $p = 5$ are not typical for ILUT. Usually, a much smaller drop tolerance and a much larger value of p are used,

particularly for hard problems: see, e.g., [10]. In other words, we chose to use a very sparse preconditioner, in many cases even sparser than the original matrix. We made this choice deliberately, with the intent to show that most problems become very easy to solve once the preprocessing phase is applied. In general, better results can be obtained with a different choice of the parameters.

The results in Table 5 show that with mps, all problems can be solved with ILUT preconditioning. In a majority of cases the basic combination of rcm reordering with $tol = 10^{-1}$ and $p = 5$ was sufficient to solve the problem. In some cases, an alternative symmetric reordering and/or additional fill-in had to be used in order to achieve convergence. The hardest problems for ILUT appear to be BAYER10, CIRCUIT3, and UTM5940. Notice that complementing the maximum product transversal with scalings has the effect of improving the reliability and performance of ILUT-preconditioned BiCGSTAB in nearly all cases. The only serious exception to this rule occurs for CIRCUIT3, for which the number of iterations increases from 36 to 90.

TABLE 6
Iteration count, AINV preconditioning.

Matrix	SO	*tol*	orig.	mc21	bt	mpd	mps
WEST0655	no	10^{-1}	‡	‡	‡	†	176
WEST0989	mmd	10^{-1}	‡	‡	‡	141	32
WEST1505	mmd	10^{-2}	‡	‡	†	1693	37
WEST2021	mmd	10^{-2}	‡	‡	†	148	8
LHR01	mmd	10^{-1}	‡	†	‡	251	74
LHR02	mmd	10^{-1}	‡	‡	‡	385	127
BAYER09	mmd	10^{-1}	‡	‡	†	15	8
BAYER10	no	3.5×10^{-2}	‡	‡	‡	781	43
MAHINDAS	mmd	10^{-1}	‡	‡	29	15	7
ORANI678	no	10^{-1}	‡	†	251	10	9
BP200	mmd	5×10^{-2}	‡	‡	20	‡	5
GEMAT11	mmd	10^{-1}	‡	‡	†	802	230
GEMAT12	mmd	10^{-1}	‡	‡	†	†	389
WATSON4a	mmd	10^{-1}	21	21	21	14	20
WATSON5a	mmd	10^{-3}	51	51	51	51	114
CIRCUIT3	no	10^{-2}	†	‡	†	268	43
SHERMAN2	mmd	10^{-1}	†	†	†	50	14
LNS3937	gnd	10^{-1}	‡	†	†	†	112
UTM5940	gnd	10^{-1}	1268	1268	†	1388	406
SLIDE	mmd	10^{-1}	†	†	†	603	306
TWO-DOM	mmd	10^{-1}	†	†	965	491	224
VENKAT25	mmd	10^{-1}	†	†	‡	385	411

In Table 6 we present results for the drop-tolerance-based sparse approximate inverse AINV preconditioner. Like ILU-type preconditioners, AINV is sensitive to the ordering. In [9] and [7] it was shown that mmd is generally a good ordering for AINV, with gnd a close second. On the other hand, rcm cannot be recommended in general. Thus, the default parameters used were mmd reordering and $tol = 10^{-1}$. For a few hard problems, it was necessary to switch to gnd reordering (or no reordering) and to reduce the drop tolerance. However, no effort was made to tune the parameters for optimal performance. As for the ILU preconditioner, failures due to pivot breakdowns are denoted by "‡" and failures due to slow convergence by "†."

TABLE 7
Test results for ILUT preconditioning.

