

A Two-Level Parallel Preconditioner Based on Sparse Approximate Inverses

MICHELE BENZI JOSÉ MARÍN MIROSLAV TŮMA

Dedicated to David M. Young on the occasion of his 75th birthday

Abstract

We introduce a novel strategy for parallel preconditioning of large-scale linear systems by means of a two-level factorized sparse approximate inverse algorithm. Using graph partitioning and incomplete biconjugation we are able to obtain a highly parallel preconditioner. The algorithm has been implemented using MPI on a SGI Origin 2000 computer at Los Alamos National Laboratory and is currently being used to solve unstructured linear systems with up to a few million unknowns from a variety of applications. The numerical experiments demonstrate the excellent scalability of the algorithm for sufficiently large problems.

Keywords: sparse linear systems, preconditioned iterative methods, AINV, graph partitioning

1 Introduction

The problem of devising robust preconditioning methods for general linear systems that can be efficiently implemented on current parallel architectures continues to attract considerable attention. A promising approach, which has seen a great deal of activity in recent years, is the one based on direct approximations to the inverse of the coefficient matrix A . With this approach, a sparse matrix $M \approx A^{-1}$ is explicitly computed and stored and used as a preconditioner for Krylov subspace methods for the solution of $Ax = b$. The main advantage of this approach is that the preconditioning operation reduces to a matrix-vector product, an operation that is easily vectorized and parallelized. This is in contrast with more traditional general-purpose preconditioners, like those based on incomplete factorizations of the coefficient matrix A , which require sparse triangular solves to be performed at each iteration. Because triangular solves are inherently sequential operations, it is difficult to efficiently implement incomplete factorization preconditioners on vector and parallel computers, particularly for irregular problems such as those arising from unstructured grid computations. Besides parallel processing, another advantage of sparse approximate inverse techniques is that they are generally less prone to instabilities than incomplete factorization methods, and therefore they sometimes succeed in solving difficult problems for which more standard methods are ineffective. A number of different sparse approximate inverse algorithms have been proposed in the literature. Some of

the main references include [1], [2], [4], [9], [10], [11], [12], [14], [15], [16], [18]. In particular, [4] provides a fairly complete survey of the existing algorithms, together with a comparison on a broad range of test problems.

In this contribution we focus on the AINV preconditioner [1], [2]. This method, which is based on incomplete (bi)conjugation, is fairly robust (comparable to ILU-type methods), delivers good convergence rates, and has low set-up costs. In our experience, AINV is one of the most competitive approximate inverse techniques [3], [4]. We review this method briefly in Section 2. The main drawback of AINV is that the construction of the preconditioner rests on a generalization of the Gram–Schmidt orthogonalization process, which is highly sequential in nature. Thus, the parallel computation of the preconditioner presents a challenge, particularly in a distributed environment. In this paper we show how this limitation can be removed using graph partitioning (Sections 3–4). The combination of the AINV technique with graph partitioning results in a highly parallel preconditioner. We have implemented the preconditioner on a SGI Origin 2000 using Fortran 90 and MPI [19], using `kmetsis` [17] for the graph partitioning. The code has been tested on a variety of large-scale linear systems arising in real-life applications. Our experimental results (reported in Section 5) demonstrate the excellent scalability features of our algorithm, provided that the problem has sufficiently large size.

Some conclusions and suggestions for future work are given in Section 6.

2 The standard AINV algorithm

The AINV algorithm [1], [2] constructs a factorized sparse approximate inverse of the form

$$M = ZD^{-1}W^T \approx A^{-1}$$

where Z, W are unit upper triangular matrices and D is diagonal. The approximate inverse factors W^T and Z are sparse approximations of the inverses of the L and U factors, respectively, in the LDU decomposition of A . The AINV algorithm computes Z, W and D directly from A by means of an incomplete biconjugation process, in which small elements are dropped to preserve sparsity. In order to describe the procedure, let a_i^T and c_i^T denote the i th row of A and A^T , respectively (i.e., c_i is the i th column of A). Also, let e_i denote the i th unit basis vector. The basic A -biconjugation procedure can be written as follows.

