ILUs And Factorized Approximate Inverses are strongly related. Part II: Applications to stabilization

*Matthias Bollhöfer and †Yousef Saad

May 1, 2000

Abstract

In an earlier paper we presented a few results which established some strong relations between incomplete factorization methods and factored approximate inverse-type methods. In this paper some of these relations are exploited to develop new techniques for stabilizing factorized approximate inverse preconditioners using pivoting. This method yields stable preconditioners in many cases and can provide successful preconditioners in many situations when the underlying system is highly indefinite. Numerical examples illustrate the effectiveness of this approach.

Keywords: sparse matrices, ILU, sparse approximate inverse, $AINV$, pivoting.
AMS subject classification: 65F05, 65F10, 65F50.

1 Introduction

Many applications lead to solving large sparse linear systems of the form

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

with $A \in \mathbb{R}^{n,n}$ and $b \in \mathbb{R}^n$. In many cases, such systems are not only very large but also exceedingly difficult to solve by iterative techniques because A is ill-conditioned or highly indefinite or both. In some instances these equations arise from special applications and solvers tailored to the underlying physical problem may give the best results.

*Fakultät für Mathematik, Technische Universität Chemnitz, D–09107 Chemnitz, Germany. Supported by grants of the DFG BO 1680/1-1 and by the University of Minnesota. This research was performed while visiting the University of Minnesota at Minneapolis. email: bolle@mathematik.tu-chemnitz.de, URL: http://www.tu-chemnitz.de/~bolle/.

†Dep. of Computer Science and Engineering, University of Minnesota, 4–192 EE/CSci Building, 200 Union St., SE, Minneapolis, MN 55455–0154. Work supported by NSF and by the Minnesota Supercomputing Institute. email: saad@cs.umn.edu, URL: http://www.cs.umn.edu/~saad/

However, there are situations in which 'general purpose' solvers are required. Such is the case when building general purpose software, or when the linear system has very little inherent structure. In addition, general purpose solvers have many advantages the most significant of which being that changes in the physics or model do not require the development of new methods. In such situations, preconditioned Krylov–subspace solvers, see, e.g., [14, 23, 11] are often seen as the most promising replacements to 'black-box' direct solution methods. Among many techniques, preconditioners based on incomplete LU–factorizations, see e.g. [18, 19, 20] are known to give excellent results for many important classes of problems, such as those arising from the discretization of elliptic partial differential equations. In recent years, a number of techniques have been developed which directly approximate the inverse of A. The popularity of these methods is due mostly to their suitability for parallel computing environments. A few of these approaches are based on minimizing the norm $\|I - AM\|$ in some appropriate norm [17, 15, 13, 7] while others directly solve the equation $Z^\top AW = D$, where the unknown matrices Z, W are unit upper triangular and D is a diagonal matrix, see e.g. [22, 4, 5, 1, 16]. The latter class of methods in particular has a similar algebraic behavior which is already well–known for incomplete LU–decompositions, e.g. they are stable for M– and H–matrices. In [6] relations between factorized approximate inverses and incomplete LU–decompositions have been analyzed. Without describing the details of these relations we simplify both methods to describe these links.

For the solution of (1) incomplete LU factorization which approximately construct a decomposition

$$A \approx LDU$$

where $L, U^\top$ are lower triangular matrices with unit diagonal and D is diagonal. One way to construct these decompositions is to partition A as

$$A = \begin{bmatrix} B & F \\ E & C \end{bmatrix} \in \mathbb{R}^{n,n}$$

with $B \in \mathbb{R}$ and the other blocks have corresponding size. Then A is factored as

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ L_E & I \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} D_B & 0 \\ 0 & S \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} 1 & U_F \\ 0 & I \end{bmatrix}}_{U}, \tag{2}$$

where

$$S = C - L_E D_B U_F \in \mathbb{R}^{n-k,n-k} \tag{3}$$

denotes the so–called Schur–complement. The exact LU–decomposition of A (if it exists) can be obtained by successively applying (2) to the Schur–complement S. Even if there exists a decomposition (2) for A and for S, there is no need to compute L_E, U_F, S exactly when constructing a preconditioner. A common approach for reducing fill-ins consists of

discarding entries L_E, U_F of small size and defining the approximate Schur–complement only with these sparsified vectors $\tilde{L}_E, \tilde{U}_F$. Here we will concentrate on

$$\tilde{S} = B - \tilde{L}_E F - \left(E - \tilde{L}_E B\right) \tilde{U}_F \tag{4}$$

as one possible definition of an approximate Schur–complement. Equation (4) can be obtained from the $(2,2)$ block of $\tilde{L}^{-1} A \tilde{U}^{-1}$. The associated ILU algorithms would be roughly as follows.

Algorithm 1 (Incomplete LU factorization (ILU))
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$, $\tau \in (0,1)$ drop tolerance. Compute $A \approx LDU$.

$L = U = I, S = A, D_{11} = S_{11}$.

for $i = 1 : n-1$

$p_{i+1:n} = S_{i+1:n,i}^{\top} / S_{ii}$, $\quad q_{i+1:n} = S_{i,i+1:n} / S_{ii}$

drop all entries $|p_i|, |q_i|$ if they are less than τ.

$L_{i+1:n,i} = p_{i+1:n}^{\top}$, $\quad U_{i,i+1:n} = q_{i+1:n}$.

$S_{i+1:n,i+1:n} = S_{i+1:n,i+1:n} - L_{i+1:n,i} D_{i,i+1:n} - \left(S_{i+1:n,i} - L_{i+1:n,i} S_{ii}\right) U_{i,i+1:n}$

$D_{i+1,i+1} = S_{i+1,i+1}$

end

Practical versions of incomplete LU decompositions are typically implemented in a slightly different way. It is usually not advisable to update the whole $S_{i+1:n,i+1:n}$ by a rank–1 or rank–2 modification. Instead, the leading row of $S_{i+1:n,i+1:n}$ is typically computed, and the transformations on the other rows are post–poned. In essence this means that the so–called I,K,J version of Gaussian elimination is used. For details, see [22]. In addition to saving memory, this has the advantage that all updates and modifications are performed only once for each row, thus making it possible to use very simple sparse row storage schemes such as the Compressed Sparse Row (CSR) format.

Algorithms for directly computing W, Z such that $W^{\top} A Z = D$, with a diagonal matrix D have recently been suggested in [4, 5, 1, 16]. Here we choose to outline a version that has been used for the symmetric positive definite case.

Algorithm 2 (Factorized Approximate Inverse)
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$, drop tolerance τ. Compute $A^{-1} \approx Z D^{-1} W^{\top}$.

