

A block iterative method for general sparse equations

Mario Arioli[†], Iain Duff^{*}, Joseph Noailles^{**}, and Daniel Ruiz

CERFACS, 42 av. G. Coriolis, 31057 Toulouse Cedex, France

Abstract

We describe a block version of Cimmino's algorithm for solving general sparse equations. We consider its acceleration using conjugate gradients and the choice of a preconditioner for the method. The effect of the block partitioning is also discussed. Experiments on an Alliant FX/80 indicate both the parallelism and effectiveness of our method on quite general systems.

1 Introduction

We consider a block iterative method for the solution of the linear equations

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

where A is a large nonsingular sparse unsymmetric matrix of order n .

Our method is based on a block version of Cimmino's method, where the blocks are obtained by partitioning A into strips. We choose to solve the undetermined systems corresponding to the strips by a direct method which uses a sparse indefinite solver on an symmetric augmented system obtained from the underdetermined system. We discuss this basic solution scheme in Section 2.

We accelerate our basic scheme using conjugate gradients and see that this is strongly influenced by the partitioning of the original system. We further improve the convergence of the conjugate gradient by preconditioning the matrix A . We multiply A by a matrix whose choice is also based on our partitioning strategy.

We consider the conjugate gradient acceleration and partitioning of the system in Section 3 and the preconditioning in Section 4. We illustrate the performance of our method on some test problems in Section 5 and present some concluding remarks in Section 6. Details of the supporting theory and a fuller description of the experiments can be found in Arioli, Duff, Noailles, and Ruiz (1989).

2 The block Cimmino method

Our method can be considered as a generalization of the method of Cimmino (Sloboda 1988), so we call it the block Cimmino method.

We obtain our blocks by partitioning the system (1.1) as

[†] Also IEI-CNR, via S. Maria 46, 56100 Pisa, Italy

^{*} Also CSS Division, HARWELL Laboratory, OXON OX110RA, England.

^{**} Also ENSEEIHT-IRIT, 2 rue Camichel, 31071 Toulouse Cedex, France.

$$\begin{pmatrix} A^1 \\ A^2 \\ \vdots \\ A^p \end{pmatrix} \mathbf{x} = \begin{pmatrix} \mathbf{b}^1 \\ \mathbf{b}^2 \\ \vdots \\ \mathbf{b}^p \end{pmatrix} \quad (2.1)$$

where $1 \leq p \leq n$.

If we define by $A^{i+} = (A^i A^{iT})^{-1} A^i$ the Moore-Penrose pseudo-inverse of A^i and by $P_{R(A^{iT})} = A^{iT} A^{i+}$ the projector onto the range of A^{iT} , we can describe the block Cimmino algorithm in the following way :

Algorithm 2.1 (Block Cimmino Method)

```

Choose  $\mathbf{x}^{(0)}$ 
repeat until convergence
  begin
    compute in parallel
       $i = 1, \dots, p \quad \delta^{i(k)} = A^{i+} \mathbf{b}^i - P_{R(A^{iT})} \mathbf{x}^{(k)}$ 
       $= A^{i+} (\mathbf{b}^i - A^i \mathbf{x}^{(k)})$ 
       $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega \sum_{i=1}^p \delta^{i(k)}$ 
    end
  
```

This gives rise to a general purpose iterative solver, well suited for both shared memory and distributed memory computers. If we take $p = n$, which means that each manifold from (2.1) is a hyperplane defined by one equation in (1.1), then algorithm 2.1 becomes the algorithm of Cimmino (Sloboda 1988).

When solving the subproblems (2.2), we have chosen to use the augmented system approach

$$\begin{bmatrix} \mathbf{G} & A^{iT} \\ A^i & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}^i \\ \mathbf{v}^i \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{b}^i - A^i \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{r}^i \end{bmatrix} \quad (2.3)$$

whose solution is

$$\begin{aligned} \mathbf{v}^i &= -(A^i \mathbf{G}^{-1} A^{iT})^{-1} \mathbf{r}^i \\ \mathbf{u}^i &= \mathbf{G}^{-1} A^{iT} (A^i \mathbf{G}^{-1} A^{iT})^{-1} \mathbf{r}^i . \end{aligned}$$

We use the augmented system approach because it is more stable than using the normal equations (Arioli, Duff, and de Rijk 1989), an important issue for the particular preconditioners we describe in Section 4. It is also trivial to include a scaling matrix in the augmented systems, as indicated by the matrix \mathbf{G} in (2.3). The solution to (2.2) would have $\mathbf{G} = \mathbf{I}$, but we consider other values for \mathbf{G} in Section 4 in connection with the preconditioning.

