BIT 38 (1998), 234–241.

NUMERICAL EXPERIMENTS WITH TWO APPROXIMATE INVERSE PRECONDITIONERS *

MICHELE BENZI[1 †] and MIROSLAV TŮMA[2 ‡]

[1] *CERFACS, 42 Ave. G. Coriolis, 31057 Toulouse Cedex, France. email: benzi@cerfacs.fr*

[2] *Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod vodárenskou věží 2, 182 07 Praha 8, Czech Republic. email: tuma@uivt.cas.cz*

Abstract.

We present the results of numerical experiments aimed at comparing two recently proposed sparse approximate inverse preconditioners from the point of view of robustness, cost, and effectiveness. Results for a standard ILU preconditioner are also included. The numerical experiments were carried out on a Cray C98 vector processor.

AMS subject classification: 65F10, 65F35, 65F50.

Key words: Sparse linear systems, preconditioned iterative methods, approximate inverses, SPAI, incomplete biconjugation, ILU.

1 Introduction.

In this paper, we consider the solution of large and sparse systems of linear equations

$$Ax = b \tag{1.1}$$

by means of iterative techniques. There are now quite a few Krylov subspace methods for solving general linear systems, e. g., GMRES, Bi-CGSTAB and QMR (see [2]). In order to be effective, these methods must be combined with a good preconditioner, and it is generally agreed that the choice of the preconditioner is even more crucial than the choice of the Krylov subspace iteration. We refer to [16] for a thorough treatment of preconditioned iterative methods. The search for effective *parallel* preconditioners is an active research topic in scientific computing. Several potentially successful methods have been recently proposed in the literature, but there is a lack of direct comparisons between these new techniques. In this paper we discuss the results of a comparison between what we regard as two of the most promising approaches, the SPAI method [12] and a factorized approach based on incomplete biconjugation [3],[4]. These techniques are also compared with a standard ILU preconditioner. All computations were performed on a Cray C98 vector processor.

*Received July 1997. Revised December 1997.

†Present address: Scientific Computing Group (CIC-19), Los Alamos National Laboratory, MS B256, Los Alamos, NM 87545, USA. email: benzi@lanl.gov.

‡This work was partially supported by the GA AS CR under grant 2030706 and by the grant GA CR 205/96/0921.

2 Parallel Preconditioned Iterative Methods.

It is well known that the three most crucial components from the point of view of the parallel implementation of preconditioned iterative methods are the construction of the preconditioner, the matrix-vector multiplication in the iteration loop, and the application of the preconditioner in the iteration loop. We postpone the first issue to the next section, and we briefly discuss here the last two operations, which account for most of the computations in the iterative part of the algorithm. Sparse matrix-vector multiplication may present a challenge because it is usually not supported by standard vendor-supplied software, as there is still no unified sparse and parallel BLAS. In the case of vector multiprocessors, the matrix-vector multiply should also be efficiently vectorized using hardware gather-scatter. To this end, it is usually recommended to store the sparse coefficient matrix in jagged diagonal format (JAD), see [16]. If several processors are available, a block version of JAD should be used [13]. In a distributed memory environment, matrix-vector multiplication can be parallelized by block row partitioning, possibly combined with some preprocessing of the data in order to minimize communication and to improve load balancing (see [16], Section 11.5.6).

The preconditioner application in the iteration loop is the most delicate part of the iterative process. The reason is that many of the best serial preconditioners, like incomplete Cholesky or ILU, are notoriously difficult to implement on high-performance computers, due to the necessity of solving sparse linear systems with triangular matrices at each iteration step. Unfortunately, this is an inherently sequential computation. Although the sparsity of the matrix can help remove some data dependencies (as in *level scheduling*, see [16]), the preconditioner computation and application still represent a sequential bottleneck limiting the effectiveness of preconditioned Krylov subspace methods on vector and parallel computers. For this and other reasons, in the last few years there has been a growing interest in alternative forms of preconditioning, such as sparse approximate inverse techniques, which are based on approximating directly the matrix inverse A^{-1}, see [3],[4],[7],[8],[9],[11],[12],[14]. With this kind of preconditioner, the triangular solves are replaced by sparse matrix-vector products, which are relatively straightforward to parallelize, and can be vectorized by storing the preconditioner in JAD format.