Matrix	NSO-time	SO-time	P-time	ρ	Its	It-time	Tot-time
WEST0655	0.009	0.006	0.005	1.22	37	0.110	0.130
WEST0989	0.011	0.004	0.004	0.967	9	0.056	0.075
WEST1505	0.018	0.017	0.007	0.916	53	0.323	0.365
WEST2021	0.024	0.009	0.011	0.997	34	0.333	0.378
LHR01	0.091	0.019	0.013	0.349	41	0.369	0.494
LHR02	0.214	0.042	0.026	0.346	78	1.54	1.82
BAYER09	0.113	0.025	0.017	0.401	14	0.280	0.440
BAYER10	0.799	0.556	0.189	1.22	65	6.63	8.19
MAHINDAS	0.019	0.009	0.009	0.613	9	0.080	0.117
ORANI678	0.185	0.132	0.077	0.087	14	0.452	0.850
BP200	0.009	0.016	0.004	0.856	11	0.049	0.078
GEMAT11	0.079	0.038	0.038	0.632	303	8.05	8.21
GEMAT12	0.097	0.038	0.039	0.662	306	8.26	8.43
WATSON4a	0.006	0.004	0.004	0.799	31	0.063	0.078
WATSON5a	0.015	0.031	0.031	1.85	6	0.128	0.211
CIRCUIT3	0.127	0.106	0.610	2.02	90	7.08	7.88
SHERMAN2	0.053	0.021	0.020	0.228	10	0.111	0.206
LNS3937	0.131	0.030	0.129	2.59	24	0.973	1.27
UTM5940	0.223	—	1.50	3.13	141	14.5	16.2
SLIDE	0.977	0.938	0.588	0.090	491	67.1	69.6
TWO-DOM	0.976	0.956	0.595	0.100	494	69.5	72.0
VENKAT25	1.43	1.28	2.59	1.05	98	48.5	53.8

As for ILUT, we see that mps preprocessing enables BiCGSTAB preconditioned with AINV to solve all our test problems. For instance, problem SHERMAN2, which is notoriously difficult for approximate inverse methods, becomes very easy to solve. We also see that in most cases scalings improve the performance of the maximum product transversal. More importantly, scalings improve reliability, as shown by the fact that there are four problems that cannot be solved with mpd alone.

5.2. Timing results. From the experiments performed so far it appears that mps preprocessing dramatically increases the reliability and performance of Krylov subspace methods preconditioned with drop-tolerance-based preconditioners, like ILUT and AINV. The question remains to be addressed of how expensive the preprocessing phase is in practice. This is an important question because this preprocessing phase, dependent as it is on the numerical values of the matrix coefficients, is not easy to amortize, except for the situation where a sequence of linear systems with the same coefficient matrix and different right-hand sides must be solved. If the coefficient matrix changes, the preprocessing has to be applied anew.

Timing results relative to ILUT and AINV are presented in Tables 7 and 8, respectively. We report the time to perform the mps preprocessing (under NSO-time), the time for the symmetric reordering (under SO-time), the time for computing the preconditioner (under P-time), the time to perform the iterative solve phase (It-time), and, in the last column, the total solution time (under Tot-time). Timings are in seconds, and were measured with the `dtime` function. We also report the density ρ of the preconditioner, defined as the ratio between the number of nonzeros in the preconditioner over the number of nonzeros in the original coefficient matrix A. The number of iterations to converge (Its) is also included. The parameters and symmetric

TABLE 8
Test results for AINV preconditioning.

Matrix	NSO-time	SO-time	P-time	ρ	Its	It-time	Tot-time
WEST0655	0.009	—	0.122	10.2	176	1.54	1.67
WEST0989	0.011	0.010	0.018	2.36	32	0.175	0.214
WEST1505	0.018	0.017	0.094	5.92	37	0.499	0.628
WEST2021	0.024	0.023	0.203	7.87	8	0.228	0.478
LHR01	0.091	0.108	0.499	2.67	74	1.56	2.26
LHR02	0.214	0.231	1.10	3.00	127	5.94	7.49
BAYER09	0.113	0.128	0.228	1.73	8	0.270	0.739
BAYER10	0.799	—	4.67	5.96	43	11.2	16.7
MAHINDAS	0.019	0.139	0.061	1.27	7	0.082	0.301
ORANI678	0.185	—	0.776	0.146	9	0.332	1.29
BP200	0.009	0.016	0.026	2.46	5	0.046	0.097
GEMAT11	0.079	0.057	0.172	1.62	230	8.57	8.88
GEMAT12	0.097	0.058	0.224	2.04	389	15.8	16.2
WATSON4a	0.006	0.014	0.011	1.01	20	0.049	0.080
WATSON5a	0.015	0.072	0.167	1.77	114	1.34	1.59
CIRCUIT3	0.127	—	3.82	1.89	43	3.43	7.38
SHERMAN2	0.053	0.071	0.094	0.352	14	0.148	0.366
LNS3937	0.131	0.059	0.971	5.89	112	7.05	8.22
UTM5940	0.223	0.158	2.80	2.77	406	45.0	48.2
SLIDE	0.977	1.39	12.0	0.273	306	50.4	64.8
TWO-DOM	0.976	1.68	11.8	0.254	224	36.1	50.6
VENKAT25	1.43	1.40	20.1	0.838	411	184.	207.

reorderings are the same as those used for the runs in Tables 5 and 6.

