RAL-TR-1998-052

Preconditioning and Parallel Preconditioning[1]

Iain S. Duff[2] and Henk A. van der Vorst[3]

ABSTRACT

We review current methods for preconditioning systems of equations for their solution using iterative methods. We consider the solution of unsymmetric as well as symmetric systems and discuss techniques and implementations that exploit parallelism.

We particularly study preconditioning techniques based on incomplete LU factorization, sparse approximate inverses, polynomial preconditioning, and block and element by element preconditioning. In the parallel implementation, we consider the effect of reordering.

Keywords: preconditioning, parallel computers, sparse matrices, incomplete factorization, sparse approximate inverses, block methods, element by element preconditioning.

AMS(MOS) subject classifications: 65F05, 65F50.

---

[1]This paper is a preprint of a Chapter of the book "Numerical Linear Algebra for High-Performance Computers" by Dongarra, Duff, Sorensen, and van der Vorst that will be published by SIAM Press.

[2]I.S.Duff@rl.ac.uk

[3]vorst@math.uu.nl, Mathematical Institute, University of Utrecht, The Netherlands.

Current reports available by anonymous ftp from matisa.cc.rl.ac.uk in the directory "pub/reports". This report is in file duvoRAL98052.ps.gz. Also published as Technical Report TR/PA/98/23 from CERFACS.

Department for Computation and Information
Atlas Centre
Rutherford Appleton Laboratory
Oxon OX11 0QX

July 28, 1998.

Contents

1 The Purpose of Preconditioning

There are many occasions and applications where iterative methods fail to converge or converge very slowly. In this paper, we consider methods of preconditioning systems so that their subsequent solution by iterative methods is made more computationally feasible.

The general problem of finding a preconditioner for a linear system $Ax = b$ is to find a matrix K (the *preconditioner* or *preconditioning matrix*) with the properties that

1. K is a good approximation to A in some sense.
2. The cost of the construction of K is not prohibitive.
3. The system $Kx = b$ is much easier to solve than the original system.

The idea is that the matrix $K^{-1}A$ may have better properties in the sense that well chosen iterative methods converge much faster. In this case, we solve the system $K^{-1}Ax = K^{-1}b$ instead of the given system $Ax = b$. Krylov subspace methods need the operator of the linear system only for computing matrix vector products. This means that we can avoid forming $K^{-1}A$ explicitly. Instead, we compute $u = K^{-1}Av$ by first computing $w = Av$ and then obtain u by solving $Ku = w$. Note that, when solving the preconditioned system using a Krylov subspace method, we will get quite different subspaces than for the original system. The aim is that approximations in this new sequence of subspaces will approach the solution more quickly than in the original subspaces.

There are different ways of implementing preconditioning; for the same preconditioner these different implementations lead to the same eigenvalues for the preconditioned matrices. However, the convergence behavior is also dependent on the eigenvectors or, more specifically, on the components of the starting residual in eigenvector directions. Since the different implementations can have quite different eigenvectors, we might thus believe that their convergence behavior might be quite different. Three different implementations are:

1. **Left-preconditioning**: apply the iterative method to $K^{-1}Ax = K^{-1}b$. We note that symmetry of A and K does not imply symmetry of $K^{-1}A$. However, if K is symmetric positive definite then $[x, y] \equiv (x, Ky)$ defines a proper inner product. It is easy to verify that $K^{-1}A$ is symmetric with respect to the new inner product $[\ ,\ ]$, so that we can use methods like MINRES, SYMMLQ, and CG (when A is positive definite as well) in this case. Popular formulations of preconditioned CG are based on this observation.

   If we are using a minimal residual method (GMRES or MINRES), we should note that with left-preconditioning we are minimizing the preconditioned residual $K^{-1}(b - Ax_k)$, which may be quite different from the residual $b - Ax_k$.

This could have consequences for stopping criteria that are based on the norm of the residual.

2. **Right-preconditioning**: apply the iterative method to $AK^{-1}y = b$, with $x = K^{-1}y$. This form of preconditioning also does not lead to a symmetric product when A and K are symmetric. With right-preconditioning we have to be careful with stopping criteria that are based upon the error: $||y - y_k||_2$ may be much smaller than the error-norm $||x - x_k||_2$ (equal to $||K^{-1}(y - y_k)||_2$) that we are interested in. Right-preconditioning has the advantage that it only affects the operator and not the right-hand side. This may be useful in the design of software.

3. **Two-sided preconditioning**: For a preconditioner K with $K = K_1K_2$, the iterative method can be applied to $K_1^{-1}AK_2^{-1}z = K_1^{-1}b$, with $x = K_2^{-1}z$. This form of preconditioning may be used for preconditioners that come in factored form. It can be seen as a compromise between left- and right-preconditioning. This form may be useful for obtaining a (near) symmetric operator for situations where K cannot be used for the definition of an inner product (as described under left-preconditioning).

The choice of K varies from purely "black box" algebraic techniques which can be applied to general matrices to "problem dependent" preconditioners which exploit special features of a particular problem class. Although problem dependent preconditioners can be very powerful, there is still a practical need for efficient preconditioning techniques for large classes of problems. One should realize that working with a preconditioner adds to the computational complexity of an iterative method, and the use of a preconditioner only pays if there is a sufficient reduction in the number of iterations. We will discuss this aspect in some more detail.

First, we consider iterative methods with a fixed amount of computational overhead per iteration step, independent of the iteration number: CG, MINRES, Bi-CG, CGS, QMR, etc. We denote by t_A the computing time for the matrix-vector product with A, and the computational overhead per iteration step (for inner products, vector updates, etc) by t_O. The computing time for k iteration steps is then given by

$$T_U = k(t_A + t_O).$$

For the preconditioned process we make the following assumptions with respect to computing time:

(a) the action of the preconditioner, for instance the computation of $K^{-1}w$ takes αt_A.

(b) the construction costs for the preconditioner are given by t_C.

(c) preconditioning reduces the number of iterations by a factor f ($f > 1$).