$p = q = (0, \ldots, 0) \in \mathbb{R}^n, Z = W = I_n$.

for $i = 1 : n$

$p_{i:n} = Z_{:,i}^{\top} A^{\top} W_{:,i:n}$ $\quad q_{i:n} = W_{:,i}^{\top} A Z_{:,i:n}$

$p_{i+1:n} = p_{i+1:n} / p_i$; $\quad q_{i+1:n} = q_{i+1:n} / q_i$

$W_{:,i+1:n} = W_{:,i+1:n} - W_{:,i} p_{i+1:n}$ $\quad Z_{:,i+1:n} = Z_{:,i+1:n} - Z_{:,i} q_{i+1:n}$,

drop all entries W_{kl}, Z_{kl}, if they are less than τ in absolute value.

end
Choose diagonal entries of D as the components of q.

In [6], a number of strong connections were established between both classes of algorithms. If we were to apply dropping in Algorithm 2 only to $p_{i+1:n}, q_{i+1:n}$ (after the division by p_i, q_i) instead of W, Z, then one can show that both Algorithms compute precisely the same $p_{i+1:n}, q_{i+1:n}$ and in this case we have $L^{-1} = W^\top, U^{-1} = Z$. Moreover we have that on entry to step i in both algorithms

$$S_{i:n,i:n} = W_{:,i:n}^\top A Z_{:,i:n}. \tag{5}$$

For details see [6]. The main interest of this result is the possibility of exploiting these connections to carry over and adapt some of the techniques used in Gaussian elimination, with a goal of improving the performance of factorized approximate inverses.

2 Approximate Inverses with Pivoting

One way to exploit the direct connection between approximate inverses and incomplete LU factorization is to introduce pivoting to approximate inverses. This can be done by first adding pivoting to Algorithm 1 and then using its relation to Algorithm 2 to introduce pivoting in Algorithm 2. The main reason for introducing pivoting to Algorithm 1 and Algorithm 2 is the fact that the algorithms might encounter a zero, or small, pivot during the computation. At some step i of either algorithm it may turn out that $p_i = q_i = 0$, in which case the algorithms break down. It is possible to shift the zero pivots away by adding an artificial small perturbation (e.g., 10^{-8}) to p_i but this will rarely solve the problem. Instead, we could ensure that zero pivots do not occur and this is traditionally achieved by pivoting in direct Gaussian elimination. This technique was implemented in incomplete factorizations as well [20]. Although this makes the underlying data structure more complex, it often stabilizes the processes and even ensures that the growth in the element size of L, U will remain fairly moderate.

We start with column/row pivoting in Algorithm 1. For generality we will not prescribe a specific rule on how to choose the pivot but will instead introduce column and row interchanges that keep the algorithm consistent when $\tau = 0$ is used. In other words, the algorithm without dropping will compute $\Pi^\top A \Sigma = LDU$, where Π, Σ are permutation matrices that will be determined throughout the process. If π, σ are permutations associated with permutation matrices Π, Σ, then we will write $A(\pi, \sigma)$ for the permuted matrix $\Pi^\top A \Sigma$.

Algorithm 3 (Incomplete LU factorization with pivoting)
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$, $\tau \in (0,1)$ drop tolerance. Compute $A(\pi, \sigma) \approx LDU$.

$L = U = I, S = A = I, D_{11} = S_{11}, \pi = \sigma = (1, \ldots, n)$.

for $i = 1 : n-1$
  while *pivots not satisfactory*
    Find a column pivot $j \geqslant i$.
    Interchange columns i, j *of* $S_{i:n,:}$, *columns* i, j *of* $U_{1:i-1,:}$,
      components i, j *of* σ.
    Find a row pivot $k \geqslant i$.
    Interchange row i, k *of* $S_{:,i:n}$, *row* i, k *of* $L_{:,1:i-1}$,
      components i, k *of* π.
  end
  $p_{i+1:n} = S^{\top}_{i+1:n,i}/S_{ii}, \qquad q_{i+1:n} = S_{i,i+1:n}/S_{ii}$
  Drop all entries $|p_i|, |q_i|$ *if they are less than* τ.
  $L_{i+1:n,i} = p^{\top}_{i+1:n}, \qquad U_{i,i+1:n} = q_{i+1:n}.$
  $S_{i+1:n,i+1:n} = S_{i+1:n,i+1:n} - L_{i+1:n,i}S_{i,i+1:n} - (S_{i+1:n,i} - L_{i+1:n,i}S_{ii})\,U_{i,i+1:n}$
  $D_{i+1,i+1} = S_{i+1,i+1}$
end

If no dropping is applied we obtain $A(\pi, \sigma) = LDU$ by construction. Note that the while–loop is optional and has been included for the purpose of greater generality. It may indeed be the case that a pivot is satisfactory for the rows in the sense that it satisfies a certain desired property, but not for the columns. The property to be satisfied by, say, a column pivot $k \geq i$ at step i, could be a criterion such as

$$|s_{i,k}| \geq \alpha\ |s_{i,j}| \quad \text{for} \quad j \geq i,$$

for a prescribed constant $0 < \alpha \leqslant 1$. After the column interchange takes place, the desired row criterion

$$|s_{i,i}| \geq \alpha\ |s_{j,i}| \quad \text{for} \quad j \geq i$$

may no longer be satisfied, in which case a row interchange will be needed which causes the column criterion no longer to be satisfied. This process usually takes a few steps to complete, in many cases requiring just one step. It allows a better selection by iterating on the choice of the pivots, if necessary, without entailing substantial additional cost in most cases. This additional row pivoting step is not so common in practice. Possibly this is due to the additional overhead for computing not only the leading row of the Schur–complement, but also its leading column. In particular, we note that this version of pivoting is hard to implement with the common IKJ variant of Gaussian Elimination since columns of the Schur complement are not available and expensive to obtain with the corresponding data structure.

After we have introduced pivoting in Algorithm 3 it is now easy to transfer the idea of pivoting to Algorithm 2. To do this we only have to keep in mind that $W = L^{-\top}$ if a modified dropping strategy is applied. Clearly the columns of W, Z have to be permuted analogously to the rows of L and columns of U. The Schur–complement (5) now obviously reads

$$S_{i:n,i:n} = W^{\top}_{:,i:n} A(\pi, \sigma) Z_{:,i:n}. \tag{6}$$

This restricts the application of π, σ to the initial matrix A, if we reorder columns of Z, W.