Concerning the cost of the preprocessing, we see that it is usually small compared with the overall solution costs. For the larger problems the cost of preprocessing, including the time to apply the symmetric reordering, is negligible compared with the total solve time. We also see that in most cases the time for the nonsymmetric permutation and scaling is comparable to the time required for the symmetric permutations, which are based on the (unweighted) graph of the matrix only.

The performance of ILUT preconditioning is impressive: In most cases, rapid convergence is obtained with a very sparse preconditioner. For the few cases where a high number of iterations is required to achieve convergence (GEMAT11, GEMAT12, SLIDE, and TWO-DOM) we found that much faster convergence and smaller timings result if more fill-in is allowed. For instance, using ILUT$(10^{-3}, 10)$ on GEMAT11 results in convergence in 40 iterations with preconditioner density $\rho = 1.46$, and the total solution time becomes 1.81 seconds.

It is our opinion that when used with mps preprocessing, ILUT preconditioning has the potential to become a useful tool for solving linear equations from chemical engineering applications. Because rapid convergence can be achieved with very sparse incomplete factorization preconditioners, this approach may be competitive with sparse direct methods. This approach also has considerable potential for matrices arising from economic modeling and management science, although we have less experience with such problems.

On the other hand, it is unclear whether this approach is really useful for matrices arising in circuit simulations. Sparse direct solvers suffer very little fill-in on such problems when a good ordering is adopted. Hence, for iterative solvers to be

competitive with direct methods they must converge extremely fast with very sparse preconditioners. From our test runs, this seems to be difficult to achieve unless the incomplete factorization closely approaches a complete one. We note that in [8] a combined direct/iterative method is described in which the preconditioned iteration is applied to a relatively small Schur complement matrix. This algorithm appears to be competitive with sparse direct solvers, so there is a role for iterative methods in this area.

In terms of timings and storage requirements, AINV (Table 8) is generally more expensive than ILUT, but it should be kept in mind that an important advantage of AINV, its parallelizability, is not captured by these one-processor experiments. There are a few cases where the density of the preconditioner is rather high. Sparser preconditioners can be obtained by using a larger value of *tol*, but this may slow down convergence considerably. This is especially true for the chemical engineering problems. On the other hand, we found that for the PDE problems faster convergence and smaller overall timings can be obtained by allowing more fill in the preconditioner. The same applies to the GEMAT$*$ problems. Notice the good performance of AINV on the matrices from economics, MAHINDAS and ORANI678. As for the circuit matrices, similar remarks as for ILUT apply. The main point here is that mps preprocessing dramatically increases the reliability of both ILUT and AINV preconditioning, therefore considerably widening the range of applicability of such techniques.

5.3. Further analysis of the results. In order to have a better understanding of the effect of the MC64 permutations and scalings used for preprocessing, we collected the statistics presented in Table 9. Under "condest" we report the condition number (estimated with the MATLAB function `condest`) for the original matrix and for the matrix scaled by the row and column scalings associated with mps preprocessing. The condition number could not be estimated for the three largest matrices.

Under "d.d. rows" we report the number of (weakly) diagonally dominant rows in the original matrix, in the matrix permuted with the maximum product transversal (mpd), and in the matrix scaled and permuted with mps. Under "d.d. cols" similar statistics are reported relative to the number of (weakly) diagonally dominant columns.

The statistics show that the chemical engineering matrices greatly benefit from the permutations and scalings: The number of diagonally dominant rows and columns is greatly increased, and ill conditioning is drastically reduced. Analogous remarks apply to problems MAHINDAS, ORANI678, BP200, and GEMAT11. Hence, it is not surprising that preconditioned iterative methods perform well on these problems when mps preprocessing is used.