We solve the augmented systems with the sparse symmetric linear solver MA27 from the Harwell Subroutine Library (Duff and Reid 1983). The solver MA27 computes the \mathbf{LDL}^T decomposition of a permutation of the augmented matrix, using a mixture of 1×1 or 2×2 pivots chosen during the numerical factorization. The resulting \mathbf{L} and \mathbf{D} factors are then used to solve the augmented systems by forward and backward substitution in the usual way.

We emphasize that algorithm 2.1 is totally independent of the choice of the solvers for the

underdetermined subproblems (2.2). For instance, in some cases, it may be preferable to use a solver adapted to the structure of the block A^i . In special cases, one may even wish to use many different solvers, one for each different kind of block A^i . Note that, because we are using a block Cimmino scheme in (2.2), the systems are totally independent. We further enhance this independence through the partitionings discussed in the next section.

3 Conjugate gradient acceleration

From (2.2), the iteration matrix, Q , for our block Cimmino algorithm with $\omega=1$ is of the form

$$Q = I - H = I - \sum_{i=1}^p A^{iT} (A^i A^{iT})^{-1} A^i,$$

where H is symmetric and positive definite if the A^i all have full row rank. Thus, it is appropriate (Hageman and Young, 1981) to use the method of conjugate gradients to solve the iteration equations in the following manner :

Algorithm 5.1 (Conjugate gradient acceleration)

$x^{(0)}$ is arbitrary, $p^{(0)} = \delta^{(0)}$,

and $\delta^{(k)}$ is the pseudo-residual vector defined, for $k=0, 1, \dots$ by

$$\delta^{(k)} = \sum_{i=1}^p A^{iT} b^i + Hx^{(k)}$$

for $k=1, 2, \dots$, until convergence do:

$$x^{(k)} = x^{(k-1)} + \lambda_{k-1} p^{(k-1)}$$

$$p^{(k)} = \delta^{(k)} + \alpha_k p^{(k-1)}$$

$$\alpha_k = \frac{(\delta^{(k)}, \delta^{(k)})}{(\delta^{(k-1)}, \delta^{(k-1)})}$$

$$\lambda_{k-1} = \frac{(\delta^{(k-1)}, \delta^{(k-1)})}{(p^{(k-1)}, Hp^{(k-1)})}$$

An important aspect of using conjugate gradient acceleration is that we can show that the method is independent of the ω in algorithm 2.1 (Arioli, Duff, Noailles, and Ruiz 1989), which is why we set it equal to 1 above. Another aspect is the influence of the partitioning on the number of CG iterations, which we now discuss.

In a two-block partitioning, we partition the matrix A in two blocks

$$A = \begin{bmatrix} B^1 \\ B^2 \end{bmatrix},$$

where B^1 and B^2 have n_1 and n_2 rows respectively. We assume without loss of generality that $n_1 \geq n_2$.

With such a partitioning, the iteration matrix can be written in the form

$$\begin{pmatrix} 0_{n_1 \times n_1} & -Q^1 T Q^2 \\ -Q^2 T Q^1 & 0_{n_2 \times n_2} \end{pmatrix}, \tag{3.1}$$

all the columns of the matrix, because in that case the preconditioning would be equivalent to a scaling of the complete matrix, which would not modify at all the principal angles between the two spaces.

It is easy to incorporate the scaling G within the CG scheme in Section 3.

5 Numerical experiments

In this section, we present some numerical experiments for testing the behaviour of our Block Cimmino method. We first describe our different test problems and partitionings used, and then we analyse the results obtained on the eight processor Alliant FX/80 at CERFACS.

We will use four different test problems. The first two problems come from simulations of real flows developed at CERFACS by the fluid dynamics team and the remaining two are from the literature.