Algorithm 2.1 Biconjugation algorithm

- (1) Let $w_i^{(0)} = z_i^{(0)} = e_i$ ($1 \leq i \leq n$)
- (2) For $i = 1, 2, \dots, n$ do
- (3) For $j = i, i + 1, \dots, n$ do
- (4) $p_j^{(i-1)} := a_i^T z_j^{(i-1)}$; $q_j^{(i-1)} := c_i^T w_j^{(i-1)}$
- (5) End do
- (6) if $i = n$ go to (11)
- (7) For $j = i + 1, \dots, n$ do
- (8) $z_j^{(i)} := z_j^{(i-1)} - \left(\frac{p_j^{(i-1)}}{p_i^{(i-1)}} \right) z_i^{(i-1)}$; $w_j^{(i)} := w_j^{(i-1)} - \left(\frac{q_j^{(i-1)}}{q_i^{(i-1)}} \right) w_i^{(i-1)}$
- (9) End do
- (10) End do
- (11) Let $z_i := z_i^{(i-1)}$, $w_i := w_i^{(i-1)}$ and $p_i := p_i^{(i-1)}$, for $1 \leq i \leq n$. Return $Z = [z_1, z_2, \dots, z_n]$, $W = [w_1, w_2, \dots, w_n]$ and $D = \text{diag}(p_1, p_2, \dots, p_n)$.

Sparsity is preserved by dropping in the z - and w -vectors after the updates at step (8). If $A = A^T$ then $Z = W$ and the columns of Z are (approximately) A -conjugate. The incomplete procedure is well-defined, i.e., no breakdown can occur, if A is an H-matrix. In the general case, diagonal shifts may be necessary in order to prevent breakdowns. See [1], [2] for details.

Direct comparisons [3], [4] have demonstrated that this method is significantly faster than approximate inverse preconditioners based on norm-minimization techniques like SPAI [15] on scalar and vector computers. This is due to the fact that the serial arithmetic cost of the preconditioner construction is much smaller for AINV than for most competing methods and, to a lesser extent, to the fact that AINV results in faster convergence on average. Being a factorized preconditioner, AINV can be used with the conjugate gradient (CG) method for solving symmetric positive definite (SPD) problems, unlike SPAI. Also, sparse matrix reorderings can be used to increase the efficiency of the preconditioner; see [7], [5]. In contrast to other factorized sparse approximate inverse preconditioners, like the FSAI method [18], it is not necessary to prescribe the sparsity pattern of the preconditioner factors in advance. Indeed, significant entries in the inverse factors are automatically captured by means of a drop tolerance, thus making the preconditioner well-suited for problems with general sparsity patterns.

As already indicated in the Introduction, a drawback of AINV is that, as it stands, the (bi)conjugation process is highly sequential. However, it is possible to exploit certain sparse matrix reorderings, such as those induced by graph partitioning, to introduce parallelism in the preconditioner construction phase. This idea was first put forth in [5]; the remaining of the present paper is devoted to investigating and testing such idea.

3 Graph partitioning

Graph partitioning has become a universal tool in parallel computing, where it is routinely used for partitioning a problem among processors in a parallel environment. Here we will describe how it can be used in conjunction with the AINV algorithm to get a highly parallel, two-level preconditioner. Besides allowing the introduction of parallelism in the preconditioner construction phase, this approach is very natural since in most parallel applications the computational mesh is typically partitioned at the outset by some form of graph partitioning.

Recall that any sparse, structurally symmetric matrix has an associated adjacency graph $G = (V, E)$ where $V = \{1, \dots, n\}$ is the vertex (or node) set and E is the set of edges $\langle i, j \rangle$ with $i, j \in V$ such that $a_{ij} \neq 0$. Note that the graph is undirected: that is, no distinction is made between $\langle i, j \rangle$ and $\langle j, i \rangle$. If A is not structurally symmetric, we associate to it the adjacency graph of $A + A^T$. Graph partitioning algorithms split the graph in such a way that the resulting p subgraphs are of roughly equal size and the number of edge cuts is approximately minimized. Nodes which are connected by cut edges are removed from the subgraphs and put in the separator node set. By numbering the nodes by subgraphs and taking the separator nodes last, the matrix is permuted into the following block angular form:

$$P^T A P = \begin{pmatrix} A_1 & & & B_1 \\ & A_2 & & B_2 \\ & & \ddots & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_S \end{pmatrix}$$

where P is a permutation matrix. The diagonal blocks A_1, \dots, A_p correspond to subgraphs

induced by the interior nodes in the graph decomposition, the off-diagonal blocks B_i and C_i represent the connections between the subgraphs, and A_S the connections between nodes in the separator set. Such matrix structure frequently arises from the application of domain decomposition with no overlap (also known as substructuring) for the solution of discretized PDEs. Graph partitioning algorithms, however, are applicable to arbitrary sparse matrices and make no use of underlying geometric information—indeed, there may be no underlying physical grid.

Consider now a distributed parallel system consisting of p computational nodes N_1, \dots, N_p ; each node can either be a single processor or, in some cases, a “box” comprising several processors. (For example, the ASCI Blue Mountain massively parallel system at Los Alamos National Laboratory is presently configured as a cluster of $p = 48$ SGI Origin 2000 multiprocessors, each consisting of 128 processors, for a total of 6144 processors.) Using graph partitioning, the problem can be split between the p nodes so that node N_i holds A_i, B_i and C_i , $1 \leq i \leq p$. One of the nodes, marked N_S , should also hold A_S . As we shall see in the next section, this distribution of A allows the parallel computation of a factorized sparse approximate inverse of $P^T AP$. Here we address the question of computing matrix–vector products, an operation that is required at each step of a Krylov subspace method. Assuming that the vector x is partitioned conformally to $P^T AP$, with node N_i holding the subvector x_i , the matrix–vector product of $P^T AP$ times x is:

$$\begin{pmatrix} A_1 & & & B_1 \\ & A_2 & & B_2 \\ & & \ddots & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_S \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ x_S \end{pmatrix} = \begin{pmatrix} A_1 x_1 + B_1 x_S \\ A_2 x_2 + B_2 x_S \\ \vdots \\ A_p x_p + B_p x_S \\ \sum_{i=1}^p C_i x_i + A_S x_S \end{pmatrix}.$$

It is clear that the first p components of the product can be computed completely in parallel, provided that each node N_i holds a copy of x_S ; hence, at each iteration, it is necessary to broadcast a copy of x_S from node N_S to each remaining node. The last component of the product, on the other hand, requires the computation of the local contributions $C_i x_i$ ($1 \leq i \leq p$), which is done completely in parallel, followed by a reduction (fan-in) across nodes to accumulate the local contributions on node N_S , which in the meantime has calculated its own contribution $A_S x_S$. The fan-in requires $k = \log_2 p$ steps. It is already clear from this discussion that good parallel efficiency requires that the size of x_S , which is the number of graph vertices in the separator set, is as small as possible; as we will see, this condition is also crucial for having an efficient parallel construction of the preconditioner. Another obvious requirement is good load balancing, which means that the A_i blocks ($1 \leq i \leq p$) should have approximately the same number of nonzero entries. These are precisely the goals that good graph partitioning algorithms strive to achieve.

For a fixed graph, the number of nodes in the separator set increases with the number p of partitions, while the number of interior nodes in each partition decreases. Because good parallel efficiency requires that the number of nodes in each partition (the order of each A_i) be large relative to the number of nodes in the separator set (the order of A_S), it is obvious that for a given $N \times N$ problem there is a maximum number $p \ll N$ of processors that can be used efficiently, and increasing p after that point will actually decrease the efficiency of the algorithm.

4 The two-level approximate inverse algorithm

It is easy to see that the inverse of the permuted matrix $P^T AP$ can be decomposed as

$$(P^T AP)^{-1} = U^{-1}L^{-1} = \begin{pmatrix} U_1^{-1} & & & E_1 \\ & U_2^{-1} & & E_2 \\ & & \ddots & \vdots \\ & & & U_p^{-1} & E_p \\ & & & & U_S^{-1} \end{pmatrix} \begin{pmatrix} L_1^{-1} & & & & \\ & L_2^{-1} & & & \\ & & \ddots & & \\ & & & L_p^{-1} & \\ F_1 & F_2 & \dots & F_p & L_S^{-1} \end{pmatrix},$$