For problems GEMAT12 and CIRCUIT3, the scalings have the effect of increasing the condition number. However, for GEMAT12 the number of weakly diagonally dominant rows and columns is greatly increased by the mpd preprocessing. The increase is somewhat smaller when scalings are used, but it is still a significant improvement. This makes the preconditioner computation stable, and therefore the net effect is positive. Matrices WATSON4a and WATSON5a are better conditioned after scaling and have a greater fraction of diagonally dominant rows after mpd. However, the scalings have the effect of reducing the number of diagonally dominant rows as compared with using mpd alone. Perhaps this explains why using mps often gives worse results than using mpd alone for these matrices (see Tables 3, 5, and 6).

TABLE 9
Conditioning and diagonal dominance statistics.

	condest		d.d. rows			d.d. cols		
Matrix	orig.	scaled	orig.	mpd	mps	orig.	mpd	mps
WEST0655	1.6E+12	2.0E+04	4	321	355	3	260	340
WEST0989	5.7E+12	1.9E+04	2	638	666	0	412	641
WEST1505	9.0E+12	2.5E+04	2	957	1004	0	619	969
WEST2021	7.5E+12	2.4E+04	2	1256	1340	0	808	1309
LHR01	5.4E+06	4.4E+04	20	323	438	0	766	710
LHR02	8.2E+06	5.5E+04	40	626	856	0	1532	1409
BAYER09	2.3E+21	1.1E+04	1	1610	1733	1	1737	2168
BAYER10	3.8E+15	1.5E+05	2	4653	7048	0	7081	7501
MAHINDAS	1.0E+13	5.0E+03	2	910	896	0	635	762
ORANI678	1.0E+07	1.1E+06	72	1866	1826	0	1653	1692
BP200	8.9E+08	3.8E+03	0	317	317	1	368	480
GEMAT11	3.7E+08	6.8E+06	2	1508	1536	2	1409	1238
GEMAT12	3.7E+08	1.2E+13	1	1465	1213	1	1398	1298
WATSON4a	8.8E+10	8.9E+07	161	374	270	377	161	261
WATSON5a	5.5E+07	3.2E+06	187	1264	1099	1268	187	332
CIRCUIT3	3.6E+07	3.0E+08	7865	7503	8245	7354	7650	8227
SHERMAN2	1.4E+12	3.3E+03	634	204	292	74	560	460
LNS3937	1.0E+17	1.9E+04	509	1283	689	307	892	701
UTM5940	1.9E+09	7.6E+08	762	915	882	925	766	926
SLIDE	n/a	n/a	1330	1203	1885	1201	1277	1274
TWO-DOM	n/a	n/a	2917	4627	4727	2917	4627	4700
VENKAT25	n/a	n/a	0	0	0	0	0	0

For the matrices arising from PDE problems, the permutations and scalings are generally beneficial. For VENKAT25, it is not clear from the results reported in Table 9 whether the preprocessing does any good. However, we know that ILUT and AINV benefit from the preprocessing. Although no row or column became diagonally dominant after the permutations and scalings, we found that the Gerschgorin bounds on the real and imaginary parts of the eigenvalues were one order of magnitude smaller after mps preprocessing, suggesting that the scaled and permuted matrix is not as far from diagonally dominant and has a more clustered spectrum than the original matrix.

5.4. Experiments with ILUTP and SPAI. We also experimented with the ILUTP preconditioner [39], which is generally more reliable than ILUT. However, we found that ILUTP preconditioning applied to the original matrices, or even to those permuted with mc21, is hardly better than ILUT, unless very large amounts of fill are allowed. Moreover, we found several problems that could not be solved by ILUTP even with very large amounts of fill ($tol = 0$, $p = 30$). These are WEST0655, LHR01, LHR02, BAYER09, BAYER10, BP200, and LNS3937. The failure of ILUTP was due to pivot breakdowns in all cases, except for LNS3937, where it appeared to be due to unstable ILU factors. It is mentioned in [10] that ILUTP with row pivoting (rather than column pivoting) is able to solve LHR01, but it took 134 iterations of GMRES(50) and a rather dense preconditioner. Using ILUT with mps, we can solve LHR01 in 41 iterations with a very sparse preconditioner (see Table 7). On the other hand, none of the techniques used in [10] succeeded in solving problem LNS3937, which is easily solved by ILUT with mps preprocessing. Hence, based on our experiments,

using standard ILUT with mps preprocessing is a more robust approach than using ILUTP alone on the original problem.