The computing time for the preconditioned process, to obtain an approximation comparable to the unpreconditioned process, can be expressed as:

$$T_P = \frac{k}{f}\left((\alpha+1)t_A + t_O\right) + t_C.$$

The goal of preconditioning is that $T_P < T_U$, which is the case if

$$f > \frac{(\alpha+1)t_A + t_O}{t_A + t_O - \frac{t_C}{k}}.$$

We see that the construction of expensive preconditioners is pointless if the number of iterations k for the unpreconditioned process is low. It is thus realistic to consider only cases where k is so large that the initial costs t_C play no role: $t_A + t_O \gg t_C/k$. Furthermore, in many cases the matrix-vector products are the most expensive part of the computation: $t_A \geq t_O$, so that we only profit from preconditioning if the reduction f in the number of iteration steps is significantly bigger than $\alpha + 1$. In view of the fact that many popular preconditioners are difficult to parallelize, this requirement is certainly not trivially fulfilled in many situations.

For methods such as GMRES and FOM, the situation is slightly more complicated because of the fact that the overhead costs increase quadratically with the number of iterations. Let us assume that we use GMRES(m), and that we characterize the computational time as:

t_A for the matrix-vector product with A,

t_O for the costs of one inner product plus one SAXPY.

Then the computing time for k cycles of unpreconditioned GMRES(m) is given roughly by

$$T_U = k(m\, t_A + \frac{1}{2}m^2 t_O).$$

Again we assume that km is so large that the time for constructing a preconditioner can be ignored, and that with preconditioning we need f times fewer iterations. The computing time per action of the preconditioner is again given by αt_A. Then the computing time for preconditioned GMRES(m) is given by

$$T_P = \frac{k}{f}(m(\alpha+1)t_A + \frac{1}{2}m^2 t_O),$$

and, after some manipulation, we find that preconditioning only helps to reduce the computational time if

$$f > \frac{(\alpha+1)t_A + \frac{1}{2}m t_O}{t_A + \frac{1}{2}m t_O}.$$

We see that, if m is small and if t_A dominates, then the reduction f has to be (much) bigger than $\alpha + 1$, in order to make preconditioning practical. If m is so large that

$\frac{1}{2}mt_O$ dominates over t_A, then obviously a much smaller f may be sufficient to amortize the additional costs for the preconditioner.

We now say something about the effects of preconditioning. There is very little theory for what one can expect *a priori* with a specific type of preconditioner. It is well known that incomplete LU decompositions exist if the matrix A is an M-matrix, but that does not say anything about the potential reduction in the number of iterations. For the discretized Poisson equation, it has been proved (Sleijpen and van der Vorst 1995) that the number of iterations will be reduced by a factor larger than 3.

For systems that are not positive definite, almost anything can happen. For instance, let us consider a symmetric matrix A that is indefinite. The goal of preconditioning is to approximate A by K, and a common strategy is to ensure that the preconditioned matrix $K^{-1}A$ has its eigenvalues clustered near 1 as much as possible. Now imagine some preconditioning process in which we can improve the preconditioner continuously from $K = I$ to $K = A$. For instance, one might think of incomplete LU factorization with a drop-tolerance criterion. For $K = I$, the eigenvalues of the preconditioned matrix are clearly those of A and thus are at both sides of the origin. Since eventually when the preconditioner is equal to A all eigenvalues are exactly 1, the eigenvalues have to move gradually in the direction of 1, as the preconditioner is improved. The negative eigenvalues, on their way towards 1 have to pass the origin, which means that while improving the preconditioner the preconditioned matrix may from time to time have eigenvalues very close to the origin. In our chapter on iterative methods, we explained that the residual in the i-th iteration step can be expressed as

$$r_i = P_i(B)r_0,$$

where B represents the preconditioned matrix. Since the polynomial P_i has to satisfy $P_i(0) = 1$, and since the values of P_i should be small on the eigenvalues of B, this may help to explain why there may not be much reduction for components in eigenvector directions corresponding to eigenvalues close to zero, if i is still small. This means that, when we improve the preconditioner, in the sense that the eigenvalues are getting more clustered towards 1, its effect on the iterative method may be dramatically worse for some "improvements". This is a qualitative explanation of what we have observed many times in practice. By increasing the number of fill-in entries in ILU, sometimes the number of iterations increases. In short, the number of iterations may be a very irregular function of the level of the incomplete preconditioner. For other types of preconditioners similar observations may be made.

There are only very few specialized cases where it is known *a priori* how to construct a good preconditioner and there are few proofs of convergence except in very idealized cases. For a general system, however, the following approach may help to build up one's insight into what is happening. For a representative linear system,

one starts with unpreconditioned GMRES(m), with m as high as possible. In one cycle of GMRES(m), the method explicitly constructs an upper Hessenberg matrix of order m, denoted by H_m. This matrix is reduced to upper triangular form but, before this takes place, one should compute the eigenvalues of H_m, called the Ritz values. These Ritz values usually give a fairly good impression of the most relevant parts of the spectrum of A. Then one does the same with the preconditioned system and inspects the effect on the spectrum. If there is no specific trend of improvement in the behavior of the Ritz values, when we try to improve the preconditioner, then obviously we have to look for another class of preconditioner. If there is a positive effect on the Ritz values, then this may give us some insight as to how much more the preconditioner has to be further improved in order to be effective. At all times, we have to keep in mind the rough analysis that we made in this chapter, and check whether the construction of the preconditioner and its costs per iteration are still inexpensive enough to be amortized by an appropriate reduction in the number of iterations.

2 Incomplete LU Decompositions

Iterative methods converge very fast if the matrix A is close to the identity matrix in some sense, and the main goal of preconditioning is to obtain a matrix $K^{-1}A$ which is close to I. The phrase "in some sense" may have different meanings for different iterative methods: for the standard unpreconditioned Richardson method we want $||I - A||_2$ to be (much) smaller than 1; for many Krylov subspace methods it is desirable that the condition number of $K^{-1}A$ is (much) smaller than that of A, or that the eigenvalues of $K^{-1}A$ are strongly clustered around some point (usually 1). In all these situations the preconditioning operator K approximates A.

It is quite natural to start looking at a direct solution method for $Ax = b$, and to see what variations we can make if the direct approach becomes too expensive. The most common direct technique is to factorize A as $A = LU$, if necessary with permutations for pivoting. One of the main problems with the LU factorization of a sparse matrix is that often the number of entries in the factors is substantially greater than in the original matrix so that, even if the original matrix can be stored, the factors can not.

