SIAM J. SCI. STAT. COMPUT.
Vol. 3, No. 3, September 1982

0196-5204/82/0303-0005 $01.00/0

A VECTORIZABLE VARIANT OF SOME ICCG METHODS*

HENK A. VAN DER VORST†

Abstract. The preconditioned conjugate gradient method can be a useful tool in solving certain very large sparse linear systems. If this is done on a vector machine like the CRAY-1, then it appears that some of the most effective preconditionings are difficult to vectorize. In this paper it is shown how a class of preconditionings can be modified in such a way that they become highly vectorizable while still easy to program. Numerical experiments that show the improvement in performance of the preconditioned conjugate gradient algorithm have been included.

Key words. preconditioned conjugate gradients, ICCG methods, incomplete Choleski, vector/parallel computers

0. Introduction. The ICCG methods arise when the conjugate gradient algorithm (CG) is applied to the preconditioned system $K^{-1}Ax = K^{-1}b$ in order to solve the linear system $Ax = b$, where K is an incomplete Choleski (IC) decomposition of the n by n symmetric positive real matrix A [6], [9], [10]. Since most of the CG algorithm is vectorizable (assuming that fast code is available for inner products) and the matrix vector product Ax is also vectorizable, the main difficulty arises in the computation of $K^{-1}y$ for a given vector y on a vector processor.

This is due to the fact that in the IC decomposition we have $K = LL^T$, where L is lower triangular, and computation of $L^{-1}z$ as well as $L^{-T}z$ requires in many relevant cases recurrence relations in the components of z and these are not vectorizable directly. This has been recognized by many authors who propose techniques to overcome this problem. Significant improvement in efficiency can be made if one uses cyclic reduction techniques [3], [5], [7], [11]. Using these techniques rather fast code can be written for vector/parallel processors such as the CRAY-1, but this requires a permutation of the unknowns and has often to be done in assembler code in order to achieve optimal performance. Dubois, Greenbaum and Rodrigue [1] propose to approximate the inverse of A by a truncated Neumann expansion and to use this as a preconditioning. Although their preconditioning is completely vectorizable, they report a significant increase in the number of *cg* iterations as compared to the ICCG methods. Johnson and Paul [4] claim that the number of iterations can be reduced to about the same level as for the ICCG methods, by introducing parameters that depend on extreme eigenvalues of the matrix A.

In this paper we return to the incomplete Choleski decomposition and describe a useful variant that can be vectorized completely very easily. The variant arises if one applies similar techniques as proposed in [1] in an intermediate step of the preconditioning process. We will show that for this variant the number of iterations will be about the same as for the standard ICCG algorithms. No information is required about eigenvalues of A and neither are other parameters introduced. The resulting preconditioned CG method has been tested on a CRAY-1 computer and numerical examples indicating the efficiency of the new variants have been included.

---

* Received by the editors September 9, 1981, and in revised form December 17, 1981. The research reported in this paper was supported in part by the European Research Office, London, through grant DAJA 37-80-C-0243.

† Academisch Computer Centrum Utrecht, Budapestlaan 6, Utrecht, the Netherlands.

1. The vectorizable ICCG algorithm. The ICCG algorithm is defined by

$$\alpha_i = \frac{(r_i, K^{-1}r_i)}{(p_i, Ap_i)}, \quad x_{i+1} = x_i + \alpha_i p_i, \quad r_{i+1} = r_i - \alpha_i A p_i,$$

$$\beta_i = \frac{(r_{i+1}, K^{-1}r_{i+1})}{(r_i, K^{-1}r_i)}, \qquad p_{i+1} = K^{-1}r_{i+1} + \beta_i p_i. \tag{1.1}$$

The preconditioning matrix K is an incomplete Choleski factorization of the matrix A and will be written as $K = (L+D)D^{-1}(D+L)^T$. L is a strict lower triangular matrix and D is a diagonal matrix.