where

$$A_i = L_i U_i, \quad E_i = -U_i^{-1} L_i^{-1} B_i U_S^{-1}, \quad F_i = -L_S^{-1} C_i U_i^{-1} L_i^{-1}$$

and L_S, U_S are the triangular factors of the Schur complement matrix

$$S = A_S - \sum_{i=1}^p C_i A_i^{-1} B_i.$$

An important observation is that L^{-1}, U^{-1} preserve a good deal of sparsity, since fill-in can occur only within the nonzero blocks [5]. This is in contrast with A^{-1} , which is generally completely full (the same is true, of course, for the inverse of $P^T AP$). Additional sparsity can be obtained by reordering the diagonal blocks A_i , for instance with Minimum Degree, Generalized Nested Dissection, or a recursive application of graph partitioning. (The latter strategy would be the natural choice in the case where each computational node N_i is itself a multiprocessor, since in this case several processors would be assigned to each diagonal block A_i and graph partitioning would be needed to exploit this further level of parallelism.)

Even though the inverse factors of $P^T AP$ preserve a good deal of sparsity, it is generally too expensive to compute them exactly. Instead, we compute sparse approximations $Z \approx U^{-1}$ and $W^T \approx L^{-1}$ so that a sparse approximate inverse of $P^T AP$ is obtained. This approximate inverse is used as a preconditioner for an iterative method applied to

$$(P^T AP)y = P^T b, \quad y = P^T x$$

whence the solution of the original linear system $Ax = b$ is obtained as $x = Py$.

The parallel computation of the approximate inverse factors $Z \approx U^{-1}$ and $W^T \approx L^{-1}$ proceeds as follows. Recall that node N_i holds A_i, B_i, C_i ($1 \leq i \leq p$). One of these nodes, marked N_S , also holds A_S . First, each node N_i computes sparse triangular matrices $Z_i \approx U_i^{-1}$ and $W_i^T \approx L_i^{-1}$ such that $Z_i W_i^T \approx A_i^{-1}$ using the standard AINV algorithm (Algorithm 2.1), in parallel for $i = 1, \dots, p$. Note that for notational convenience we are now absorbing the diagonal factor D_i^{-1} in $A_i^{-1} \approx Z_i D_i^{-1} W_i^T$ in either one of the triangular factors. Once this is done, each node computes the local contribution $C_i Z_i W_i^T B_i \approx C_i A_i^{-1} B_i$ to the approximate Schur complement

$$\hat{S} := A_S - \sum_{i=1}^p C_i Z_i W_i^T B_i \approx S$$

which is then accumulated on node N_S by a fan-in across nodes in $k = \log_2 p$ steps. At this point a sparse approximate inverse factorization of the approximate Schur complement is computed, sequentially, on node N_S , resulting in sparse triangular matrices $Z_S \approx U_S^{-1}$ and $W_S^T \approx L_S^{-1}$. The two-level nature of this preconditioner is now apparent: at a first level, sparse approximate

inverses of the diagonal blocks A_i are computed; the second level consists of the computation of a sparse approximate inverse of the approximate Schur complement. The factorized sparse approximate inverse $M \approx (P^T AP)^{-1}$ is then

$$M := ZW^T = \begin{pmatrix} Z_1 & & & \hat{E}_1 \\ & Z_2 & & \hat{E}_2 \\ & & \ddots & \vdots \\ & & & Z_p & \hat{E}_p \\ & & & & Z_S \end{pmatrix} \begin{pmatrix} W_1^T & & & & \\ & W_2^T & & & \\ & & \ddots & & \\ & & & W_p^T & \\ \hat{F}_1 & \hat{F}_2 & \dots & \hat{F}_p & W_S^T \end{pmatrix},$$

where $\hat{E}_i = -Z_i W_i^T B_i Z_S$ and $\hat{F}_i = -W_S^T C_i Z_i W_i^T$. Notice that it is not necessary to compute the off-diagonal blocks \hat{E}_i and \hat{F}_i explicitly, since the application of the preconditioner only requires matrix–vector products.