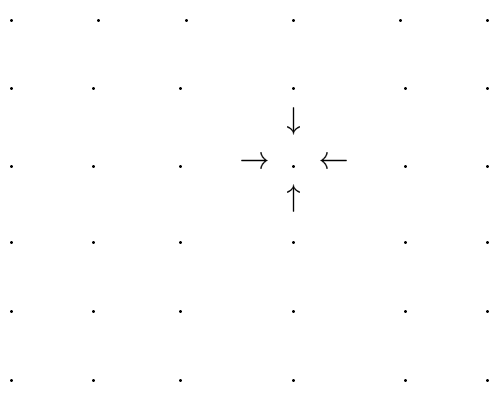
In *Incomplete LU factorization*, we keep the factors artificially sparse in order to save computer time and storage for the decomposition. The incomplete factors are used for preconditioning in the following way. First note that, for all iterative methods discussed, we never need the matrices A or K explicitly, but we only need to be able to compute the result of Ay for any given vector y. The same holds for the preconditioner K, and typically we see in codes that these operations are performed by calls to appropriate subroutines. We need to be able to compute efficiently the result of $K^{-1}y$ for any given vector y. In the case of an incomplete LU factorization, K is given in the form $K = \tilde{L}\tilde{U}$. Note that, in this chapter, $\tilde{L}$ and

$\widetilde{U}$ will denote incomplete factors and the use of L and U will be reserved for the actual LU factors. $z = K^{-1}y$ is computed by solving z from $\widetilde{L}\widetilde{U}z = y$. This is done in two steps: first solve w from $\widetilde{L}w = y$ and then compute z from $\widetilde{U}z = w$. Note that these solution steps are simple backsubstitutions and, if the right-hand sides are not required further in the iterative process, the solution of either back-substitution may overwrite the corresponding right-hand side, in order to save memory. We hope that it is not necessary to stress this, but one should **never** compute the inverse of K, or of its factors, explicitly, **unless** the inverse has some very convenient sparse form (for instance when K is a diagonal matrix).

We shall illustrate the above sketched process for a popular preconditioner for sparse positive definite symmetric matrices, namely, the *incomplete Cholesky factorization* (Golub and Van Loan 1996, Meijerink and van der Vorst 1977, Meijerink and van der Vorst 1981, Varga 1960) with no fill-in. We will denote this preconditioner as IC(0). CG in combination with IC(0) is often referred to as ICCG(0). We shall consider IC(0) for the matrix with five nonzero diagonals, that arises after the 5-point finite-difference discretization of the 2-dimensional Poisson equation over a rectangular region, using a grid of dimensions n_x by n_y. If the entries of the three nonzero diagonals in the upper triangular part of A are stored in three arrays $a(\cdot, 1)$ for the main diagonal, $a(\cdot, 2)$ for the first co-diagonal, and $a(\cdot, 3)$ for the n_x-th co-diagonal, then the i-th row of the symmetric matrix A can be represented as in (2.1).

$$A = \begin{pmatrix} \ddots & & \ddots & \ddots & \ddots & & \ddots & \\ & a_{i-n_x,3} & & a_{i-1,2} & a_{i,1} & a_{i,2} & & a_{i,3} \\ & \ddots & & \ddots & \ddots & \ddots & & \ddots \end{pmatrix} \tag{2.1}$$

This corresponds to the unknowns over a grid as shown below:



If we write A as $A = L_A + diag(A) + {L_A}^T$, in which L_A is the strictly lower triangular part of A, then the IC(0)-preconditioner can be written as

$$K = (L_A + D)D^{-1}({L_A}^T + D).$$

This relation only holds if there are no corrections to off-diagonal non-zero entries in the incomplete elimination process for A and if we ignore all fill-in outside the non-zero structure of A. It is easy to do this for the 5-point Laplacian. For other matrices, we can force the relation to hold only if we ignore also Gaussian elimination corrections at places where A has non-zero entries. This may decrease the effectiveness of the preconditioner, because we then neglect more operations in the Gaussian elimination process.

For IC(0), the entries d_i of the diagonal matrix D can be computed from the relation

$$\mathrm{diag}(K) = \mathrm{diag}(A).$$

For the five-diagonal A, this leads to the following relations for the d_i:

$$d_i = a_{i,1} - a_{i-1,2}^2/d_{i-1} - a_{i-n_x,3}^2/d_{i-n_x}. \tag{2.2}$$

Obviously this is a recursion in both directions over the grid. This aspect will be discussed later when considering the application of the preconditioner in the context of parallel and vector processing.

The so-called *modified incomplete decompositions* (Dupont, Kendall and Rachford Jr. 1968, Gustafsson 1978) follow from the requirement that

$$\mathrm{rowsum}(K) = \mathrm{rowsum}(A) + ch^2. \tag{2.3}$$

The term ch^2 is for grid-oriented problems with mesh-size h. Although in many applications this term is skipped (that is, one often takes $c = 0$), this may lead to ineffective preconditioning or even break-down of the preconditioner, see Eijkhout (1992). In our context, the rowsum requirement in (2.3) amounts to an additional correction to the diagonal entries d_i, compared to those computed in (2.2).

Axelsson and Lindskog (1986) describe a relaxed form of this modified incomplete decomposition that, for the five-diagonal A, leads to the following relations for the d_i:

$$\begin{aligned} d_i \;=\; & a_{i,1} - a_{i-1,2}(a_{i-1,2} + \alpha a_{i-1,3})/d_{i-1} \\ & - a_{i-n_x,3}(a_{i-n_x,3} + \alpha a_{i-n_x,2})/d_{i-n_x}. \end{aligned}$$

Note that, for $\alpha = 0$ we have the standard IC(0) decomposition, whereas for $\alpha = 1$ we have the modified incomplete Cholesky decomposition MIC(0) proposed by Gustafsson (1978). It has been observed that, in many practical situations, $\alpha = 1$ does not lead to a reduction in the number of iteration steps, with respect to $\alpha = 0$, but in our experience, taking $\alpha = .95$ almost always reduces the number of iteration steps significantly (van der Vorst 1989*b*). The only difference between the IC(0) and MIC(0) is the choice of the diagonal D; in fact, the off-diagonal entries of the triangular factors are identical.