We will now assume that A has been scaled in such a way that $D = I$, which simplifies most of the coming formulas and makes actual computation more efficient. In most situations the computation of Ap_i is completely vectorizable and we therefore focus attention to the computation of $K^{-1}r_i$, which causes the bottleneck on vector computers since it requires back substitution. In our further analysis we will restrict ourselves to matrices A that come from 5-point finite difference approximations of elliptic p.d.e.'s over rectangular regions. From the presentation it will be clear for which other matrices the ideas presented in this paper can be applied.

In our case the matrix A has the block structure shown in Fig. 1.

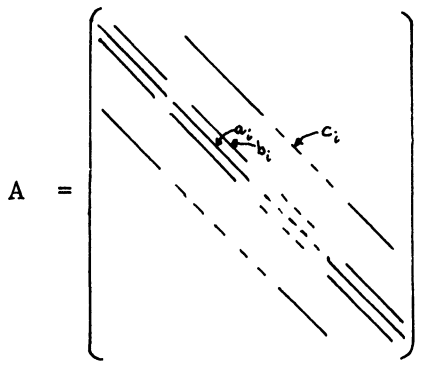

FIG. 1

We now consider for K the ICCG(1, 1) variant [10], which means that the factor L of K has the same sparsity structure as the corresponding part of A. It follows that the elements of L are identical to those of A.

We write the decomposition as

$$A = K + R = (-F - E + I)(I - E - F)^T + R,$$

where E^T is the matrix consisting of only the upper diagonal elements b_i and F^T contains the upper diagonal elements c_i. The matrix R represents the approximation error of K with respect to A.

A typical step in the determination of $K^{-1}r_i$ is the solution of z from

$$(I - E - F)z = y$$

or

$$(I - E_j)z_j = y_j + F_j z_{j-1},$$

where the index j refers to the jth block of the matrix and where z and y have been partitioned accordingly.

Since the elements b_i of E_j in general are small compared to 1.0, the idea is now to compute z_j from

$$z_j = (I - E_j)^{-1}(y_j + F_j z_{j-1}) = (I + E_j + E_j^2 + E_j^3 + \cdots)(y_j + F_j z_{j-1}),$$

and to truncate the power series after some term, the mth term say. We observe that the approximate solution

$$(I + E_j + \cdots + E_j^m)(y_j + F_j z_{j-1}) \tag{1.2}$$

has now been written in fully vectorizable form. Our main concern is to investigate whether a value of m exists such that the computation of (1.2) will not be too expensive and also that this truncation will not lead to a much higher number of iterations in the resulting ICCG algorithm.

2. The effect of truncation on the preconditioning matrix. We now try to determine where to truncate the power series for $(I-E)^{-1}$ in such a way that the error due to this truncation is small compared to the approximation error $R = A - K$. We use the relation

$$(I + E + \cdots + E^m)^{-1} = (I - E)(I - E^{m+1})^{-1}, \tag{2.1}$$

and recall that

$$A = (I - E - F)(I - E - F)^T + R = K + R. \tag{2.2}$$

If we approximate the factor $(I-E)^{-1}$ in the back substitution by $I + E + E^2 + \cdots + E^m$, then we have effectively

$$A = \tilde{K} + S + R, \tag{2.3}$$

where $\tilde{K}$ is the matrix which describes the truncated back substitution and S is the error matrix due to the truncation.

It follows from (2.1) that

$$\tilde{K} = ((I - E)(I - E^{m+1})^{-1} - F)((I - E)(I - E^{m+1})^{-1} - F)^T. \tag{2.4}$$

The matrix S_1 is defined by

$$\begin{aligned}(I - E)(I - E^{m+1})^{-1} &= (I - E)(I + E^{m+1} + E^{2(m+1)} + \cdots) \\ &= I - E + (I - E)E^{m+1}(I - E^{m+1})^{-1} = I - E + S_1.\end{aligned} \tag{2.5}$$