When ILUTP is used in combination with mps it gives good results, sometimes better than those obtained with ILUT for the same choice of the parameters. However, the use of column pivoting makes ILUTP slightly more expensive than ILUT, and it is unclear whether ILUTP is worth using with mps preprocessing.

Additional experiments were performed with the sparse approximate inverse preconditioner SPAI [26]. This technique is based on adaptive Frobenius norm minimization. Because the SPAI preconditioner is not factorized, it is insensitive to the ordering of the equations and unknowns. However, it is sensitive to scalings. We tested SPAI on the original matrices and with mps preprocessing. We found that overall, the use of scalings improves the reliability and performance of SPAI. This was especially true for the chemical engineering matrices; none of these matrices could be solved with SPAI, even with generous amounts of fill, but all of them could be solved after applying mps preprocessing. The same happened with MAHINDAS, BP200, SHERMAN2, LNS3937, and UTM5940.

In [1, p. 112], the authors give results for LNS3937 using a parallel implementation of BiCGSTAB preconditioned with SPAI. This took 1942 iterations with an approximate inverse containing 1588045 nonzeros, corresponding to $\rho = 61.3$! The total solution time was 567.3 seconds using 16 processors of a Cray T3E. Using SPAI with mps preprocessing, we can solve LNS3937 in 660 iterations with a preconditioner containing 150270 nonzeros, corresponding to $\rho = 5.9$, for a total solution time of 88.5 seconds on a Sun Ultra 5 workstation. Nevertheless, SPAI is outperformed by ILU(1), ILUT, and AINV used in combination with mps. Furthermore, SPAI failed on GEMAT12 and CIRCUIT3 (with or without mps). Finally, we found that mps had a detrimental effect on the convergence rate obtained with SPAI applied to matrices SLIDE, TWO-DOM, and VENKAT25.

6. Conclusions. The experiments presented in this paper indicate that the reliability and performance of Krylov subspace methods preconditioned with standard incomplete factorizations can be dramatically enhanced by means of nonsymmetric permutations and scalings aimed at placing large entries on the main diagonal. A similar effect is observed for other preconditioning techniques, such as factorized sparse approximate inverses. This preprocessing phase is inexpensive, both in absolute terms and relative to the total solution costs.

Of the heuristics considered in this paper, the maximum product transversal algorithm gave the best results. With this preprocessing, many of the diagonal entries are large relative to the off-diagonal ones, and the matrix is closer to being diagonally dominant. This not only has a stabilizing effect on the computation of the preconditioner, but also results in preconditioners of high quality in terms of convergence rates in many cases. When used in combination with scalings, which in most cases improve the conditioning of the problem, preconditioners based on drop tolerances (like ILUT and AINV) become quite reliable.

Much work remains to be done before preconditioned iterative methods can become a viable alternative to direct methods in areas such as chemical engineering, economics and management, circuit design, etc. Nevertheless, it is now at least conceivable to use iterative methods in such areas, and fair comparisons with direct solvers can be established. In addition to this, our experiments suggest that nonsymmetric permutations and scalings can be useful to improve the performance of iterative solvers even in areas where these are already widely used, such as PDE problems.

Acknowledgments. We would like to thank Iain Duff and Jacko Koster for providing us with the MC64 subroutines. We are indebted to Jane Cullum, Iain Duff, Daniel Szyld, and two anonymous referees for their constructive criticism of an early draft of the paper. This work was performed while the second author was a Graduate Research Assistant at Los Alamos National Laboratory (LANL): The hospitality and support of LANL are greatly appreciated.