For the solution of systems $Kw = r$, given by

$$K^{-1}r = (L_A^T + D)^{-1}D(L_A + D)^{-1}r,$$

it will almost never be advantageous to determine the matrices $(L_A^T + D)^{-1}$ and $(L_A + D)^{-1}$ explicitly, since these matrices are usually dense triangular matrices. Instead, for the computation of, say, $y = (L_A + D)^{-1}r$, y is solved from the linear lower triangular system $(L_A + D)y = r$. This step then leads typically to relations for the entries y_i, of the form

$$y_i = (r_i - a_{i-1,2}y_{i-1} - a_{i-n_x,3}y_{i-n_x})/d_i,$$

which again represents a recursion in both directions over the grid, of the same form as the recursion for the d_i.

For differently structured matrices, we can also perform incomplete LU factorizations. For efficient implementation, often many of the ideas, shown here for incomplete Cholesky factorizations, apply. For more general matrices with the same non-zero structure as the 5-point Laplacian, some other well known approximations lead to precisely the same type of recurrence relations as for Incomplete LU and Incomplete Cholesky: for example, Gauss-Seidel, SOR, SSOR (Hageman and Young 1981), and SIP (Stone 1968). Hence these methods can often be made vectorizable or parallel in the same way as for incomplete Cholesky preconditioning.

Since vector and parallel computers do not lend themselves well to recursions in a straightforward manner, the recursions just discussed may seriously degrade the effect of preconditioning on a vector or parallel computer, if carried out in the form given above. This sort of observation has led to different types of preconditioners, including diagonal scaling, polynomial preconditioning, and truncated Neumann series. Such approaches may be useful in certain circumstances, but they tend to increase the computational complexity (by requiring more iteration steps or by making each iteration step more expensive). On the other hand, various techniques have been proposed to vectorize the recursions, mainly based on reordering the unknowns or changing the order of computation. For regular grids, such approaches lead to highly vectorizable code for the standard incomplete factorizations (and consequently also for Gauss-Seidel, SOR, SSOR, and SIP). If our goal is to minimize computing time, there may thus be a trade-off between added complexity and increased vectorization. However, before discussing these techniques, we shall present a method of reducing the computational complexity of preconditioning.

2.1 Efficient Implementations of ILU(0) Preconditioning

Suppose that the given matrix A is written in the form $A = L_A + \mathrm{diag}(A) + U_A$, in which L_A and U_A are the strictly lower and upper triangular part of A, respectively.

Eisenstat (1981) has proposed an efficient implementation for preconditioned iterative methods, when the preconditioner K can be represented as

$$K = (L_A + D)D^{-1}(D + U_A), \tag{2.4}$$

in which D is a diagonal matrix. Some simple Incomplete Cholesky, incomplete LU, modified versions of these factorizations, as well as SSOR can be written in this form. For the incomplete factorizations, we have to ignore all the LU factorization corrections to off-diagonal entries (Meijerink and van der Vorst 1981); the resulting decomposition is referred to as $ILU(0)$ in the unsymmetric case, and IC(0) for the incomplete Cholesky situation. For the 5-point finite-difference discretized operator over rectangular grids in 2D, this is equivalent to the incomplete factorizations with no fill-in, since in these situations there are no Gaussian elimination operations on non-zero off-diagonal entries.

The first step to make the preconditioning more efficient is to eliminate the diagonal D in (2.4). We rescale the original linear system $Ax = b$ to obtain

$$D^{-1/2}AD^{-1/2}\tilde{x} = D^{-1/2}b, \tag{2.5}$$

or $\tilde{A}\tilde{x} = \tilde{b}$, with $\tilde{A} = D^{-1/2}AD^{-1/2}$, $\tilde{x} = D^{1/2}x$, and $\tilde{b} = D^{-1/2}b$. With $\tilde{A} = L_{\tilde{A}} + \mathrm{diag}(\tilde{A}) + U_{\tilde{A}}$, we can easily verify that

$$\widetilde{K} = (L_{\tilde{A}} + I)(I + U_{\tilde{A}}). \tag{2.6}$$

Note that the corresponding triangular systems, like $(L_{\tilde{A}} + I)r = w$, are more efficiently solved, since the division by the entries of D is avoided. We also note that this scaling does not necessarily have the effect that $\mathrm{diag}(\tilde{A}) = I$.

The key idea in Eisenstat's approach (also referred to as *Eisenstat's trick*) is to apply standard iterative methods (that is, in their formulation with $K = I$) to the explicitly preconditioned linear system

$$(L_{\tilde{A}} + I)^{-1}\tilde{A}(I + U_{\tilde{A}})^{-1}y = (L_{\tilde{A}} + I)^{-1}\tilde{b}, \tag{2.7}$$

where $y = (I + U_{\tilde{A}})\tilde{x}$. This explicitly preconditioned system will be denoted by $Py = c$. Now we can write $\tilde{A}$ in the form

$$\tilde{A} = L_{\tilde{A}} + I + \mathrm{diag}(\tilde{A}) - 2I + I + U_{\tilde{A}}. \tag{2.8}$$

This expression, as well as the special form of the preconditioner given by (2.6), is used to compute the vector Pz for a given z by

$$Pz = (L_{\tilde{A}} + I)^{-1}\tilde{A}(I + U_{\tilde{A}})^{-1}z = (L_{\tilde{A}} + I)^{-1}(z + (\mathrm{diag}(\tilde{A}) - 2I)t) + t, \tag{2.9}$$

with

$$t = (I + U_{\tilde{A}})^{-1}z. \tag{2.10}$$

Note that the computation of Pz is equivalent to solving two triangular systems plus the multiplication of a vector by a diagonal matrix $(\mathrm{diag}(\widetilde{A}) - 2I)$ and an addition of this result to z. Therefore the matrix-vector product for the preconditioned system can be computed virtually at the cost of the matrix-vector product of the unpreconditioned system. This fact implies that the preconditioned system can be solved by any of the iterative methods for practically the same computational cost per iteration step as the unpreconditioned system. That is to say, the preconditioning comes essentially for free, in terms of computational complexity.