If (2.5) is inserted in (2.4), then it follows from (2.3) that

$$S = -S_1(I - E - F)^T - (I - E - F)S_1^T - S_1 S_1^T. \tag{2.6}$$

The elements of E are just the b_i and those of F are given by c_i. Therefore we can derive bounds on S in the following way:

$$\|I - E - F\|_\infty \leqq 1 + b + c, \tag{2.7}$$

where $b = \max b_i$, $c = \max c_i$. Thus

$$\|S_1\|_\infty \leqq (1 + b)b^{m+1}(1 - b^{m+1})^{-1} \simeq b^{m+1}(1 + b). \tag{2.8}$$

If we neglect also the higher order term $S_1 S_1^T$, then from (2.6), (2.7) and (2.8) we have

$$\|S\|_\infty \leqq 2(1 + b)(1 + b + c)b^{m+1}. \tag{2.9}$$

The matrix R has the structure in Fig. 2, where $r_i = b_i c_{i-1}$. Thus

$$\|R\|_\infty = 2 \max |b_i c_{i-1}|. \tag{2.10}$$

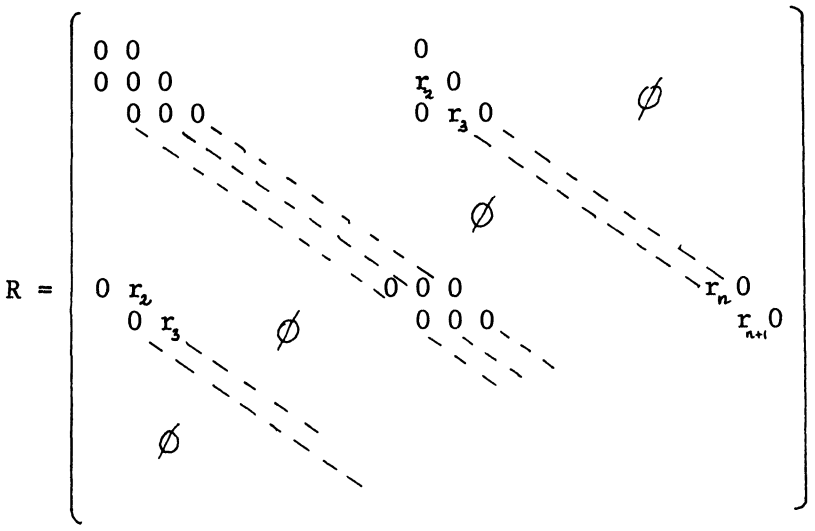

FIG. 2

If we choose as a model problem the set of equations that arises from 5-point finite difference approximation to the Poisson equation, with Dirichlet boundary conditions, then it follows that the elements of the matrices E and F are approximately $b_i \simeq 0.3$ and $c_i \simeq 0.3$. In this case we have $\|R\|_\infty \simeq 0.18$ and for $\|S\|_\infty$ we find the upper bounds in Table 1, depending on the level of truncation.

TABLE 1
Upper bounds for $\|S\|_\infty$.

m	1	2	3
$\|S\|_\infty$	0.4	0.11	0.034

We see from this table that for $m = 2$ the truncation error S is comparable to the approximation error R, while for $m = 3$ the approximation error R dominates.

Since $(I+E+E^2+E^3)y$ can be computed as $(I+E^2)(I+E)y$ which can be computed almost at the cost of $(I+E+E^2)y$, we prefer truncation after 3 terms over truncation after 2 terms.

3. Numerical results. In this section we give the numbers of iterations for both the "standard" ICCG(1, 1) algorithm and the truncated ICCG(1, 1) algorithm, as well as the CPU times required for both algorithms on a CRAY-1 computer. The truncation idea can also be applied similarly to the more complicated ICCG algorithms like ICCG(1, 2) and ICCG(1, 3) [10]. In the latter variant 2 extra nonzero diagonals are allowed in the factors of K, situated near the F-diagonals. Since the approximation error matrix R for the ICCG(1, 3) decomposition is in general much smaller than for the ICCG(1, 1) case, it may be expected that in this case the number of iterations will be increased when we apply truncation. The results for this decomposition have also been included. All the algorithms have been coded in standard Fortran except for the