Problem 1, from a model developed by Françoise Perrel, concerns the study of a body entering the atmosphere at a high mach number. It is based on a finite-volume discretisation of the Navier Stokes equations coupled with chemistry. This leads to block tridiagonal matrices of order 20700 where the blocks are of order 345. The matrix is unsymmetric and non-diagonally dominant and, in this case, we do not require the solution to be very accurate.

Problem 2, from a model developed by Per Weinerfelt, is from the study of a two-dimensional wing profile at transonic flow (with no chemistry effects). It is still based on finite-volume discretisation, but this time of the Euler equations. This leads to unsymmetric but diagonally dominant block tridiagonal matrices of order 7680 with each block of order 240. This time, the test problem we are solving is close to the steady-state solution, and a very accurate solution is required.

The two other test problems come from the paper by Kamath and Sameh (1988). These are obtained by application of the finite-difference method to a two-dimensional partial differential equation. The right-hand side is computed using a given solution.

Problem 3

$$-u_{xx} - [(1+xy)u_y]_y - 10000 [\cos(x)u_x + (e^{-x} + x)u_y] + 3u = g, \quad u = x+y,$$

on a 32 by 32 grid.

Problem 4

$$-u_{xx} - u_{yy} + 1000 e^{xy} u_x - 1000 e^{xy} u_y = g, \quad u = x+y.$$

on a 64 by 64 grid.

Since we are performing our tests on an 8 processor Alliant, we consider the three following partitionings, which are a subset of those considered by Arioli, Duff, Noailles, and Ruiz (1989).

The first partitioning has only a single block and thus corresponds to the direct solution of the augmented system on the overall initial matrix.

In the second partitioning we choose the smallest possible size for the blocks in the interface. This should decrease the number of iterations at the possible cost of a reduction in parallelism.

Finally, the third partitioning is the one used by Kamath and Sameh (1988) and is defined by taking equal-sized blocks of the smallest possible size (two times the number of rows in one block of the block tridiagonal structure). This partitioning gives good load balancing, but it has a large interface block. It has the added advantage, however, that it minimises the overall fill-in in the factorization of the different augmented systems because they are all of the smallest size. All the tests are performed on an Alliant FX/80 at CERFACS in double precision (64 bit words) with machine precision $2.2 \cdot 10^{-16}$. For the convergence criterion, we use the scaled residual defined by

$$\frac{\|A x^{(k)} - b\|_{\infty}}{\|A\|_{\infty} \|x^{(k)}\|_{\infty} + \|b\|_{\infty}}$$

at step k , and require, for termination, this value to be less than or equal to 10^{-14} for problems 2 and 4, and less than 10^{-7} for problems 1 and 3.

| Part. | No. iter. | Storage for factors (in thousands) | | Elapsed time (seconds) | | | speedup |
|------------|-----------|--|-------------------|------------------------|---------------|------------|---------|
| | | Integer | Real | Analysis | Factorization | Iterations | |
| Problem 1. | | Order = 20700 | Nonzeros = 511050 | | | | |
| 1 | | Too much storage for memory of Alliant | | | | | |
| 2 | 26 | 400 | 6328 | 8.9 | 231.9 | 76.7 | 3.0 |
| 3 | 46 | 676 | 1067 | 7.2 | 12.8 | 48.5 | 6.0 |
| Problem 2. | | Order = 7680 | Nonzeros = 113760 | | | | |
| 1 | 1 | 123 | 4328 | 14.7 | 773.6 | 10.7 | 1.0 |
| 2 | 19 | 142 | 907 | 2.20 | 11.88 | 9.82 | 4.6 |
| 3 | 21 | 172 | 375 | 1.97 | 4.10 | 7.32 | 6.0 |
| Problem 3. | | Order = 1024 | Nonzeros = 4992 | | | | |
| 1 | 1 | 15 | 119 | 2.02 | 5.78 | 0.42 | 1.0 |
| 2 | 197 | 11 | 23 | 0.28 | 0.31 | 6.43 | 5.7 |
| 3 | 332 | 9 | 18 | 0.20 | 0.29 | 11.50 | 6.3 |
| Problem 4. | | Order = 4096 | Nonzeros = 20224 | | | | |
| 1 | 1 | 65 | 697 | 8.30 | 49.24 | 2.29 | 1.0 |
| 2 | 214 | 55 | 207 | 1.49 | 2.48 | 35.72 | 4.8 |
| 3 | 464 | 60 | 56 | 0.77 | 0.95 | 64.25 | 6.7 |

Table 5.1. Behaviour of the Block Cimmino method on an Alliant FX/80 (8 processors).