Again, because of the sequential bottleneck associated with the (approximate) Schur complement calculation, it is in the interest of parallel efficiency to keep the separator set small relative to the partitions. This is also necessary for the efficient execution of the matrix–vector products with the approximate inverse factors W^T and Z , which are performed similarly to the matrix–vector product with $P^T AP$ illustrated in the previous section.

A fundamental question that arises in connection with this preconditioning strategy concerns the effect of the symmetric permutation $A \rightarrow P^T AP$ induced by the graph partitioning on the quality of the preconditioner as measured by the rate of convergence of the preconditioned iteration. As we will see in the next section, it turns out that in most cases the number of iterations is scarcely affected by the reordering. Indeed, there are many cases in which the number of iterations tends to decrease as the number of partitions p increases. This is because of the approximate Schur complement contribution to the preconditioner, which plays the role of a coarse grid correction and allows a global exchange of information between the computational subdomains.

Hence, we observe here a remarkable situation: we can use sparse matrix reorderings to increase parallelism in the preconditioner while at the same time reducing fill-in in the incomplete inverse factors, without negatively affecting the rate of convergence! This situation should be compared with that for ILU-type preconditioners, where it is known that parallel orderings (such as multicoloring) and fill-reducing orderings (like Minimum Degree or Nested Dissection) often have a detrimental impact on the rate of convergence of the preconditioned iteration [13].

We mention two potential problems that could arise with the approach outlined above. First, the approximate Schur complement \hat{S} could in principle become unacceptably dense, posing storage problems and increasing set-up costs. Second, it could happen that some of the local approximate inverse factors Z_i, W_i are considerably denser than others, therefore causing a load imbalance which will deteriorate the parallel efficiency of the preconditioner. Both problems can be mitigated to some extent by modifying somewhat the algorithm, but since they have not occurred in practice (so far), we do not discuss them further.

5 Numerical experiments

We coded the parallel preconditioner in Fortran 90 using MPI [19]. The platform used is a SGI Origin 2000 multiprocessor. For the graph partitioning we used `kmetis` [17], available in the public domain. A number of tests on matrices from real applications (mostly LANL codes)

have been performed. Here we report on a selection of results that are representative of the behavior observed. A description of the test problems is provided in Table 1 below. Here n is the problem size and nnz the number of nonzeros in the matrix. The first four problems are symmetric positive definite, the others nonsymmetric.

Table 1: Test problems information.

Matrix	n	nnz	Application
DANTE	22,065	268,945	Neutron transport
ATTILA	61,760	882,108	Nuclear well logging
AUGUSTUS	134,144	1,510,400	Diffusion (two materials)
KOHN-SHAM	2,001,000	10,000,996	Quantum chemistry
ALE3D	22,200	1,188,152	Metal casting
VENKAT01	62,424	1,717,792	CFD (Euler solver)
YUCCA	175,885	1,115,627	Groundwater flow
CONV-DIFF	1,000,000	4,996,000	CFD (convection-diffusion)

Most of these matrices arise from finite element discretizations of three-dimensional problems on unstructured meshes. In some cases the coefficients in the corresponding partial differential equation are highly discontinuous. The hardest problem in this set is perhaps ALE3D, which is strongly indefinite (about 600 of the diagonal entries are zero). We were not able to solve this problem using various $ILU(k)$ and $ILUT$ preconditioners. A standard AMG (Algebraic Multigrid) algorithm was also unsuccessful.

The following table contains the results of our experiments on a SGI Origin 2000 at LANL. For the SPD problems, conjugate gradient acceleration was used, whereas Bi-CGSTAB [20] was the iterative method used for the nonsymmetric problems. In all cases the stopping criterion was a reduction of the initial residual norm by at least eight orders of magnitude. The initial guess was $x_0 = 0$ and an artificial right-hand side was chosen so that the solution was the vector of all ones. For the drop tolerance, the standard value $\tau = 0.1$ was used with AINV in all cases except for the runs with ATTILA, where we used $\tau = 0.05$, and with ALE3D on 8 and 16 processors where the occurrence of pivot breakdowns forced us to use $\tau = 0.3$ and $\tau = 0.25$, respectively. As usual, the coefficient matrix was first rescaled by dividing each entry by $\max_{i,j} |a_{ij}|$ so that all entries of the scaled matrix lie in the interval $[-1, 1]$. The same value of τ was always used for both the diagonal blocks A_i and the approximate Schur complement \hat{S} , because numerical experiments suggested that there is no gain in using different values of τ in the two phases. With the exception of the YUCCA and CONV-DIFF problems, the value $\tau = 0.1$ results in preconditioners which are considerably sparser than the original matrix. The same choice $\tau = 0.1$ has also been recommended by other authors (see [7] and [6]).