inner products which were available in assembler code. In the truncated ICCG algorithms the vector $(I-E)^{-1}y$ has been approximated by $(I+E^2)(I+E)y$, where E^2 was computed once and stored in memory. Note that E^2 is a matrix that has only one nonzero diagonal. All the matrices occurring in the algorithms have been represented in diagonal form, this seems to be the most effective way on a vector/parallel processor [8]. It should be stressed here that the CPU times listed for the standard ICCG algorithm may give the reader a too optimistic impression with respect to their behavior on vector machines. Their CPU times are relatively low due to the fact that A has only 5 nonzero diagonals in our examples, the decompositions have been scaled such that their main diagonal elements are equal to 1.0, Eisenstat's efficient implementation has been applied [2], and they have been vectorized also as far as possible (Eisenstat's ideas have not been applied to the new variants). Further on the size of the matrices is comparatively small so that any initial scalar arithmetic influences the results.

The computer time required for the construction of the incomplete decompositions as well as for the scaling of the matrix has been included in the listed CPU times.

In order to be better able to judge the results for the vectorizable variants we have also included the results for the vectorizable truncated Neumann expansion preconditioning as proposed in [1]. The value of p in this case refers to the number of terms after which the Neumann expansion has been truncated.

Although more complicated problems, e.g., 9-point finite difference operators or finite element problems, would give a higher improvement ratio for the vectorizable algorithms, we have chosen simple examples in order to facilitate for the reader the reconstruction of these problems for testing purposes.

In both the examples 20 different starting vectors, with entries chosen at random in (0, 1), have been taken and the average results over these 20 iteration processes have been listed. The iteration was terminated as soon as the residual vector in 2-norm was less than 10^{-10}. For the 5-point finite difference discretisation of the p.d.d.'s see [13].

Example 1. $-(Au'_x)'_x-(Au'_y)'_y=B$. This equation was solved over the unit square (see Fig. 3), where A and B are given by

region 1: $A=1$, $B=0$,

region 2: $A=100$, $B=100$.

On the boundary defined by $y=0$ we have $u=0$, along the other three boundaries we have the Neumann condition $\partial u/\partial n=0$. A uniform mesh was chosen with mesh spacing 1/32, resulting in a linear system with 1056 unknowns. Table 2 shows the iteration results.

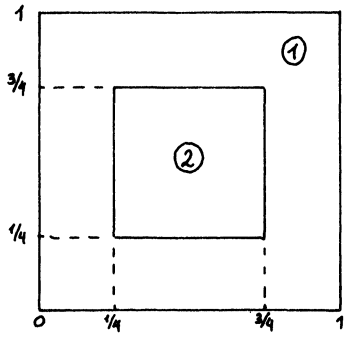

FIG. 3

TABLE 2
Iteration results for Example 1.

algorithm	number of iterations	CPU time on CRAY-1
standard ICCG(1, 1)	62	0.125
truncated ICCG(1, 1)	62	0.078
standard ICCG(1, 3)	31	0.086
truncated ICCG(1, 3)	33	0.051
truncated Neumann, $p=2$	103	0.079
truncated Neumann, $p=4$	73	0.087

Example 2. $-(Au'_x)'_x-(Au'_y)'_y+Bu=B$. (See Fig. 4.) The coefficients in this equation are defined as follows:

region 1: $A=3.0$, $B=0.05$,
region 2: $A=2.0$, $B=0.03$,
region 3: $A=1.0$, $B=0.02$.

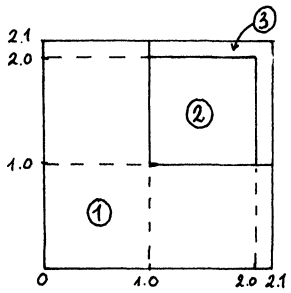

FIG. 4

Along the boundaries the Neumann condition $\partial u/\partial n=0$ is imposed. The grid spacing of the uniform mesh was chosen to be $1/42$; thus the resulting linear system is of the order $N=1849$. See Table 3.