Algorithm 4 (Factorized Approximate Inverse With Pivoting(AINVP))
Let $A = (A_{ij})_{ij} \in \mathbb{R}^{n,n}$, *drop tolerance* τ. *Compute* $A^{-1} \approx ZD^{-1}W^\top$.
$p = q = (0, \ldots, 0) \in \mathbb{R}^n, Z = W = I_n, \pi = \sigma = (1, \ldots, n)$.

for $i = 1 : n$
  while *pivots not satisfactory*
    $p_{i:n} = Z_{:,i}^\top A(\pi, \sigma)^\top W_{:,i:n}$
    Find a column pivot $j \geqslant i$.
    Interchange components i, j *of* p, *column* i, j *of* $Z_{1:i-1,:}$,
      components i, j *of* σ.
    $q_{i:n} = W_{:,i}^\top A(\pi, \sigma) Z_{:,i:n}$
    Find a row pivot $k \geqslant i$.
    Interchange components i, j *of* q, *column* i, k *of* $W_{1:i-1,:}$,
      components i, k *of* π.
  end
  $p_{i+1:n} = p_{i+1:n}/p_i \qquad q_{i+1:n} = q_{i+1:n}/q_i$
  $W_{:,i+1:n} = W_{:,i+1:n} - W_{:,i}p_{i+1:n}, \quad Z_{:,i+1:n} = Z_{:,i+1:n} - Z_{:,i}q_{i+1:n},$
  drop all entries W_{kl}, Z_{kl}, *if they are less than* τ *in absolute value.*
end
Choose diagonal entries of D *as the components of* q.

The same comments as those made following Algorithm 3 regarding the while–loop also apply to this algorithm. Similarly as well, if no dropping is applied, we obtain $W^\top A(\pi, \sigma)Z = D$, by construction. Pivoting for a related direct projection method can already be found in [3]. Instead of using $W^\top AZ$ to compute p, q only $W^\top A$ is used, which is equivalent in this case because no dropping is applied. In a sense Algorithm 4 generalizes the pivoting approach of [3] in that it is applied to an incomplete factorization and both row and column interchanges are performed.

As they are described, neither Algorithm 3 nor Algorithm 4 specify any rule on how to select the pivots j and k. A reasonable strategy could be to choose j such that $|p_j|$ is maximal and this obviously requires p to be computed before pivoting is applied. A major difference with Algorithm 3 is that now the p and p columns are inside the while loop which searches for adequate pivots. Indeed, p and q must be recomputed whenever q (resp. p) requires an interchange.

In the situation when one pivoting step is applied for any i, then there is no need to recompute p and q. But when more than one pivoting step is required, one of p or q at least must be recomputed. In this case the algorithm incurs some additional overhead. As a result, for any pivoting strategy we should try to keep the additional overhead

associated with recomputing p, q small. For a better stability in Algorithm 4 the pivot p_j should satisfy:

$$|p_j| \geqslant \alpha \max_l |p_l|$$

for some constant $0 < \alpha \leqslant 1$, e.g. $\alpha = 0.1$. But since Algorithm 4 is a biorthogonalization technique, i.e. the outcome is to compute Z **and** W, we should ensure that $|p_j| = |q_j| \geqslant \alpha \max_l |q_l|$ is also fulfilled to guarantee, that the entries of Z and W are sufficiently bounded. This gives us a simple and relatively inexpensive strategy for controlling the growth of the entries in Z, W and to stabilize Algorithm 2.

Algorithm 5 (Controlled Pivoting)
Prescribe a tolerance $\alpha \in (0, 1]$, e.g. $\alpha = 0.1$

satisfied_p=false, satisfied_q=false
while *not satisfied_p*
  $p_{i:n} = Z_{:,i}^{\top} A(\pi, \sigma)^{\top} W_{:,i:n}$
  if $|p_i| < \alpha \max_l |p_l|$
    satisfied_q=false, choose j such that $|p_j| = \max_l |p_l|$
    Interchange column i and j of $Z_{1:i-1,:}$ and components i and j of σ.
  end
  satisfied_p=true
  if *not satisfied_q*
    $q_{i:n} = W_{:,i}^{\top} A(\pi, \sigma) Z_{:,i:n}$
    end
  if $|q_i| < \alpha \max_l |q_l|$
    satisfied_p=false, choose k such that $|q_k| = \max_l |q_l|$
    Interchange column i and k of $W_{1:i-1,:}$ and components i and k of π.
  end
  satisfied_q=true
end

An additional improvement to prevent too many pivoting steps might be to pre-scale the rows and/or columns of A. In principle, it is seldom the case that at most one pivoting step, say column pivoting, is performed in any given step of Algorithm 4. But in order for $|p_i| \geqslant \alpha \max_l |p_l|$ to imply that $|p_i| = |q_i| \geqslant \alpha \max_l |q_l|$ it is necessary that entries of q have magnitudes that are comparable with those of p. By (6), p, q are the first column/row of

$$W_{:,i:n}^{\top} A(\pi, \sigma) Z_{:,i:n}$$

if no pivoting is applied. If W, Z are moderately bounded, which is more or less achieved by pivoting, then $A(\pi, \sigma)$ having rows of comparable absolute row sums might be a good start to prevent too many pivoting steps.

Of course one might think about different pivoting strategies especially with respect to parallel computations. There it might be sensible to restrict j, k to a certain subset to maintain distributed storage schemes. Other strategies could be for example restrictions on j, k to keep $W_{:,i+1:n}^{\top} A(\pi, \sigma) Z_{:,i+1:n}$ as sparse as possible.

3 Numerical Results

This section presents numerical experiments to validate the algorithms. In addition, additional details and comments on the implementation will be given.

- The input matrix is assumed to be given in the CSR format [22].
- The matrices are initially scaled such that they have unit 1–norm for any row. As commented in Section 2 this is done to reduce the number of column/row interchanges necessary.
- $W^{\top}$ and $Z^{\top}$ are stored in CSR format.
- Interchanges of columns of W, Z are performed by only interchanging the references (pointers) instead of the whole data array.
- The computation of p, q in Algorithm 4 requires a multiplication $A(\pi, \sigma) Z_{:,i}$, $A(\pi, \sigma)^{\top} W_{:,i}$. To be efficient, this operation must be done in sparse–sparse mode. If A is given in CSR format only $A(\pi, \sigma)^{\top} W_{:,i}$ is easy to access, while $A(\pi, \sigma) Z_{:,i}$ requires $A^{\top}$ to be stored in CSR format. For this purpose we initially compute the pattern of $A^{\top}$ in CSR format but we omit the numerical values. The computed vectors $A(\pi, \sigma) Z_{:,i}$, $A(\pi, \sigma)^{\top} W_{:,i}$ are stored as a full vector with an additional index lists of the nonzeros. Finally for $W_{:,i:n}^{\top}(A(\pi, \sigma) Z_{:,i})$ and $Z_{:,i:n}^{\top}(A(\pi, \sigma)^{\top} W_{:,i})$ we use a list which contains the non–trivial columns of W, Z, i.e., those columns which contain more than the diagonal entry only. The use of permutation vectors π, σ requires to have the inverse permutations π^{-1}, σ^{-1} which are computed simultaneously.
- Two values were used for the parameter α which controls the pivoting process: $\alpha = 0.1$ and $\alpha = 1.0$.
- Two different values were used for the drop tolerance $\tau = 0.1$, and $\tau = 0.01$.