In Table 2 we provide for each matrix and for each number p of processors (partitions) used the number of nonzeros in the approximate inverse factors (in thousands) with the time to compute the preconditioner (above), and the number of iterations with the corresponding timing (below). Timings are wall-clock times and are measured in seconds. The compiler option -O3 was used. The time for the graph partitioning is not included, but it is always negligible relative to the total solve time. Also, in a real application, the connectivity information about the matrix is usually already partitioned among processors, and the calculation of the matrix elements is also done in parallel.

Table 2: Test results for different numbers of processors.

p	2	4	8	16	32	64
DANTE	87.6/2.01 62/1.90	83.6/1.12 60/0.86	81.9/0.84 59/0.69	77.8/0.57 57/0.76	– –	– –
ATTILA	259/3.74 246/24.4	257/2.05 247/10.2	255/1.26 240/5.90	251/1.16 228/5.28	– –	– –
AUGUSTUS	671/7.05 184/48.1	660/3.52 193/24.0	648/1.91 192/10.5	628/1.44 190/7.32	609/1.42 181/8.27	– –
KOHN-SHAM	8567/36.3 1584/7398	8551/18.1 1584/3592	8535/9.1 1594/1589	8517/4.8 1586/818	8490/2.7 1584/440	8461/1.8 1582/277
ALE3D	330/17.9 575/91.6	313/18.2 538/46.2	91/8.8 612/28.8	92/13.4 802/42.3	– –	– –
VENKAT01	907/14.2 26/8.79	898/7.37 26/4.18	882/4.08 27/2.04	862/2.63 26/1.13	830/2.49 23/0.98	– –
YUCCA	2158/9.85 50/37.1	2126/5.41 46/17.0	1964/3.37 56/10.2	1859/3.17 63/8.39	1749/3.23 53/7.86	– –
CONV-DIFF	5988/34.3 666/2377	5972/17.4 881/1437	5961/8.6 844/654	5937/4.6 758/424	5907/3.0 669/257	5863/2.72 818/207

From these results we see that for most problems, the algorithm exhibits good scalability properties as long as the problem size is significant relative to the number of processors used. The reason is clear: for a small number of processors the (approximate) Schur complement is very small compared to the size of each of the p diagonal blocks. Hence, the sequential part of the computation is very small relative to the parallel part, and good parallel efficiency is achieved. When the number of processors grows, however, the relative size of the approximate Schur complement quickly approaches and eventually surpasses the size of the diagonal blocks (which is decreasing), and the efficiency decreases. We have invariably observed that once the approximate Schur complement size reaches that of the diagonal blocks, the total computational time begins to increase rather than to decrease, signaling the fact that the process is now dominated by the sequential part of the computation and by the communication. All this holds for a fixed problem size; if we double the problem size each time we double the number of processors, as is required in a scalability study, then the Schur complement can be kept small relative to the diagonal blocks and a good parallel efficiency is retained. We observed this in particular for a suite of problems of the KOHN-SHAM type of order ranging between approximately 60,000 and 2,000,000 using 2, 4, 8, 16, 32 and 64 processors, respectively. In this sense, our algorithm appears to be perfectly scalable, provided of course that each processor has enough work to do. In the case of matrix ALE3D, which is somewhat atypical, we see virtually no improvement in the timing for computing the preconditioner when going from 2 to 4 processors (cf. Table 2). This is due to the fact that the approximate inverse is very sparse, and the Schur complement computation dominates this phase of the solution process.