3 Approximate Inverse Preconditioners.

We now come to the construction of the preconditioner. Obviously, it is desirable that this part of the computation also be performed in parallel. This is especially true for distributed memory computers, since in general both the coefficient matrix and the preconditioner will be distributed among the memories local to each processor. The most popular approach to approximate inverse preconditioning, the SPAI algorithm [12], certainly meets this requirement, and parallel implementations of this technique already exist [1]. We recall that SPAI is based on the idea of Frobenius norm minimization: the sparse approximate in-

verse is computed as the matrix M which minimizes $\|I - MA\|$ (or $\|I - AM\|$ for right preconditioning) subject to some sparsity constraint, see [9],[11],[12]. Here the matrix norm is the Frobenius norm. With this choice, the constrained minimization problem decouples into n independent linear least squares problems (one for each row, or column of M), the number of unknowns for each problem being equal to the number of nonzeros allowed in each row (or column) of M. If the sparsity pattern can be prescribed in advance, the parallel construction of M is straightforward. For general sparse matrices, however, it is very difficult to prescribe the sparsity pattern in advance, and adaptive strategies must be used to capture the "large" entries of the inverse (see [12] for details). As a result, the construction of the approximate inverse becomes very expensive, and the parallel implementation is highly nontrivial [1].

Another important class of methods is that of *factorized* sparse approximate inverse preconditioners, where the approximate inverse is of the form $M = M_U M_L$. Here M_U (M_L) is upper (lower) triangular, and is a sparse approximation of U^{-1} (L^{-1}) where $A = LU$ is the LU factorization of A. It should be emphasized that M_U and M_L can be computed directly from A, without any knowledge of L and U. Methods for computing factorized sparse approximate inverses were first proposed in [14], but current implementations of these techniques rely on a prescribed sparsity pattern and are not well suited for general sparse matrices.

Another approach for computing a factorized approximate inverse is the one based on incomplete (bi)conjugation, described in detail in [3] and [4], and hereafter referred to as the AINV method. This approach does not require that the sparsity pattern be known in advance, and is applicable to matrices with general sparsity patterns. The construction of the AINV preconditioner is based on an algorithm which computes two sets of vectors $\{z_i\}_{i=1}^n$, $\{w_i\}_{i=1}^n$, which are A-biconjugate, i.e. such that $w_i^T A z_j = 0$ if and only if $i \neq j$. Given a nonsingular matrix $A \in \mathbb{R}^{n \times n}$, there is a close relationship between the problem of inverting A and that of computing two sets of A-biconjugate vectors $\{z_i\}_{i=1}^n$ and $\{w_i\}_{i=1}^n$. If we introduce the matrices

$$Z = [z_1,\ z_2, \ldots, z_n] \quad \text{and} \quad W = [w_1,\ w_2, \ldots, w_n],$$

then

$$W^T A Z = D = \mathrm{diag}(p_1, p_2, \ldots, p_n)$$

where $p_i = w_i^T A z_i \neq 0$. It follows that W and Z are necessarily nonsingular and

$$A^{-1} = Z D^{-1} W^T = \sum_{i=1}^{n} \frac{z_i w_i^T}{p_i} \quad . \tag{3.1}$$

Hence, the inverse of A is known if two complete sets of A-biconjugate vectors are known. Assuming that A has an LU factorization, matrices W and Z whose columns are A-biconjugate can be explicitly computed by means of a biconjugation process applied to the standard basis vectors $e_1, \ldots, e_n$. It is easy to see that $Z = U^{-1}$ and $W = L^{-T}$ where $A = LDU$ with L and U unit lower and upper

triangular and D diagonal. If a_i^T denotes the ith row of A, the biconjugation procedure to compute Z can be written as follows:

ALGORITHM 3.1. Left-looking AINV algorithm

1: *Let* $z_1^{(0)} = e_1$; $\quad p_1^{(0)} = a_{11}$

2: **for** $i = 2, \ldots, n$

$z_i^{(0)} = e_i$

for $j = 1, \ldots, i-1$

$p_i^{(j-1)} := a_j^T z_i^{(j-1)}$

$$z_i^{(j)} := z_i^{(j-1)} - \left(\frac{p_i^{(j-1)}}{p_j^{(j-1)}} \right) z_j^{(j-1)}$$

end

$p_i^{(i-1)} := a_i^T z_i^{(i-1)}$

end

The computation of W is identical, except that a_i^T is replaced by c_i^T, where c_i is the ith column of A. Note that Z and W can be computed simultaneously. In order to get a sparse preconditioner, Z and W are computed incompletely, by removing elements less than a prescribed drop tolerance. This leads to incomplete factors $\bar{Z} \approx Z$, $\bar{W} \approx W$ and $\bar{D} \approx D$, and the factorized approximate inverse takes the form $M = \bar{Z}\bar{D}^{-1}\bar{W}^T$. In its original formulation this approach seems to offer little opportunity for parallelization in the preconditioner set-up phase. We point out, however, that it is possible to apply standard graph partitioning techniques in order to increase the parallelism in the construction of the preconditioner [6]. However, in this paper we will not make use of such techniques.

There exist several other sparse approximate inverse techniques, but in this paper we limit ourselves to a comparison between SPAI and AINV. A more complete comparative study of these and other techniques is forthcoming [5].

4 Numerical Experiments.

The numerical experiments were performed on a set of twenty nonsymmetric sparse matrices (mostly drawn from the Harwell-Boeing collection [10], a few from other sources). These matrices originate from a variety of applications such as oil reservoir simulation, circuit design, semiconductor device modelling, etc. Table 1 gives, for each matrix, the order N and number of nonzeros NZ. In addition, the results of runs with a state-of-the-art nonsymmetric solver, Bi-CGSTAB (see [17]), and a standard serial preconditioner, ILU(0) (see [15]), are provided. We report the number of iterations for convergence (Its), the CPU time for constructing the preconditioner (P-time), and the CPU time for the iterative part of the computation (It-time). Convergence is considered attained when the 2-norm of the residual is less than 10^{-9}. The initial guess is taken to be $x_0 = 0$, and the right-hand side is chosen so that the exact solution of the linear system is the vector of all ones. Right preconditioning is used in all cases. The experiments were performed on a single processor of a Cray C98 vector computer at Météo France. All codes were written in Fortran77 and compiled with the

optimization option $-Zv$. Matrix-vector products were vectorized by storing the matrices in jagged diagonal format. The triangular factors in the ILU(0) factorization were stored in CSR format (see [16]), and no attempt was done to vectorize the triangular solves. From the number of iterations required to satisfy the convergence criterion, it can be seen that the test problems considered are, on average, moderately difficult.

Table 1: *Test problems (N = order of matrix, NZ = nonzeros in matrix) and results for Bi-CGSTAB with ILU(0) preconditioning. Timings are in seconds.*

MATRIX	N	NZ	Its	P-time	It-time
3DCD	8000	53600	15	0.058	0.551
ALE3D	1590	45090	38	0.218	0.293
ORSREG1	2205	14133	29	0.016	0.301
ORSIRR1	1030	6858	27	0.008	0.134
ORSIRR2	886	5970	28	0.007	0.121
SAYLR3	1000	3750	33	0.005	0.124
SAYLR4	3564	22316	33	0.025	0.556
ADD32	4960	23884	30	0.044	0.593
ADD20	2395	17319	168	0.082	1.668
MEMPLUS	17758	99147	224	1.911	16.38
SWANG1	3169	20841	6	0.027	0.083
SHERMAN1	1000	3750	33	0.005	0.124
SHERMAN2	1080	23094	9	0.059	0.054
SHERMAN3	5005	20033	68	0.024	1.121
SHERMAN4	1104	3786	27	0.005	0.095
SHERMAN5	3312	20793	30	0.032	0.321
PORES2	1224	9613	26	0.013	0.157
PORES3	532	3474	22	0.004	0.057
WATT1	1856	11360	12	0.012	0.102
WATT2	1856	11550	112	0.015	0.964