For the numerical experiments several collections were chosen from the Harwell–Boeing Collection [9], the SPARSKIT Collection [21], and finally from the Davis collection [8]. Throughout the computations the matrices were initially reordered using the symmetric minimum degree ordering [12]. The computations were performed on an SGI workstation with two 190 MHz R10000 (IP25) processors under IRIX 6.2 and 512 MB memory.

The approximate inverse algorithms were implemented in C using dynamic memory allocation.

As iterative solvers we used GMRES(30) and QMR. The iteration was stopped after the residual norm was less than $\sqrt{\text{eps}}$ times the initial residual norm, where eps $\approx 2.2204 \cdot 10^{-16}$ denotes the machine precision. For some matrices a smaller tolerance was necessary, since the exact solution $(1, \ldots, 1)^\top$ was not sufficiently well approximated. In this case eps was used. The iteration was stopped after 500 steps. Every iterative solution which broke down or did not converge within this number of steps was noted as a failure. The approximate inverse algorithms were compared with the SPARSKIT algorithms ILUT and ILUTP [22] using the same settings.

We briefly describe the results for several matrices and then give detailed numerical results for several selected examples. We focus on examples where we observed major differences for AINV with and without pivoting.

To give a rough idea on how the method performed on the Harwell–Boeing collection we simply summarize in Table 1, which method successfully solved how many problems with respect to the parameters τ, α. The tests were done on 94 matrices from the Harwell-Boeing collection 26 matrices from the Davis–collection and 58 matrices from the SPARSKIT–collection.

From Table 1 one gets the impression, that Algorithm 4 (AINVP) behaves slightly better than ILUTP. This might have the following reasons.

1. AINVP uses column **and** row pivoting to ensure that W **and** Z are well–bounded. Pivoting only applied to the columns, for example, would locally only bound one factor. But this is essentially what ILUTP does. For reasons of efficiency pivoting with respect to the rows is not done.
2. dropping in AINVP is less harmful than in ILUTP. Approximation errors caused by dropping in AINVP behave somehow between linear and quadratic with respect to the values that are dropped. For ILUTP the analogous effect is rational which means that small perturbation in L, U may cause huge approximation errors in $L^{-1}AU^{-1}$.
3. AINVP sometimes ends up with more fill–in. In the numerical examples dropping was only performed with respect to a fixed drop tolerance but not with respect to the number of nonzeros. The results show that sometimes AINVP needs significantly more fill–in than ILUTP (e.g. Table 8).

We now comment on some matrices from the Harwell–Boeing–Collection for those cases where we observed major differences between AINV with and without pivoting.

- CHEMIMP: without pivoting, AINV, ILUT do not converge neither for $\tau = 0.1$ nor for $\tau = 0.01$. With pivoting AINVP converges for all choices of τ, α. ILUTP

Table 1: **Summary of results - total of 178 matrices**

Preconditioner	Accelerator	Parameters			
		$\tau = 0.1$		$\tau = 0.01$	
		$\alpha = 0.1$	$\alpha = 1.0$	$\alpha = 0.1$	$\alpha = 1.0$
Harwell–Boeing Collection (94 test matrices)					
AINV	GMRES(30)	35		39	
AINV	QMR	38		38	
AINVP	GMRES(30)	57	63	78	86
AINVP	QMR	66	72	85	84
ILUT	GMRES(30)	44		43	
ILUT	QMR	41		44	
ILUTP	GMRES(30)	53	54	69	71
ILUTP	QMR	59	58	74	76
Davis Collection (26 matrices)					
AINV	GMRES(30)	14		14	
AINV	QMR	14		14	
AINVP	GMRES(30)	12	16	17	19
AINVP	QMR	15	17	18	20
ILUT	GMRES(30)	14		14	
ILUT	QMR	14		14	
ILUTP	GMRES(30)	13	15	16	18
ILUTP	QMR	14	15	16	17
SPARSKIT Collection (58 matrices)					
AINV	GMRES(30)	2		6	
AINV	QMR	4		9	
AINVP	GMRES(30)	3	16	14	32
AINVP	QMR	4	27	17	34
ILUT	GMRES(30)	7		18	
ILUT	QMR	9		20	
ILUTP	GMRES(30)	6	9	19	19
ILUTP	QMR	11	12	25	23

also converged for almost all matrices and choice of τ, α, except for IMPCOLA, GMRES(30) and $\tau = 0.1, \alpha = 0.1$. The fill–in for $\tau = 0.1, \alpha = 0.1$ is at most four times the initial number of nonzeros. For ILUTP the fill–in was even less.

- CHEMWEST: without pivoting neither AINV nor ILUT were able to solve these problems. With pivoting $\alpha = 0.1$, AINV and ILUTP solved several small problems but failed for most of the bigger matrices. With $\alpha = 1$ the results are much better for both algorithms and, most notably, $AINV$ solves almost all problems. Detailed results for the four biggest WEST–matrices are given in Table 2, 3, 4, 5. In these examples $GMRES(30), QMR$ did not converge with $ILUTP$ for any selection of τ, α.

- ECONAUS: Without pivoting neither AINV nor ILUT preconditioned iterations converged. With pivoting, both AINVP and ILUTP converged in a few number of steps. This is already true for all choices of τ and α. In this example the error between the true solution and the computed solution was sometimes much bigger than the residual. For this reason the iteration was stopped after the computed residual was less than eps times the initial residual.

- FACSIMILE: AINV without pivoting sometimes performs much better than AINV with pivoting, especially for $\tau = 0.1, \alpha = 0.1$. This effect is much weaker for all other choices of τ, α. For some of these matrices the error in the approximate solution was still big after the residual has been reduced by $\sqrt{\text{eps}}$ (machine precision). For that reason eps was used as tolerance. For some selected examples see Table 6 and 7. FS7602, FS7603 could not be solved by any of the methods and any settings.

- PORES: PORES1, PORES3 could be solved by any method and $\tau = 0.1, \alpha = 0.1$. For PORES3 and GMRES(30), AINV with pivoting had significantly fewer iterations than all other methods while the fill–in was less than 3 times the number of the initial matrix. ILUT, ILUTP had even less fill–in than the original matrix. For matrix PORES2 see Table 8.

- SAYLOR: SAYLR1/SAYLR3 were solved by all methods already for $\tau = 0.1, \alpha = 0.1$ and a moderate number steps as well as a moderate amount of fill–in. For SAYLR4 and $\tau = 0.1$ none of the algorithms performed well using GMRES. The situation was much better for $\tau = 0.01$. With pivoting the number of iteration steps was much less for AINV than without pivoting but this advantage was compensated by much less fill–in without pivoting.