In most situations we see, unfortunately, that while we have avoided the fast part of the iteration process (the matrix-vector product Ap), we are left with the most problematic part of the computation, namely, the triangular solves. However, in some cases, as we shall see, these parts can also be optimized to about the same level of performance as the matrix-vector multiplies.

2.2 General Incomplete Decompositions

We have discussed at some length the incomplete Cholesky decomposition for the matrix corresponding to a regular 5-point discretization of the Poisson operator in 2D. It was shown by Meijerink and van der Vorst (1977) that incomplete LU factorizations exist for M-matrices with an arbitrary sparsity structure, where fill-in is only accepted for specified indices.

Let the positions in which corrections to matrix entries may occur (note that this includes both fill-ins and changes to original entries) be given by the index set S, that is

$$l_{i,j} = 0 \quad \text{if} \quad j > i \quad \text{or} \quad (i,j) \notin S; \quad u_{i,j} = 0 \quad \text{if} \quad i > j \quad \text{or} \quad (i,j) \notin S. \tag{2.11}$$

In the previous section, we considered incomplete factorizations with no fill-in outside the sparsity pattern of A; the corresponding S would have been:

$$S = \{(i,j) \quad | \quad a_{i,j} \neq 0\}. \tag{2.12}$$

Since we want the product, K, of the incomplete factors of A, to resemble A as much as possible, a typical strategy is to require the entries of $K = \widetilde{L}\widetilde{U}$ to match those of A on the set S:

$$k_{i,j} = a_{i,j} \quad \text{if} \quad (i,j) \in S. \tag{2.13}$$

The factors $\widetilde{L}$ and $\widetilde{U}$, that satisfy the conditions (2.12) and (2.13), can be computed by a simple modification of the Gaussian elimination algorithm; see Figure 2.1, following Axelsson (1994). The main difference from the usual LU factorization is in the innermost j-loop where an update to $a_{i,j}$ is computed only if it is allowed by the constraint set S. Although we describe the incomplete decomposition for a full matrix (by referring to all entries $a_{i,j}$), it should be clear that we will almost never

```
for r := 1 step 1 until n − 1 do
    d := 1/a_{r,r}
    for i := (r + 1) step 1 until n do
        if (i, r) ∈ S then
            e := d a_{i,r}; a_{i,r} := e;
            for j := (r + 1) step 1 until n do
                if (i, j) ∈ S and (r, j) ∈ S then
                    a_{i,j} := a_{i,j} − e a_{r,j}
                end if
            end (j-loop)
        end if
    end (i-loop)
end (r-loop)
```

Figure 2.1: **ILU** for an n by n matrix A

do this in practice for a given sparse matrix. In practical cases, we only access the entries that belong to a given data structure.
After completion of the algorithm, the incomplete LU factors are stored in the corresponding lower and upper triangular parts of the array A, which means that we have to make a copy of the array A because we need the original matrix for matrix-vector products. In special cases this can be avoided, in particular for situations where we have neglected all elimination corrections to off-diagonal entries in A. This is sometimes done in order to be able to apply Eisenstat's trick, see Section 2.1.

If all fill-ins are allowed, that is if S consists of all possible index pairs, then the above algorithm is simply the usual LU factorization.

Although the ILU preconditioner works quite well for many problems, it can be easily improved for some PDE problems. For example, when ILU is applied to elliptic problems, its asymptotic (as the mesh size h becomes small) convergence rate is only a constant factor better than that of the unpreconditioned A (for an analysis of this factor for IC(0), see van der Vorst (1982*a*)). This was already observed by Dupont et al. (1968) and, for elliptic PDEs, they proposed a simple modification which dramatically improves the performance as h tends to zero. We shall next describe the generalization of this *modified* ILU (MILU) preconditioner to a general matrix A due to Gustafsson (1978).

The condition for the diagonal entries of K, namely $k_{i,i} = a_{i,i}$, is replaced by

$$\sum_{j=1}^{n} k_{i,j} = \sum_{j=1}^{n} a_{i,j} + ch^2 \quad \text{for all } i \ldots \tag{2.14}$$

The addition of the term ch^2 has been recommended for problems stemming from discretized second order PDE's over grids with mesh size h. For more general matrices, we do not know what to add in order to make the incomplete decomposition more stable. Moreover, for more general matrices, MILU may not be very successful because the approximation is tuned so that solutions of equations with the coefficient matrix A with the property that they are only slowly varying over the grid are also solutions when A is replaced by K (modulo corrections of the order of ch^2). In popular implementations, one often takes $c = 0$, but one has to be very careful with this, see Eijkhout (1992).

For diagonally dominant M-matrices, the condition (2.14) with $c \neq 0$ is sufficient to determine the LU factors in MILU directly. However, in practice it is easier to compute these LU factors by a simple modification of the ILU algorithm: instead of dropping the forbidden fill-ins in the ILU algorithm, these terms are added to the main diagonal of the same row; see Figure 2.2. Again, it can be shown that the computed LU factors satisfy (2.14). Note that only the j-loop is different from the ILU algorithm.

```
for r := 1 step 1 until n do a_{r,r} = a_{r,r} + ch^2
for r := 1 step 1 until n − 1 do
    d := 1/a_{r,r}
    for i := (r + 1) step 1 until n do
        if (i, r) ∈ S then
            e := a_{i,r} d; a_{i,r} := e;
            for j := (r + 1) step 1 until n do
                if (r, j) ∈ S then
                    if (i, j) ∈ S then
                        a_{i,j} := a_{i,j} − e a_{r,j}
                    else
                        a_{i,i} := a_{i,i} − e a_{r,j}
                    end if
                end if
            end (j-loop)
        end if
    end (i-loop)
end (r-loop)
```

Figure 2.2: Algorithm **MILU** for a general matrix A

Again, the LU factors overwrite the lower and upper triangular parts of the array

A respectively, and A has to be saved prior to this decomposition operation, since it is required in the iterative process.