In Table 2 we present the results of test runs with Bi-CGSTAB preconditioned with the approximate inverse preconditioners SPAI and AINV. For each of the two methods we give the number of iterations for convergence (Its), the set-up time for the preconditioner (P-time), the time for the preconditioned iterations (It-time). In addition, we give for each matrix the number (A-v) of *amortization vectors* relative to ILU(0), i. e., the number of right-hand sides needed for the corresponding preconditioned iteration to become faster, overall, than Bi-CGSTAB preconditioned with ILU(0). This number is computed under the somewhat unrealistic assumption that each right-hand side will require the same number of iterations, and therefore it should be taken only as an estimate of the number of right-hand sides needed to amortize the cost of the preconditioner construction. A † means failure to attain convergence within 500 iterations. In order to make the comparison with ILU(0) meaningful, we computed sparse approximate inverses having (approximately) the same number of nonzeros as the

original matrix. The only exceptions to this rule were ALE3D, for which good results can be obtained with SPAI and AINV preconditioners having less than half the number of nonzeros as the original matrix, and SAYLR4, for which preconditioners with about twice the number of nonzeros as in the original matrix were used. The SPAI method was implemented in the form described in [12]. Implementation details for AINV are provided in [4].

Table 2: *Results for approximate inverse preconditioners SPAI and AINV.*

	SPAI				AINV			
MATRIX	Its	P-time	It-time	A-v	Its	P-time	It-time	A-v
3DCD	40	10.63	0.111	25	25	1.885	0.068	4
ALE3D	45	30.79	0.088	150	43	1.446	0.094	7
ORSREG1	40	3.309	0.033	13	33	0.550	0.031	2
ORSIRR1	33	2.804	0.020	25	31	0.253	0.019	3
ORSIRR2	36	1.700	0.019	17	30	0.219	0.018	3
SAYLR3	65	0.808	0.031	9	43	0.202	0.021	2
SAYLR4	†	–	–	–	36	0.853	0.073	2
ADD32	6	5.260	0.011	9	6	1.141	0.012	2
ADD20	6	21.95	0.009	14	6	0.726	0.009	1
MEMPLUS	177	354.0	0.944	23	29	9.167	0.272	1
SWANG1	6	5.140	0.009	70	5	0.823	0.008	11
SHERMAN1	62	0.878	0.029	10	43	0.201	0.021	2
SHERMAN2	†	–	–	–	†	–	–	–
SHERMAN3	123	3.647	0.166	4	98	1.074	0.152	2
SHERMAN4	39	0.736	0.018	10	36	0.230	0.018	3
SHERMAN5	44	11.96	0.066	47	43	0.865	0.065	4
PORES2	†	–	–	–	88	0.361	0.084	5
PORES3	111	0.941	0.044	73	75	0.127	0.038	7
WATT1	15	2.300	0.014	27	18	0.471	0.017	6
WATT2	377	2.590	0.384	5	111	0.505	0.116	1

Several observations can be made based on these experimental results. From the point of view of robustness, we note that AINV fails on matrix SHERMAN2 whereas SPAI fails in three cases (SAYLR4, SHERMAN2 and PORES2). However, SPAI can be made to succeed on SHERMAN2 by computing a substantially more dense approximate inverse, and also on PORES2 by computing a left approximate inverse, see [12]. Hence, based on this set of test problems, ILU(0), SPAI and AINV all have comparable robustness. Concerning the rate of convergence, we see that the sparse approximate inverse preconditioners, on average, are comparable with ILU(0), with AINV being somewhat better than SPAI. This is in keeping with the observation that, in general, a factorized form gives a better approximation of the inverse than a non-factorized form with the same number of nonzeros [8]. However, when we look at the time for the iterative part,