- SHERMAN: At the exception of SHERMAN2 and SHERMAN3, all other matrices could be solved with any method with $\tau = 0.1, \alpha = 0.1$ in a moderate number of steps and a moderate amount of fill–in. All methods performed poorly for SHERMAN3 with $\tau = 0.1$ and GMRES(30) and QMR required approximately 100 steps in most cases. These problems did not occur for $\tau = 0.01$. SHERMAN2 could not

be solved without pivoting. Even with pivoting only AINVP was able to solve this problem, see Table 9.

- SMTAPE: For none of the BP–matrices AINV without pivoting or ILUT worked. AINVP with pivoting as well as ILUTP worked for almost all BP–matrices with $\tau = 0.01$. For some of the BP–matrices the fill–in of AINV with pivoting is pretty big, e.g. BP800 leads to a fill–in factor 13.8 and 13.6 for $\tau = 0.1, 0.01$ and $\alpha = 0.1$. For $\alpha = 1$ the fill–in was only 2.3 and 7.0. The situation was even better for the SHL–matrices and the STR–matrices. The last two sets of matrices could already be solved for $\tau = 0.1, \alpha = 0.1$ in a small number of steps (often less than 20) with a moderate fill–in (less than a factor 3).

The fact that many unsymmetric matrices from the Harwell–Boeing collection are not mentioned does not mean that AINV/AINVP did work for them. It only means that no great improvement was achieved with pivoting and also that there was no severe damage on the convergence properties by using pivoting. For example, none of the methods performed very well on the big LNS–matrices or on the NUCL–matrices.

Table 2: **Matrix CHEMWEST/WEST0655**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.10$, $\alpha = 0.1$	AINVP	9.4, $7.4 \cdot 10^{-1}$	—, —	—, —
$\tau = 0.01$, $\alpha = 0.1$	AINVP	11.4, $1.0 \cdot 10^{0}$	—, —	68, $7.2 \cdot 10^{-1}$
$\tau = 0.10$, $\alpha = 1.0$	AINVP	2.2, $2.7 \cdot 10^{-1}$	172, $4.7 \cdot 10^{-1}$	83, $3.6 \cdot 10^{-1}$
$\tau = 0.01$, $\alpha = 1.0$	AINVP	4.4, $4.6 \cdot 10^{-1}$	17, $6.0 \cdot 10^{-2}$	19, $1.3 \cdot 10^{-1}$

Table 3: **Matrix CHEMWEST/WEST0989**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.10$, $\alpha = 0.1$	AINVP	2.2, $3.1 \cdot 10^{-1}$	—, —	158, $1.3 \cdot 10^{0}$
$\tau = 0.01$, $\alpha = 0.1$	AINVP	3.7, $4.4 \cdot 10^{-1}$	50, $2.8 \cdot 10^{-1}$	36, $3.5 \cdot 10^{-1}$
$\tau = 0.10$, $\alpha = 1.0$	AINVP	1.4, $3.0 \cdot 10^{-1}$	69, $3.0 \cdot 10^{-1}$	69, $5.1 \cdot 10^{-1}$
$\tau = 0.01$, $\alpha = 1.0$	AINVP	2.4, $3.7 \cdot 10^{-1}$	25, $1.2 \cdot 10^{-1}$	29, $2.4 \cdot 10^{-1}$

Table 4: **Matrix CHEMWEST/WEST1505**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.10$, $\alpha = 0.1$	AINVP	2.5, $5.4{\cdot}10^{-1}$	—, —	—, —
$\tau = 0.01$, $\alpha = 0.1$	AINVP	6.4, $1.0{\cdot}10^{-1}$	—, —	333, $7.4{\cdot}10^{0}$
$\tau = 0.10$, $\alpha = 1.0$	AINVP	1.6, $3.0{\cdot}10^{-1}$	—, —	149, $2.1{\cdot}10^{0}$
$\tau = 0.01$, $\alpha = 1.0$	AINVP	2.6, $6.8{\cdot}10^{-1}$	45, $4.1{\cdot}10^{-1}$	52, $8.3{\cdot}10^{-1}$

Table 5: **Matrix CHEMWEST/WEST2021**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.10$, $\alpha = 0.1$	AINVP	2.9, $8.7{\cdot}10^{-1}$	—, —	—, —
$\tau = 0.01$, $\alpha = 0.1$	AINVP	8.4, $2.2{\cdot}10^{0}$	—, —	—, —
$\tau = 0.10$, $\alpha = 1.0$	AINVP	1.5, $6.8{\cdot}10^{-1}$	121, $1.5{\cdot}10^{0}$	81, $1.8{\cdot}10^{0}$
$\tau = 0.01$, $\alpha = 1.0$	AINVP	2.8, $1.1{\cdot}10^{0}$	31, $4.5{\cdot}10^{-1}$	37, $9.3{\cdot}10^{-1}$

Table 6: **Matrix FACSIMILE/FS1833**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.10$ $\alpha = 0.1$	AINV	0.8, $4.0{\cdot}10^{-2}$	19, $< 1.0{\cdot}10^{-2}$	26, $2.0{\cdot}10^{-2}$
	AINVP	0.8, $5.0{\cdot}10^{-2}$	211, $7.0{\cdot}10^{-2}$	98, $7.0{\cdot}10^{-2}$
	ILUT	0.5, $< 1.0{\cdot}10^{-2}$	16, $< 1.0{\cdot}10^{-2}$	16, $1.0{\cdot}10^{-2}$
	ILUTP	0.6, $< 1.0{\cdot}10^{-2}$	17, $2.0{\cdot}10^{-2}$	18, $< 1.0{\cdot}10^{-2}$
$\tau = 0.01$ $\alpha = 0.1$	AINV	1.0, $4.0{\cdot}10^{-2}$	17, $1.0{\cdot}10^{-2}$	21, $1.0{\cdot}10^{-2}$
	AINVP	1.1, $6.0{\cdot}10^{-2}$	59, $2.0{\cdot}10^{-2}$	71, $5.0{\cdot}10^{-2}$
	ILUT	0.6, $1.0{\cdot}10^{-2}$	14, $1.0{\cdot}10^{-2}$	14, $< 1.0{\cdot}10^{-2}$
	ILUTP	0.7, $< 1.0{\cdot}10^{-2}$	14, $1.0{\cdot}10^{-2}$	16, $< 1.0{\cdot}10^{-2}$
$\tau = 0.1$ $\alpha = 1.0$	AINVP	0.7, $6.0{\cdot}10^{-2}$	93, $3.0{\cdot}10^{-2}$	81, $5.0{\cdot}10^{-2}$
	ILUTP	0.6, $< 1.0{\cdot}10^{-2}$	17, $< 1.0{\cdot}10^{-2}$	18, $1.0{\cdot}10^{-2}$
$\tau = 0.01$ $\alpha = 1.0$	AINVP	1.1, $6.0{\cdot}10^{-2}$	56, $2.0{\cdot}10^{-2}$	70, $6.0{\cdot}10^{-2}$
	ILUTP	0.7, $< 1.0{\cdot}10^{-2}$	14, $1.0{\cdot}10^{-2}$	18, $< 1.0{\cdot}10^{-2}$