Even though MILU produces a better asymptotic condition number bound than ILU for elliptic problems, in practice MILU does not always perform better than ILU. This may have to do with the higher sensitivity of MILU to round-off errors (van der Vorst 1990*a*). This provides motivation for an *interpolated* version between ILU and MILU, see for example Ashcraft and Grimes (1988) and Axelsson and Lindskog (1986). The idea is that in the MILU algorithm, the update of $a_{i,i}$ in the innermost loop is replaced by:

$$a_{i,i} := a_{i,i} - \omega e a_{r,j},$$

where $0 \leq \omega \leq 1$ is a user specified relaxation parameter. Obviously, $\omega = 0$ and 1 correspond to ILU and MILU respectively. It was observed empirically by van der Vorst (1990*a*), and verified using the Fourier analysis method (Chan 1991), that a value of $\omega = 1 - ch^2$ gives the best results for some classes of matrices coming from elliptic problems. The optimal value of c can be estimated and is related to the optimal value of c in the DKR method of Dupont et al. (1968). Notay (1994) gave strategies for choosing $\omega = \omega_{i,j}$ dynamically in order to improve the robustness and performance for anisotropic problems.

2.3 Variants of ILU Preconditioners

Many variants on the theme of incomplete or modified incomplete decomposition have been proposed in the literature. These variants are designed to either reduce the total computational work or to improve the performance on vector or parallel computers. We will describe some of the more popular variants and give references to where more details can be found for other variants.

A natural approach is to allow more fill-in in the LU factor (that is a larger set S), than those allowed by the condition (2.12). Several possibilities have been proposed. The most obvious variant is to allow more fill-ins in specific locations in the LU factors, for example allowing more nonzero bands in the $\widetilde{L}$ and $\widetilde{U}$ matrices (that is larger stencils) (Axelsson and Barker 1984, Gustafsson 1978, Meijerink and van der Vorst 1981). The most common location-based criterion is to allow a set number of levels of fill-in, where original entries have level zero, original zeros have level ∞ and a fill-in in position (i, j) has level determined by

$$Level_{ij} = \min_{1 \leq k \leq \min(i,j)} \{Level_{ik} + Level_{kj} + 1\}.$$

In the case of simple discretizations of partial differential equations, this gives a simple pattern for incomplete factorizations with different levels of fill-in. For example, if the matrix is from a five-point discretization of the Laplacian in two-dimensions, level 1 fill-in will give the original pattern plus a diagonal inside the

outermost band (for instance, see Meijerink and van der Vorst (1981) and Watts-III (1981)).

The other main criterion for deciding which entries to omit is to replace the *drop-by-position* strategy in (2.12) by a *drop-by-size* one. That is, a fill-in entry is discarded if its absolute value is below a certain threshold value. This *drop tolerance* strategy was proposed by Munksgaard (1980), Østerby and Zlatev (1983), and Zlatev (1991). For application to fluid flow problems, see D'Azevedo, Forsyth and Tang (1992) and Young, Melvin, Johnson, Bussoletti, Wigton and Samanth (1989). For the regular problems just mentioned, it is interesting that the level fill-in and drop strategies give a somewhat similar incomplete factorization, because the numerical value of successive fill-in levels decreases markedly, reflecting the characteristic decay in the entries in the factors of the LU decomposition of A. For general problems, however, the two strategies can be significantly different. Since it is usually not known *a priori* how many entries will be above a selected threshold, the dropping strategy is normally combined with restricting the number of fill-ins allowed in each column (Saad 1994). When using a threshold criterion, it is possible to change it dynamically during the factorization to attempt to achieve a target density of the factors (Axelsson and Munksgaard 1983, Munksgaard 1980).

Although the notation is not yet fully standardized, the nomenclature commonly adopted for incomplete factorizations is ILU(k), when k levels of fill-in are allowed and ILUT(α, f), for the threshold criterion when entries of modulus less than α are dropped and the maximum number of fill-ins allowed in any column is f. There are many variations on these strategies and the criteria are sometimes combined. In some cases, constraining the row sums of the incomplete factorization to match those of the matrix, as in MILU, can help (Gustafsson 1978).

Shifts can be introduced to prevent break down of the incomplete factorization process. It was proved by Meijerink and van der Vorst (1977) that incomplete decompositions exist for general M-matrices. It is well known that they may not exist if the matrix is positive definite, but does not have the M-matrix property. Manteuffel (1980) considered incomplete Cholesky factorizations of diagonally shifted matrices. He proved that if A is symmetric positive definite, then there exists a constant $\alpha > 0$, such that the incomplete Cholesky factorization of $A + \alpha I$ exists. Since we make an incomplete factorization for $A + \alpha I$, instead of A, it is not necessarily the case that this factorization is also efficient as a preconditioner; the only purpose of the shift is to avoid breakdown of the decomposition process. Whether there exist suitable values for α such that the preconditioner exists and is efficient is a matter of trial and error.

Another point of concern is that for non M-matrices the incomplete factors of A may be very ill-conditioned. For instance, it has been demonstrated by van der Vorst (1981) that, if A comes from a 5-point finite-difference discretization of $\Delta u + \beta(u_x + u_y) = f$, then for β sufficiently large, the incomplete LU factors may be very ill conditioned even though A has a very modest condition number. Remedies

for reducing the condition numbers of $\widetilde{L}$ and $\widetilde{U}$ have been discussed by Elman (1989) and van der Vorst (1981).

2.4 Some General Comments on ILU

The use of incomplete factorizations as preconditioners for symmetric systems has a long pedigree (Meijerink and van der Vorst 1977) and good results have been obtained for a wide range of problems. An incomplete Cholesky factorization where one level of fill-in is allowed (IC(1)) has been shown to provide a good balance between reducing the number of iterations and the cost of computing and using the preconditioning. Although it may be thought that a preordering that would result in low fill-in for a complete factorization (for example, minimum degree) might be advantageous for an incomplete factorization, it is not true in general (Duff and Meurant, 1989 and Eijkhout, 1991) and sometimes the number of iterations of ICCG(0) (=CG+IC(0)-preconditioning) can double if a minimum degree ordering is used. This effect of reordering is not so apparent for ILUT preconditioners.

The situation with symmetric M-matrices has been analyzed and is well understood. For more general symmetric matrices, the analysis is not as refined, but there is much recent effort to develop preconditioners that can be computed and used on parallel computers. Most of this work has, however, been confined to highly structured problems from discretizations of elliptic partial differential equations in two and three dimensions, see for example van der Vorst (1989*b*). Experiments with unstructured matrices have been reported by Heroux, Vu and Yang (1991) and Jones and Plassmann (1994), with reasonable speed-ups being achieved by Jones and Plassmann (1994).