REFERENCES

[1] S. T. Barnard, L. M. Bernardo, and H. D. Simon, *An MPI implementation of the SPAI preconditioner on the T3E*, Int. J. High. Perf. Comput. Appl., 13 (1999), pp. 107–123.
[2] M. Benzi, J. K. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comput., 22 (2000), pp. 1318–1332.
[3] M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149.
[4] M. Benzi, D. B. Szyld, and A. van Duin, *Orderings for incomplete factorization preconditioning of nonsymmetric problems*, SIAM J. Sci. Comput., 20 (1999), pp. 1652–1670.
[5] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems,* SIAM J. Sci. Comput., 19 (1998), pp. 968–994.
[6] M. Benzi and M. Tůma, *A comparative study of sparse approximate inverse preconditioners*, Appl. Numer. Math., 30 (1999), pp. 305–340.
[7] M. Benzi and M. Tůma, *Orderings for factorized sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1851–1868.
[8] W. Bomhof and H. A. van der Vorst, *A parallel linear system solver for circuit simulation problems*, Numer. Linear Algebra Appl., submitted.
[9] R. Bridson and W.-P. Tang, *Ordering, anisotropy and factored sparse approximate inverses*, SIAM J. Sci. Comput., 21 (1999), pp. 867–882.
[10] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414.
[11] H. N. Cofer and M. A. Stadtherr, *Reliability of iterative linear equations solvers in chemical process simulation*, Computers Chem. Engrg., 20 (1996), pp. 1123–1132.
[12] T. Davis, *University of Florida Sparse Matrix Collection*, University of Florida, Gainesville, FL, available online from http://www.cise.ufl.edu/∼davis/sparse/.
[13] E. W. Dijkstra, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269–271.
[14] I. S. Duff, *On algorithms for obtaining a maximum transversal*, ACM Trans. Math. Software, 7 (1981), pp. 315–330.
[15] I. S. Duff, *Algorithm* 575: *Permutations for a zero-free diagonal*, ACM Trans. Math. Software, 7 (1981), pp. 387–390.
[16] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Clarendon Press, Oxford, UK, 1986.
[17] I. S. Duff, R. G. Grimes, and J. G. Lewis, *The Rutherford–Boeing Sparse Matrix Collection*, Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, Chilton, UK, 1997.
[18] I. S. Duff and J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 889–901.
[19] I. S. Duff and J. Koster, *On Algorithms for Permuting Large Entries to the Diagonal of a Sparse Matrix*, Technical Report RAL-TR-99-030, Rutherford Appleton Laboratory, Chilton, UK, 1999.
[20] I. S. Duff and G. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.
[21] J. Edmonds, *Maximum matching and a polyhedron with* 0,1 *vertices*, J. Res. Nat. Bur. Standards Sect. B, 69B (1965), pp. 125–130.
[22] H. C. Elman, *A stability analysis of incomplete LU factorizations*, Math. Comp., 47 (1986), pp. 191–217.
[23] R. W. Freund, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Comput., 14 (1993), pp. 470–482.
[24] M. Gilli and G. Pauletto, *Krylov methods for solving models with forward-looking variables*, J. Econom. Dynam. Control, 22 (1998), pp. 1275–1289.
[25] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.
[26] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*,

SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
[27] N. I. M. Gould and J. A. Scott, *Sparse approximate-inverse preconditioners using norm-minimization techniques*, SIAM J. Sci. Comput., 19 (1998), pp. 605–625.
[28] S. A. Kharchenko, L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin, *A Robust AINV-Type Preconditioning Method for Constructing Sparse Approximate Inverse Preconditioners in Factored Form*, preprint, Russian Academy of Sciences, Moscow, 1999.
[29] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditioning* I. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.
[30] X. S. Li and J. W. Demmel, *Making sparse Gaussian elimination scalable by static pivoting*, in Proceedings of the SuperComputing'98 Conference, CD-ROM, ACM, New York, 1998.
[31] R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.
[32] J. W. H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Trans. Math. Software, 11 (1985), pp. 141–153.
[33] W. D. McQuain, C. J. Ribbens, L. T. Watson, and R. C. Melville, *Preconditioned iterative methods for sparse linear algebra problems arising in circuit simulation*, Comput. Math. Appl., 27 (1994), pp. 25–45.
[34] 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. Comp., 31 (1977), pp. 148–162.
[35] *Matrix Market*, National Institute of Standards and Technology, Gaithersburg, MD, available online from http://math.nist.gov/MatrixMarket.
[36] M. Olschowka and A. Neumaier, *A new pivoting strategy for Gaussian elimination*, Linear Algebra Appl., 240 (1996), pp. 131–151.
[37] G. Pauletto, *Solution and Simulation of Macroeconometric Models*, Ph.D. thesis, Department of Econometrics, University of Geneva, Switzerland, 1995.
[38] Y. Saad, *Preconditioning techniques for indefinite and nonsymmetric linear systems*, J. Comput. Appl. Math., 24 (1988), pp. 89–105.
[39] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996.
[40] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.
[41] H. A. van der Vorst, *BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.