Table 7: **Matrix FACSIMILE/FS5413**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.1$ $\alpha = 0.1$	AINV	0.9, $1.6{\cdot}10^{-1}$	—, —	220, $6.6{\cdot}10^{-1}$
	AINVP	1.0, $1.7{\cdot}10^{-1}$	—, —	—, —
	ILUT	0.5, $< 1.0{\cdot}10^{-2}$	299, $2.2{\cdot}10^{-1}$	172, $2.3{\cdot}10^{-1}$
	ILUTP	0.6, $< 1.0{\cdot}10^{-2}$	—, —	—, —
$\tau = 0.01$ $\alpha = 0.1$	AINV	1.6, $1.9{\cdot}10^{-1}$	335, $6.6{\cdot}10^{-1}$	124, $4.5{\cdot}10^{-1}$
	AINVP	1.8, $2.3{\cdot}10^{-1}$	40, $8.0{\cdot}10^{-2}$	39, $1.6{\cdot}10^{-1}$
	ILUT	0.7, $< 1.0{\cdot}10^{-2}$	230, $1.7{\cdot}10^{-1}$	107, $1.5{\cdot}10^{-1}$
	ILUTP	0.8, $< 1.0{\cdot}10^{-2}$	—, —	—, —
$\tau = 0.1$ $\alpha = 1.0$	AINVP	0.8, $1.8{\cdot}10^{-1}$	—, —	—, —
	ILUTP	0.6, $1.0{\cdot}10^{-2}$	—, —	—, —
$\tau = 0.01$ $\alpha = 1.0$	AINVP	1.6, $2.2{\cdot}10^{-1}$	41, $7.0{\cdot}10^{-2}$	44, $1.7{\cdot}10^{-1}$
	ILUTP	0.8, $1.0{\cdot}10^{-2}$	—, —	—, —

Table 8: **Matrix PORES/PORES2**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.1$ $\alpha = 0.1$	AINV	1.1, $3.4{\cdot}10^{-1}$	—, —	208, $2.3{\cdot}10^{0}$
	AINVP	2.8, $6.2{\cdot}10^{-1}$	—, —	—, —
	ILUT	0.5, $1.0{\cdot}10^{-2}$	434, $7.4{\cdot}10^{-1}$	150, $4.4{\cdot}10^{-1}$
	ILUTP	0.4, $1.0{\cdot}10^{-2}$	—, —	289, $1.0{\cdot}10^{0}$
$\tau = 0.01$ $\alpha = 0.1$	AINV	2.4, $4.1{\cdot}10^{-1}$	60, $5.4{\cdot}10^{-1}$	52, $7.2{\cdot}10^{-1}$
	AINVP	8.9, $2.2{\cdot}10^{0}$	26, $7.4{\cdot}10^{-1}$	27, $8.5{\cdot}10^{-1}$
	ILUT	0.8, $1.0{\cdot}10^{-2}$	276, $5.5{\cdot}10^{-1}$	89, $2.8{\cdot}10^{-1}$
	ILUTP	0.8, $1.0{\cdot}10^{-2}$	—, —	474, $1.7{\cdot}10^{0}$
$\tau = 0.1$ $\alpha = 1.0$	AINVP	1.4, $6.3{\cdot}10^{-1}$	—, —	282, $1.0{\cdot}10^{0}$
	ILUTP	0.5, $1.0{\cdot}10^{-2}$	—, —	—, —
$\tau = 0.01$ $\alpha = 1.0$	AINVP	6.0, $2.6{\cdot}10^{0}$	29, $4.0{\cdot}10^{-1}$	31, $7.4{\cdot}10^{-1}$
	ILUTP	1.1, $2.0{\cdot}10^{-2}$	—, —	—, —

Table 9: **Matrix SHERMAN/SHERMAN2**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.10$, $\alpha = 0.1$	AINVP	0.6, $6.4 \cdot 10^{-1}$	—, —	—, —
$\tau = 0.01$, $\alpha = 0.1$	AINVP	1.0, $1.2 \cdot 10^{0}$	55, $5.5 \cdot 10^{-1}$	—, —
$\tau = 0.10$, $\alpha = 1.0$	AINVP	0.3, $6.8 \cdot 10^{-1}$	151, $1.2 \cdot 10^{0}$	100, $1.2 \cdot 10^{0}$
$\tau = 0.01$, $\alpha = 1.0$	AINVP	0.6, $1.0 \cdot 10^{0}$	55, $4.3 \cdot 10^{-1}$	38, $5.3 \cdot 10^{-1}$

Next are some comments on matrices from the Davis–Collection. Sample matrices from Hamm, Mallya, Portfolio, Shyy, Simon, Wang, and Zitney were used for the tests.

- Mallya: None of the methods was able to solve any of the matrices lhr01,lhr02,lhr04 or lhr07 no matter which set of parameters was used. There was one exception. AINVP with $\tau = 0.02, \alpha = 1$ and QMR as well as ILUTP with the same parameters and GMRES solved lhr01. This is remarkable because these matrices are strongly off–diagonal dominant but even pivoting is not able handle them.

- simon: For the RAEFSKY–matrices it was necessary to use a different stopping criterion (eps instead of $\sqrt{\text{eps}}$). For $\sqrt{\text{eps}}$ all methods converged in a moderate number of steps, except ILUTP which did not converge for RAEFSKY6 for any set of parameters. For eps and $\tau = 0.1$ some methods did not converge within 500 steps but reduced the residual norm by much more than a factor of $\sqrt{\text{eps}}$ (again except ILUTP, RAEFSKY6). For these matrices AINV,AINVP had much less fill–in than ILUT,ILUTP and the original matrices.

- Zitney: HYDR1 could not be solved by any method and EXTR1 only worked with AINVP, $\tau = 0.01, \alpha = 1$ with QMR. The other matrices could finally be solved with $AINVP$ and $ILUTP$ with several choices of τ, α while $ILUT$ and $AINV$ did not converge for any choice of parameters. As an example for the performance of $AINVP$ and $ILUTP$ see Table 10. The Zitney matrices are strongly off–diagonally dominant and thus it is no surprise that with relaxed pivoting $\alpha = 0.1$ much more fill–in is produced than with strict pivoting.

Finally, we provide some comments on the numerical experiments with sample matrices from the SPARSKIT–Collection.