The results in Table 5.1 illustrate the power of our method and the effect of the partitioning. In some cases, we converge very quickly to the desired solution. The results support the comments made when we introduced the partitionings. The amount of time spent in factorizations decreases from partitioning 1 to 2 and 3 while the number of iterations increase. Sometimes the reduction in storage for the factors when we work with blocks means that the iterations take less time although there are more of them, notable in problem 1. In general, partitioning 3 takes the least elapsed time although for the small pde problems, a direct solution or partitioning 2 is competitive. Good speedups are obtained, particularly with partitioning 3.

For problem 3, we can examine the use of the scaling of Section 4. We show in Table 5.2 the variation in number of iterations and elapsed time as α is changed. Partitioning 2 was used and, in all cases, the iteration was terminated when the scaled residual had value less than 10^{-14} . As we remarked in Section 4, increasing α generates more ill-conditioning in the subproblems. This is reflected a little in the increasing contribution of the factorization to the overall time and a doubling of storage for the factors. The number of iterations drop dramatically and the overall effect is to reduce the elapsed time substantially as can be seen in Table 5.2.

| α^2 (in powers of 10) | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 14 |
|---------------------------------|--------------------|------|------|------|-----|-----|-----|-----|
| Elapsed time | 17.0 | 16.4 | 14.7 | 12.0 | 8.2 | 4.6 | 1.5 | 1.1 |
| No. iterations | 500 ^(*) | 366 | 297 | 233 | 152 | 73 | 10 | 1 |

Table 5.2. Effect of scaling on solution of problem 3 using partitioning 2. (*) For $\alpha^2=1$ the algorithm did not reach the required accuracy and was stopped after 500 iterations with a scaled residual of $0.87 \cdot 10^{-13}$.

6 Conclusion

In this paper, we have introduced a method for the solution of large sparse general linear systems. There are many different components to this method, most of which we have explored here, but some of which warrant further study. The main components are: the partitioning, the choice of method for solving the subproblems, the iterative scheme used, and the scaling employed. For the solution of the subproblems, a new version of MA27 that recognizes and exploits the zeros on the diagonal and behaves much better in terms of fill-in is being developed now at Harwell (Duff, Gould, Reid, Scott, and Turner 1989) and should improve all the results.

We have shown that we can, for block tridiagonal matrices, partition the system in order to reduce the number of conjugate gradient iterations, while maintaining a good degree of parallelism. We have also illustrated the power of preconditioning the method.

References

- Arioli, M., Duff, I. S., Noailles, J., and Ruiz, D. (1989). A block projection method for general sparse matrices. Report TR 89/11, CERFACS, Toulouse.
- Arioli, M., Duff, I. S., and de Rijk, P. P. M. (1989). On the augmented system approach to sparse least-squares problems. *Numerische Math.* **55**, pp.667-684.
- Duff, I. S. and Reid, J. K. (1983). The multifrontal solution of indefinite sparse linear systems. *ACM Trans. Math. Softw.* **9**, pp. 302-325.
- Duff, I. S., Gould, N. I. M., Reid, J. K., Scott, J. A., and Turner, K. (1989). Factorization of sparse symmetric indefinite matrices. Report CSS 236, Computer Science and Systems Division, Harwell Laboratory. Submitted to *IMA J. Numer. Anal.*
- Hageman, L. A. and Young, D. M. (1981). *Applied iterative methods*. Academic Press, New York and London.
- Kamath, C. and Sameh, A. (1988). A projection method for solving nonsymmetric linear systems on multiprocessors. *Parallel Computing* **9**, pp. 291-312.
- Sloboda, F. (1988). A projection method of the Cimmino type for linear algebraic systems, Technical Report n. 16, Dipartimento di Matematica, University of Bergamo, Italy.