To better illustrate the behavior of our algorithm, we consider the AUGUSTUS problem in more detail. In Table 3 we report for several values of p the number $nnz(Z)$ of nonzeros in the approximate inverse factor Z , the number $nnz(Z_S)$ of nonzeros in the approximate inverse factor of the Schur complement Z_S , the average order $|A_i|$ of the diagonal block A_i , the number σ of nodes in the separator set (which is equal to the order of the approximate Schur complement), and the total solution time T_p . It can be seen that the total solution time decreases as long as $|A_i| > \sigma$, and it increases as soon as $|A_i| < \sigma$. This kind of behavior is typical.

Table 3: Additional results for AUGUSTUS.

p	2	4	8	16	32
$nnz(Z)$	671150	659551	648413	628025	609371
$nnz(Z_S)$	1974	4541	6782	11268	15712
$ A_i $	66405	32849	16262	7976	3915
σ	1334	2747	4050	6527	8863
T_p	55.1	27.5	12.0	8.8	9.7

As already mentioned, we see that the rate of convergence (number of iterations to converge) is not strongly affected by the number of processors used. In several cases, the number of iterations tends to decrease with increasing p , and so does the number of nonzeros in the preconditioner. (These considerations do not apply to ALE3D, for which we had to use different values of the drop tolerance for $p = 8, 16$.) The problem that exhibited the strongest sensitivity to p is CONV-DIFF, for which we observe some fairly strong fluctuations of the number of iterations for different values of p . This is probably due to the fact that in some cases the graph partitioning results in a decomposition of the computational grid into subdomains the boundaries of which cut the convection streamlines. For this kind of problems, some sort of weighted graph partitioning heuristic should be used.

Concerning the performance of our code, its Mflops rate per processor is currently somewhat disappointing, typically of the order of about 10 Mflops/processor. For comparison, the usual DGEMM routine for the product of two dense matrices runs at about 270 Mflops on one processor of the SGI Origin 2000. Unfortunately, this low performance is not uncommon for applications involving sparse, unstructured matrices. Better performance could be achieved by using blocking, a technique that avoids indirect addressing in the innermost loop and allows for in-cache data reuse. A blocked version of AINV has been implemented in [8] for use within a quantum chemistry code, with excellent results. We plan to use a similar technique in a future implementation of our method.

We conclude this section with a comment on diagonal (Jacobi) scaling, the simplest parallel preconditioner. Generally speaking, AINV was found to be more robust than diagonal scaling, especially for problems which are not SPD. Note that, for instance, diagonal scaling cannot be applied to the ALE3D problem, which has zero entries on the main diagonal. Moreover, because of the better rate of convergence obtained, AINV is usually faster, though the difference between the two techniques can vary greatly depending on the problem. For a comparison on both sequential and parallel computers between diagonal preconditioning and the standard AINV algorithm on several finite difference and finite element matrices, see [6].

6 Conclusions and future work

In this paper we have described a novel approach to parallel preconditioning for large, sparse matrices with general sparsity patterns. A highly parallel, two-level algorithm is obtained by combining graph partitioning with the AINV sparse approximate inverse technique. An important feature of this method is that for most problems, the rate of convergence does not deteriorate as the number of partitions (processors) increases; furthermore, the number of nonzeros in the preconditioner tends to decrease as the number of partitions increases. Good parallel efficiency is observed provided that the problem size is sufficiently large relative to the

number of partitions. Our parallel code is being used to solve large linear systems from a variety of applications, most of which involve unstructured grids. With this approach we have been able to solve difficult problems which could not be solved by other techniques. Unlike other algebraic preconditioners, ours results in low set-up costs. We also stress that our preconditioner is easy to use since it requires the input of a single user-defined parameter: the drop tolerance τ . Our experience is that the code is fairly robust with respect to the choice of τ , and we recommend $\tau = 0.1$ as the default value.

Among the issues that warrant further investigation we mention the possibility of improving the local performance of the code by use of blocking, and the study of systematic solutions to the problem of possible (near) breakdown of the local AINV processes.

Acknowledgments. Parts of this work were completed during visits by the second and third author at Los Alamos National Laboratory in the Summer and Fall of 1998. The hospitality and support provided by LANL are greatly appreciated. The work of the first author was supported in part by the Department of Energy through grant W-7405-ENG-36 with Los Alamos National Laboratory. The second author was supported in part by the same grant and also by the DGES of Spain through project No. PB97-0334. The third author was supported in part by the Grant Agency of the Czech Republic through grant No. 205/96/0921 and by the Czech Academy of Sciences through grant No. 2030706. We are grateful to the Advanced Computing Laboratory at Los Alamos National Laboratory for granting access to computing resources.