- DRIVCAV: Most of the small matrices (e05r*–matrices) could be solved with pivoting but $\tau = 0.01$. Without pivoting $ILUT$ worked for the smaller matrices

Table 10: **Matrix ZITNEY/RDIST1**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.1$	AINVP	78.7, $2.0 \cdot 10^{3}$	—, —	—, —
$\alpha = 0.1$	ILUTP	1.5, $4.5 \cdot 10^{-1}$	—, —	—, —
$\tau = 0.01$	AINVP	21.8, $3.9 \cdot 10^{2}$	—, —	236, $1.6 \cdot 10^{2}$
$\alpha = 0.1$	ILUTP	3.2, $1.5 \cdot 10^{0}$	60, $3.9 \cdot 10^{0}$	55, $6.5 \cdot 10^{0}$
$\tau = 0.1$	AINVP	8.8, $1.6 \cdot 10^{2}$	—, —	—, —
$\alpha = 1.0$	ILUTP	1.3, $3.3 \cdot 10^{-1}$	—, —	—, —
$\tau = 0.01$	AINVP	7.7, $1.4 \cdot 10^{2}$	50, $7.1 \cdot 10^{0}$	46, $1.3 \cdot 10^{1}$
$\alpha = 1.0$	ILUTP	2.9, $1.2 \cdot 10^{0}$	23, $1.4 \cdot 10^{0}$	30, $3.5 \cdot 10^{0}$

and smaller Reynolds numbers. AINV without pivoting did not work for any of these matrices. For almost all larger matrices no method was satisfactory. AINVP could solve some of the e20r*–matrices with Reynolds number 2000 or less but only with $\tau = 0.01, \alpha = 1$. $ILUT(P)$ did not solve any of the bigger matrices.

- FIDAP: Most of the systems could not be solved by any of the methods. Some matrices (FIDAP–001, 004, 005, 022, 023, 027, 029, 031, 037, M03, M05, M10) were solved with $\tau = 0.01, \alpha = 1$, i.e. strict pivoting but only a few of these matrices were solved without pivoting or relaxed pivoting. With $\tau = 0.1$ the situation became much worse. Examples are given in Table 11.

- TOKAMAK: In this set of sample matrices the methods without pivoting performed significantly better than those which include pivoting. UTM1700a, UTM1700b, UTM300 could be solved with almost all parameters and AINV, ILUT while AINVP did not converge for any parameters for UTM1700a, UTM1700b. ILUTP performed better but even in this case the situation was better with relaxed pivoting ($\alpha = 0.1$) than with strict pivoting. Only ILUT was able to partially solve the bigger matrices UTM3060, UTM5940 together with QMR.

For several matrices here in most cases the codes for $ILUT/ILUTP$ are much faster than those for the approximate inverses. In fact the implementation of the approximate inverse algorithm with or without pivoting is much more technical and the codes used for the experiments are research codes which have not been profiled yet. A much improved implementation is still possible, see, e.g., the numerical results in [5]. However, to compare the total amount of work for the decomposition, we will show the fill–in and the computation time for some larger matrices. These examples are not necessarily spectacular with respect to the iterative solution since all methods performed very well.

Table 11: **Matrix SPARSKIT/FIDAP31**

Method		Decomposition	$GMRES(30)$	QMR
		Fill–in/time [sec]	Steps/time [sec]	Steps/time[sec]
$\tau = 0.1$ $\alpha = 0.1$	AINV	1.1, $4.2{\cdot}10^{0}$	—, —	—, —
	AINVP	28.5, $1.0{\cdot}10^{3}$	—, —	—, —
	ILUT	0.9, $2.0{\cdot}10^{-1}$	—, —	—, —
	ILUTP	1.8, $1.0{\cdot}10^{0}$	—, —	418, $3.8{\cdot}10^{1}$
$\tau = 0.01$ $\alpha = 0.1$	AINV	3.5, $1.1{\cdot}10^{1}$	—, —	—, —
	AINVP	9.9, $2.4{\cdot}10^{2}$	—, —	167, $6.3{\cdot}10^{1}$
	ILUT	1.2, $2.6{\cdot}10^{-1}$	92, $3.9{\cdot}10^{0}$	76, $5.7{\cdot}10^{0}$
	ILUTP	3.7, $4.4{\cdot}10^{0}$	26, $1.9{\cdot}10^{0}$	26, $3.7{\cdot}10^{0}$
$\tau = 0.1$ $\alpha = 1.0$	AINVP	0.6, $5.2{\cdot}10^{0}$	—, —	195, $1.7{\cdot}10^{1}$
	ILUTP	3.9, $6.4{\cdot}10^{0}$	—, —	—, —
$\tau = 0.01$ $\alpha = 1.0$	AINVP	3.8, $3.6{\cdot}10^{1}$	48, $4.7{\cdot}10^{0}$	37, $7.1{\cdot}10^{0}$
	ILUTP	5.8, $1.5{\cdot}10^{1}$	27, $2.9{\cdot}10^{0}$	29, $6.7{\cdot}10^{0}$

So we restrict ourselves to the factorization times. In addition we compare the computation time on a different machine. The computer we took is a Linux PC with Pentium II–400 processor and 64 MB memory. See Table 12 for detailed results. We can see that clearly the results with $ILUT/ILUTP$ are superior with respect to computation time, but the differences are less dramatic and the overhead for pivoting seems to be quite moderate. A time–consuming part in our approximate inverse implementation was the dynamic memory allocation.

Table 12: **Computation time for matrix WANG/WANG4** ($n = 26068$)

Method		Decomposition		
		Fill–in	time [sec] IRIX	time [sec] Linux
$\tau = 0.1$ $\alpha = 0.1$	AINV	1.1	$8.7{\cdot}10^{0}$	$2.7{\cdot}10^{0}$
	AINVP	1.4	$1.2{\cdot}10^{1}$	$4.8{\cdot}10^{0}$
	ILUT	0.9	$2.1{\cdot}10^{-1}$	$1.6{\cdot}10^{-1}$
	ILUTP	0.9	$2.8{\cdot}10^{-1}$	$2.1{\cdot}10^{-1}$
$\tau = 0.1$ $\alpha = 1.0$	AINVP	1.1	$1.1{\cdot}10^{1}$	$4.0{\cdot}10^{0}$
	ILUTP	0.9	$2.7{\cdot}10^{-1}$	$2.1{\cdot}10^{-1}$

Finally, after illustrating the benefits of using pivoting in the approximate inverse preconditioner with several examples we will examine the combination of pivoting with an a priori permutation and scaling suggested in [2]. At first glance the use of pivoting and especially the use of strict pivoting seems to be a complimentary approach to gain more stability. But clearly combining two different approaches in an appropriate way can be a good compromise. We illustrate this on some matrices which have been reordered and scaled using the method from [2] together with relaxed pivoting ($\alpha = 0.1$). We compare these results with strict pivoting ($\alpha = 1$) and no a priori permutation and with only a priori permutation but no pivoting. See Table 13, 14, 15.