TABLE 3
Iteration results for Example 2.

algorithm	number of iterations	CPU-time on CRAY-1
standard ICCG(1, 1)	87	0.304
truncated ICCG(1, 1)	89	0.181
standard ICCG(1, 3)	44	0.206
truncated ICCG(1, 3)	47	0.116
truncated Neumann, $p=2$	149	0.194
truncated Neumann, $p=4$	105	0.215

4. Conclusions. We see that the vectorizable ICCG variants give a considerable improvement in CPU time on the CRAY-1. The variants are also more efficient than the truncated Neumann expansion preconditioning as suggested in [1]. It seems that they are very near to the equivalent incomplete Choleski preconditioning the existence

of which has been questioned by Dubois, Greenbaum and Rodrigue [1, see Conclusions]. Since the programming effort is only slightly increased as compared to the standard ICCG algorithms, we believe that it is advisable to prefer the truncated algorithms for computers like the CRAY-1. From numerical experiments it follows that truncation after 2 terms increases the number of iterations by a small percentage. Since the computational effort to compute $(I+E+E^2)y$ is almost the same as the effort for $(I+E+E^2+E^3)y$, truncation after 3 terms pays off most. This truncation idea can be easily applied on more complicated ICCG algorithms and their nonsymmetric equivalents [12]. As soon as there are more nonzero diagonals near the main diagonal then the computer time required for evaluation of the truncated power series (applied on a given vector) increases accordingly, making this idea possibly less attractive in those cases. It is an open question whether and to what extent the introduction of parameters in the truncated power series similar to those as described by Johnson and Paul [4], could improve the performance of the truncated ICCG methods.

Acknowledgments. Discussions with Thomas Jordan (Los Alamos), Gene Golub and Rob Schreiber (both at Stanford) proved to be quite helpful and stimulating. Both referees draw my attention to several useful references for which I am very much indebted. They also contributed to the presentation of this paper.

REFERENCES

[1] P. F. DUBOIS, A. GREENBAUM AND G. H. RODRIGUE, *Approximating the inverse of a matrix for use in iterative algorithms on vector processors*, Computing, 22 (1979), pp. 257–268.

[2] S. C. EISENSTAT, *Efficient implementation of a class of preconditioned conjugate gradient methods*, this Journal, 2 (1981), pp. 1–4.

[3] A. GREENBAUM AND G. RODRIGUE, *The incomplete Choleski conjugate gradient method for the STAR (5-point operator)*, Res. Rep. UCID-17574, Lawrence Livermore Lab., Livermore, CA, 1977.

[4] O. G. JOHNSON AND G. PAUL, *Optimal parameterized incomplete inverse preconditioning for conjugate gradient calculations*, Res. Rep. RC8644, IBM-Research Center, Yorktown Heights, NY, 1981.

[5] T. L. JORDAN, *A guide to parallel computation and some CRAY-1 experiences*, LANL Report LA-UR-81-247, Los Alamos National Laboratory, Los Alamos, NM, 1981.

[6] D. S. KERSHAW, *The ICCG method for the iterative solution of systems of linear equations*, J. Comp. Phys., 26 (1978), pp. 43–65.

[7] ———, *The solution of single linear tridiagonal systems and vectorization of the ICCG algorithm on the CRAY-1*, Res. Rep., UCID-19085, Lawrence Livermore Lab., Livermore, CA, 1981.

[8] N. K. MADSEN, G. H. RODRIGUE AND J. I. KARUSH, *Matrix multiplication by diagonals on a vector/parallel processor*, Inform. Proc. Letters, 5 (1976), pp. 41–45.

[9] J. A. MEYERINK 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.

[10] ———, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems*, J. Comp. Phys., 44 (1981), pp. 134–155.

[11] G. RODRIGUE AND D. WOLITZER, *Incomplete block cyclic reduction*, 2nd IMACS International Symposium on Parallel Computation, Newark, Delaware, 1981.

[12] H. A. VAN DER VORST, *Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems*, J. Comp. Phys., 44 (1981), pp. 1–19.

[13] R. VARGA, *Matrix Iterative Analysis*, Prentice Hall, Englewood Cliffs, NJ, 1962.