References

- [1] Benzi, M., Meyer, C. D., and Tũma, M., A sparse approximate inverse preconditioner for the conjugate gradient method, *SIAM J. Sci. Comput.* **17** (1996), 1135–1149.
- [2] Benzi, M. and Tũma, M., A sparse approximate inverse preconditioner for nonsymmetric linear systems, *SIAM J. Sci. Comput.* **19** (1998), 968–994.
- [3] Benzi, M. and Tũma, M., Numerical experiments with two approximate inverse preconditioners, *BIT* **38** (1998), 234–241.
- [4] Benzi, M. and Tũma, M., A comparative study of sparse approximate inverse preconditioners, *Appl. Numer. Math.* (1999), to appear.
- [5] Benzi, M. and Tũma, M., Orderings for factorized sparse approximate inverse preconditioners, *SIAM J. Sci. Comput.*, to appear.
- [6] Bergamaschi, L., Pini, G., and Sartoretto, F., Approximate inverse preconditioning in the parallel solution of sparse eigenproblems, Preprint, Dipartimento di Metodi e Modelli Matematici per le Scienze Applicate, Università di Padova, Italy, 1998.
- [7] Bridson, R. and Tang, W.-P., Ordering, anisotropy and factored sparse approximate inverses, Preprint, Department of Computer Science, University of Waterloo, Canada (1998).
- [8] Challacombe, M., A simplified density matrix minimization for linear scaling SCF theory, *J. Chem. Phys.* (1999), to appear.
- [9] Chan, T., Tang, W.-P., and Wan, W., Wavelet sparse approximate inverse preconditioners, *BIT* **37** (1997), 644–660.

- [10] Chow, E. and Saad, Y., Approximate inverse preconditioners via sparse-sparse iterations, *SIAM J. Sci. Comput.* **19** (1998), 995–1023.
- [11] Chow, E. and Saad, Y., Approximate inverse techniques for block-partitioned matrices, *SIAM J. Sci. Comput.* **18** (1997), 1657–1675.
- [12] Cosgrove, J. D. F., Díaz, J. C., and Griewank, A., Approximate inverse preconditioning for sparse linear systems, *Internat. J. Comput. Math.* **44** (1992), 91–110.
- [13] Duff, I. S. and Meurant, G., The effect of ordering on preconditioned conjugate gradients, *BIT* **29** (1989), 635–657.
- [14] Gould, N. I. M. and Scott, J. A., Sparse approximate-inverse preconditioners using norm-minimization techniques, *SIAM J. Sci. Comput.* **19** (1998), 605–625.
- [15] Grote, M. and Huckle, T., Parallel preconditioning with sparse approximate inverses, *SIAM J. Sci. Comput.* **18** (1997), 838–853.
- [16] Kaporin, I. E., New convergence results and preconditioning strategies for the conjugate gradient method, *Numer. Linear Algebra Appl.* **1** (1994), 179–210.
- [17] Karypis, G. and Kumar, V., A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.* **20** (1999), 359–392.
- [18] Kolotilina, L. Yu. and Yeremin, A. Yu., Factorized sparse approximate inverse preconditioning I: Theory, *SIAM J. Matrix Anal. Appl.* **14** (1993), 45–58.
- [19] Message Passing Interface Forum. MPI: A message-passing interface standard. Computer Science Dept. Technical Report CS-94-230, University of Tennessee, Knoxville, TN, April 1994.
- [20] van der Vorst, H. A., 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), 631–644.

Michele Benzi

Scientific Computing Group (CIC-19)
Mail Stop B256
Los Alamos National Laboratory
Los Alamos, NM 87545 USA
benzi@lanl.gov

José Marín

Department of Applied Mathematics
Polytechnic University of Valencia
46071 Valencia, Spain
jmarinma@mat.upv.es

Miroslav Tůma

Institute of Computer Science
Academy of Sciences of the Czech Republic
18207 Prague 8, Czech Republic
tuma@uivt.cas.cz