Table 13: **Matrix BP/BP1200**

version of AINV		Fill–in/ time[sec]	$GMRES(30)$ steps/time[sec]	QMR steps/time[sec]
$\tau = 0.1$	only	3.1	—	—
	pivoting	$5.5 \cdot 10^{-1}$	—	—
	only	5.1	—	277
	preprocessing	$3.2 \cdot 10^{-1}$	—	$2.5 \cdot 10^{0}$
	preprocessing	4.8	49	37
	+ pivoting	$4.1 \cdot 10^{-1}$	$2.4 \cdot 10^{-1}$	$3.2 \cdot 10^{-1}$
$\tau = 0.01$	only	8.3	—	55
	pivoting	$1.4 \cdot 10^{0}$	—	$8.1 \cdot 10^{-1}$
	only	7.3	20	40
	preprocessing	$3.9 \cdot 10^{-1}$	$1.2 \cdot 10^{-1}$	$4.6 \cdot 10^{-1}$
	preprocessing	7.5	9	8
	+ pivoting	$5.1 \cdot 10^{-1}$	$7.0 \cdot 10^{-2}$	$1.2 \cdot 10^{-1}$

4 Conclusions

We have presented a version of a factorized approximate inverse with enhanced stability properties. The algorithm is obtained by carrying over pivoting strategies from LU–decomposition techniques to approximate inverse, exploiting a strong connection between ILU-type methods and Factored Approximate inverse type methods. A test with a fairly large collection of test matrices established clearly the advantages of using pivoting. Pivoting in AINV increases robustness in the harder cases and is unlikely to hamper performance too much in the easier cases. Combining Approximate Inverse with pivoting, row scaling, and a technique of nonsymmetric permutation developed elsewhere [2, 10], shows excellent improvements in robustness of AINV, and opens the possibility of developing reliable preconditioners for very poorly structured matrices.

Table 14: **Matrix WEST/WEST0655**

version of AINV		Fill–in/ time[sec]	$GMRES(30)$ steps/time[sec]	QMR steps/time[sec]
$\tau = 0.1$	only pivoting	2.2 $2.7 \cdot 10^{-1}$	172 $4.7 \cdot 10^{-1}$	83 $3.6 \cdot 10^{-1}$
	only preprocessing	10.4 $3.2 \cdot 10^{-1}$	— —	172 $1.4 \cdot 10^{0}$
	preprocessing + pivoting	5.7 $3.6 \cdot 10^{-1}$	16 $5.0 \cdot 10^{-2}$	16 $1.1 \cdot 10^{-1}$
$\tau = 0.01$	only pivoting	4.4 $4.6 \cdot 10^{-1}$	17 $6.0 \cdot 10^{-2}$	19 $1.3 \cdot 10^{-1}$
	only preprocessing	7.3 $3.9 \cdot 10^{-1}$	20 $1.2 \cdot 10^{-1}$	40 $4.6 \cdot 10^{-1}$
	preprocessing + pivoting	7.5 $5.1 \cdot 10^{-1}$	9 $7.0 \cdot 10^{-2}$	8 $1.2 \cdot 10^{-1}$

Table 15: **Matrix WEST/WEST2021**

version of AINV		Fill–in/ time[sec]	$GMRES(30)$ steps/time[sec]	QMR steps/time[sec]
$\tau = 0.1$	only pivoting	1.5 $6.8 \cdot 10^{-1}$	121 $1.5 \cdot 10^{0}$	81 $1.8 \cdot 10^{0}$
	only preprocessing	3.7 $5.0 \cdot 10^{-1}$	— —	— —
	preprocessing + pivoting	3.5 $6.0 \cdot 10^{-1}$	24 $3.0 \cdot 10^{-1}$	25 $5.7 \cdot 10^{-1}$
$\tau = 0.01$	only pivoting	2.8 $1.1 \cdot 10^{0}$	31 $4.5 \cdot 10^{-1}$	37 $9.3 \cdot 10^{-1}$
	only preprocessing	8.1 $6.5 \cdot 10^{-1}$	13 $2.3 \cdot 10^{-1}$	13 $4.2 \cdot 10^{-1}$
	preprocessing + pivoting	7.8 $8.3 \cdot 10^{-1}$	9 $1.4 \cdot 10^{-1}$	8 $2.8 \cdot 10^{-1}$

On the negative side, pivoting is undoubtedly harder to implement in parallel. As is often done however, it is possible to exploit relaxed pivoting to search for satisfactory pivots locally, i.e., in each processor.

Acknowledgment. We wish to thank Michele Benzi for providing us with sample matrices that have been preprocessed using the technique from [2]. This allowed us to obtain the results at the end of Section 3 which combined the methods described in this paper with this technique.

References

[1] M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. Technical report LA–UR–99–2899, Los Alamos National Laboratory, Scientific Computing Group, 1999.

[2] M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. Technical Report LA–UR–99–4857, Los Alamos National Laboratory, Scientific Computing Group (CIC–19), 1999.

[3] M. Benzi and C. D. Meyer. A direct projection method for sparse linear systems. *SIAM J. Sci. Comput.*, 16(5):1159–1176, 1995.

[4] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17:1135–1149, 1996.

[5] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994, 1998.

[6] M. Bollhoefer and Y. Saad. *ILU*s and factorized approximate inverses are strongly related. Part I: Overview of results. Technical Report umsi–2000-39, Minnesota Supercomputer Institue, University of Minnesota, 2000.

[7] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse–sparse iterations. *SIAM J. Sci. Comput.*, 19(3):995–1023, 1998.

[8] T. Davis. Sparse matrix collection. NA Digest, 1994.

[9] I. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM Trans. Math. Software*, 15:1–14, 1989.

[10] 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.*, 20:889–901, 1999.

[11] R. Freund, G. Golub, and N. Nachtigal. Iterative solution of linear systems. *Acta Numerica*, pages 1–44, 1992.

[12] J. A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[13] M. Grote and T. Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18(3), 1997.

[14] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.

[15] I. E. Kaporin. New convergence results and preconditioning strategies for conjugate gradient method. *Numer. Lin. Alg. w. Appl.*, 1(2):179–210, 1994.

[16] S. Kharchenko, L. Kolotilina, A. Nikishin, and A. Yeremin. A reliable AINV–type preconditioning method for constructing sparse approximate inverse preconditioners in factored form. Technical report, Russian Academy of Sciences, Moscow, 1999.

[17] Y. Kolotilina and Y. Yeremin. Factorized sparse approximate inverse preconditionings I. theory. *SIAM J. Matrix Anal. Appl.*, 14:45–58, 1993.

[18] J. Meijerink and H. A. V. der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric m–matrix. *Math. Comp.*, 31:148–162, 1977.

[19] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method. *ACM Trans. Math. Software*, 6:206–219, 1980.

[20] Y. Saad. ILUT: a dual treshold incomplete ILU factorization. *Numer. Lin. Alg. w. Appl.*, 1:387–402, 1994.

[21] Y. Saad. SPARSKIT and sparse examples. NA Digest, 1994.

[22] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.

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