The situation for unsymmetric systems is, however, much less clear. Although there have been many experiments on using incomplete factorizations and there have been studies of the effect of orderings on the number of iterations (Benzi, Szyld and van Duin 1997, Dutto 1993), there is very little theory governing the behavior for general systems and indeed the performance of ILU preconditioners is very unpredictable. Allowing high levels of fill-in can help but again there is no guarantee, as we have argued in Section 1.

3 Some Other Forms of Preconditioning

3.1 Sparse Approximate Inverse (SPAI)

Of course, the LU factorization is one way of representing the inverse of a sparse matrix in a way that can be economically used to solve linear systems. The main reason why explicit inverses are not used is that, for irreducible matrices, the inverse will always be structurally dense. That is to say, sparse techniques will produce a dense matrix even if some of its entries are zero (Duff, Erisman, Gear and Reid 1988).

However, this need not be a problem if we follow the flavor of ILU factorizations and compute and use a sparse approximation to the inverse. Perhaps the most obvious technique for this is to solve the problem

$$\min_K ||I - AK||_F, \quad [1] \tag{3.1}$$

where K has some fully or partially prescribed sparsity structure. One advantage of this is that this problem can be split into n independent least-squares problems for each of the n columns of K. Each of these least-squares problems only involves a few variables (corresponding to the number of entries in the column of K) and, because they are independent, they can be solved in parallel. With these techniques it is possible (Cosgrove, Diaz and Griewank 1992) to successively increase the density of the approximation to reduce the value of (3.1) and so, in principle, ensure convergence of the preconditioned iterative method. The small least-squares subproblems can be solved by the standard (dense) QR factorizations (Cosgrove et al. 1992, Gould and Scott 1998, Grote and Huckle 1997). In a further attempt to increase sparsity and reduce computational costs in the solutions of the subproblems, it has been suggested to use a few steps of GMRES to solve the subsystems (Chow and Saad 1994). A recent study indicates that the computed approximate inverse may be a good alternative for ILU (Gould and Scott 1998), but it is much more expensive to compute both in terms of time and storage, at least if computed sequentially. This means that it is normally only attractive to use this technique if the computational costs for the construction can be amortized by using the preconditioner for more right-hand sides. One other problem with these approaches is that, although the residual for the approximation of a column of K can be controlled (albeit perhaps at the cost of a rather dense column in K), the nonsingularity of the matrix K is not guaranteed. The singularity of K does not prevent us from multiplying by this preconditioner but will cause us problems if the solution vector has components in the null-space of K. Partly to avoid this, it was proposed to approximate the triangular factors of the inverse (Kolotilina and Yeremin 1993). The non-singularity of the factors can be easily controlled and, if necessary, the sparsity pattern of the factors may also be controlled. Following this approach, it has been suggested to generate sparse approximations to an A-biconjugate set of vectors using drop tolerances (Benzi, Meyer and Tůma 1996, Benzi and Tůma 1998*b*). In a scalar or vector environment, it is also much cheaper to generate the factors in this way than to solve the least-squares problems for the columns of the approximate inverse (Benzi and Tůma 1998*a*).

One of the main reasons for the interest in sparse approximate inverse preconditioners is the difficulty of parallelizing ILU preconditioners, not only in their construction but also in their use, which requires a sparse triangular solution. However, although almost every paper on approximate inverse preconditioners states

[1] We recall that $\| \|_F$ denotes the Frobenius norm of a matrix viz. $\|A\|_F \equiv \sqrt{\sum_{i,j} a_{i,j}^2}$.

that the authors are working on a parallel implementation, it is only quite recently that papers on this have appeared (Barnard, Bernardo and Simon 1997, Barnard and Clay 1997). For highly structured matrices, some experiences have been reported by Grote and Simon (1993). Gustafsson and Lindskog (1995), have implemented a fully parallel preconditioner based on truncated Neumann expansions (van der Vorst 1982*b*) to approximate the inverse SSOR factors of the matrix. Their experiments (on a CM-200) show a worthwhile improvement over a simple diagonal scaling.

Note that, because the inverse of the inverse of a sparse matrix is sparse, there are classes of dense matrices for which a sparse approximate inverse might be a very appropriate preconditioner. This may be the case for matrices that arise from inverse problems (Alleon, Benzi and Giraud 1997). For some classes of problems, it may be attractive to construct the explicit inverses of the LU factors, even if these are considerably less sparse than the factors L and U, because such a factorization can be more efficient in parallel (Alvarado and Schreiber 1993). An incomplete form of this factorization for use as a preconditioner has been proposed by Alvarado and Dağ (1994).

3.2 Polynomial Preconditioning

Of course it is, in theory, possible to represent the inverse by a polynomial in the matrix and one could use this polynomial as a preconditioner. However, one should realize that iterative Krylov subspace methods, such as GMRES, CG, etc., construct approximate solutions in a Krylov subspace. This means that the solutions can be interpreted as polynomials in the (preconditioned) matrix, applied to the right-hand side. Since the Krylov methods construct such solutions with certain optimality properties (for instance minimal residual), it is not so obvious why an additional polynomial might be effective as a preconditioner. The main motivation for considering polynomial preconditioning is to improve the parallel performance of the solver, since the matrix-vector product is often more parallelizable than other parts of the solver (for instance the inner products). The main problem is to find effective low degree polynomials. One approach, reported by Dubois, Greenbaum and Rodrigue (1979), is to use the low order terms of a Neumann expansion of $(I-B)^{-1}$, if A can be written as $A = I - B$ and the spectral radius of B is less than 1. It was suggested by Dubois et al. (1979) to use a matrix splitting $A = K - N$ and a truncated power series for $K^{-1}N$ when the condition on B is not satisfied. More general polynomial preconditioners have also been proposed (see, for example, Ashby, 1991, Johnson, Micchelli and Paul, 1983, Saad, 1985). Because the iterative solvers implicitly construct (optimal) polynomial approximations themselves, using spectral information obtained during the iterations, it is not easy to find effective alternatives without knowing such spectral information explicitly. This may help explain why the experimental results are not generally very encouraging and have been particularly disappointing for unsymmetric problems.