we see that SPAI and AINV are both much faster than ILU(0), due to the good vectorization features of the approximate inverse preconditioners in the iterative part of the computation. The price to pay for this fast execution of the iterative part is the cost of the preconditioner construction, which is much higher for the approximate inverse methods than for ILU(0). In this uniprocessor implementation SPAI is by far the most expensive method, in some cases requiring more than 30 times longer than AINV, but it must be kept in mind that SPAI was designed for use on massively parallel architectures. In a parallel implementation SPAI would certainly look much better, although parallel implementations of AINV are possible as well [5]. Furthermore, we observe that whereas the computation of the ILU(0) and AINV preconditioners did not vectorize, there was some benefit from vectorization in the computation of SPAI. This is due to the use of high-level BLAS in the SPAI algorithm. Finally, we notice that both methods can be superior to ILU(0) in the common situation were the preconditioner can be reused for solving many systems having the same coefficient matrix but different right-hand sides (given sequentially), provided that the number of such right-hand sides is sufficiently high. We notice that for AINV this number is always quite small—indeed, there are three test problems for which AINV is superior to ILU(0) even for a single right-hand side. The number of amortization vectors for SPAI is higher, but not unreasonable.

5 Conclusions.

In this paper we have presented the results of experiments, carried out on a Cray C98 vector processor, with two implementations of the sparse approximate inverse preconditioners SPAI and AINV. Based on our experiments, we conclude that these techniques are comparable from the point of view of robustness and rates of convergence, with AINV being somewhat better on average. For the common situation where the preconditioner can be reused for solving a sequence of linear systems with the same matrix and different right-hand sides, both methods can be superior to standard ILU techniques even on a single vector processor. Finally, it was found that the computation of the preconditioner is much more expensive for SPAI than for AINV, but it was observed that the situation could be different in a parallel implementation. We conclude that both techniques offer excellent potential for use on high-performance computers.

REFERENCES

1. S. T. Barnard and R. L. Clay, *A portable MPI implementation of the SPAI preconditioner in ISIS++*. Proceedings of the Eight SIAM Conference on Parallel Processing for Scientific Computing, M. Heath et al., eds., SIAM, Philadelphia, 1997.
2. R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems*, SIAM, Philadelphia, 1994.
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 and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems,* SIAM J. Sci. Comput., to appear.
5. M. Benzi and M. Tůma, *A comparative study of sparse approximate inverse preconditioners,* in preparation.
6. M. Benzi and M. Tůma, *The effect of ordering and partitioning on factorized approximate inverse preconditioners,* in preparation.
7. T. Chan, W.-P. Tang, and W. Wan, *Wavelet sparse approximate inverse preconditioners,* BIT, 37 (1997), pp. 644–660.
8. E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., to appear.
9. J. D. F. Cosgrove, J. C. Diaz, and A. Griewank, *Approximate inverse preconditionings for sparse linear systems,* Intern. J. Computer Math., 44 (1992), pp. 91–110.
10. I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems,* ACM Trans. Math. Software, 15 (1989), pp. 1–14.
11. N. I. M. Gould and J. A. Scott, *On approximate-inverse preconditioners,* Technical Report RAL-95-026, Rutherford Appleton Laboratory, Chilton, England, 1995.
12. M. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses,* SIAM J. Sci. Comput., 18 (1997), pp. 838–853.
13. M. A. Heroux, P. Vu, and C. Yang, *A parallel preconditioned conjugate gradient package for solving sparse linear systems on a Cray Y-MP,* Appl. Num. Math., 8 (1991), pp. 93–115.
14. L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditioning I: Theory,* SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.
15. 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.
16. Y. Saad, *Iterative Methods for Sparse Linear Systems,* PWS Publishing Company, Boston, 1996.
17. H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems,* SIAM J. Sci. Stat. Comput., 12 (1992), pp. 631–644.