3.3 Preconditioning by Blocks or Domains

Another whole class of preconditioners that use direct methods are those where the direct method, or an incomplete version of it, is used to solve a subproblem of the original problem. This is often used in a domain decomposition setting, where problems on subdomains are solved by a direct method but the interaction between the subproblems is handled by an iterative technique.

If the system is reducible and the matrix is block diagonal, then the solution to the overall problem is just the union of the solution of the subproblems corresponding to the diagonal blocks. Although the overall problem may be very large, it is possible that the subproblems are small enough to be solved by a direct method. This solution is effected by a block Jacobi factorization and the preconditioned blocks of this matrix are just the identity. In general, our system will not be reducible, but it might still be appropriate to use the block Jacobi method as a preconditioner. For general systems, one could apply a block Jacobi preconditioning to the normal equations which would result in the block Cimmino algorithm (Arioli, Duff, Noailles and Ruiz 1992). A similar relationship exists between a block SOR preconditioning and the block Kaczmarz algorithm (Bramley and Sameh 1992). Block preconditioning for symmetric systems is discussed by Concus, Golub and Meurant (1985); in Concus and Meurant (1986), incomplete factorizations are used within the diagonal blocks. Attempts have been made to preorder matrices to put large entries into the diagonal blocks so that the inverse of the matrix is well approximated by the block diagonal matrix whose block entries are the inverses of the diagonal blocks (Choi and Szyld 1996).

In a domain decomposition approach, the physical domain or grid is decomposed into a number of overlapping or non-overlapping subdomains on each of which an independent complete or incomplete factorization can be computed and applied in parallel. The main idea is to obtain more parallelism at the subdomain level rather than at the grid-point level. Usually, the interfaces or overlapping region between the subdomains must be treated in a special manner. The advantage of this approach is that it is quite general and different methods can be used within different subdomains.

Radicati di Brozolo and Robert (1989) used an algebraic version of this approach by computing ILU factors within overlapping block diagonals of a given matrix A. When applying the preconditioner to a vector v, the values on the overlapped region are averaged from the two values computed from the two overlapping ILU factors. The approach of Radicati and Robert has been further refined by De Sturler (1994), who studies the effects of overlap from the point of view of geometric domain decomposition. He introduces artificial mixed boundary conditions on the internal boundaries of the subdomains. In De Sturler (1994):Table 5.8, experimental results are shown for a decomposition into 20×20 slightly overlapping subdomains of a 200×400 mesh for a discretized convection-diffusion equation (5-point stencil). Using an ILU preconditioning on each subdomain, it is shown that the complete linear

system can be solved by GMRES on a 400-processor distributed memory Parsytec system with an efficiency of about 80% (that means that, with this domain adapted preconditioner, the process is about 320 times faster than ILU preconditioned GMRES for the unpartitioned linear system on a single processor).

In Tan (1995), Tan studied the interface conditions along boundaries of subdomains and forced continuity for the solution and some low order derivatives at the interface. He also proposed including mixed derivatives in these relations, in addition to the conventional tangential and normal derivatives. The parameters involved are determined locally by means of normal mode analysis, and they are adapted to the discretized problem. It is shown that the resulting domain decomposition method defines a standard iterative method for some splitting $A = K - N$, and the local coupling aims to minimize the largest eigenvalues of $I - AK^{-1}$. Of course this method can be accelerated, and impressive results for GMRES acceleration are shown by Tan (1995). Some attention is paid to the case where the solutions for the subdomains are obtained with only modest accuracy per iteration step.

Chan and Goovaerts (1990) showed that the domain decomposition approach can actually lead to *improved* convergence rates, at least when the number of subdomains is not too large. This is because of the well known divide and conquer effect when applied to methods with superlinear complexity such as ILU: it is more efficient to apply such methods to smaller problems and piece the global solution together.

Recently, Washio and Hayami (1994) employed a domain decomposition approach for a rectangular grid in which one step of SSOR is performed for the interior part of each subdomain. In order to make this domain-decoupled SSOR more like global SSOR, the SSOR iteration matrix for each subdomain is modified by premultiplying it by a matrix $(I - X_L)^{-1}$ and postmultiplying it by $(I - X_U)^{-1}$. The SSOR preconditioner, with relaxation parameter ω, can then be expressed as

$$K = (I - X_L)^{-1} \left(\frac{D}{\omega} + L_B\right) \left(\frac{D}{\omega}\right)^{-1} \left(\frac{D}{\omega} + U_B\right) (I - X_U)^{-1}.$$

In this expression, D represents the diagonal of the given matrix A and L_B, U_B represent the strict lower and upper triangular parts of A, respectively, in which connections to neighboring domains are set to zero. The idea is to choose X_L and X_U in order to correct for these neglected couplings, and so that K is equal to the full SSOR decomposition. Let L_N, U_N represent the neglected couplings in the lower and upper triangular parts of A, respectively, then one approach is to compute the correction matrices as

$$X_L = L_N \left(\frac{D}{\omega} + L_B\right)^{-1},$$

$$X_U = \left(\frac{D}{\omega} + U_B\right)^{-1} U_N.$$

In order to further improve the parallel performance, the inverses in these expressions are approximated by low-order truncated Neumann series. A similar approach is suggested by Washio and Hayami (1994) for a block modified ILU preconditioner. Experimental results have been reported for a 32-processor NEC Cenju distributed memory computer.

3.4 Element by Element Preconditioners

In finite-element problems, it is not always possible or sensible to assemble the entire matrix, and hence preconditioners are required that can be constructed at the element level. The first to propose such *element by element* preconditioners were Hughes, Levit and Winget (1983). The main idea in these element by element preconditioners is that the element matrices are decomposed into their LU factors and that the back and forward sweeps associated with the (incomplete) LU factorizations are replaced by a series of mini-sweeps for the element factorizations. We will explain this in more detail.

Let $Ax = b$ denote the global assembled system in the finite-element model. The matrix A is assembled from the local element matrices A_e, and we have

$$A = \sum_{e=1}^{n_e} A_e,$$

where n_e denotes the number of elements.

In many codes, the assembly of A is avoided, which may help to reduce memory storage and/or communication with secondary storage. Instead, all computations are done with the local A_e. The preconditioners that we discussed earlier are, however, based on the structure of the global matrix A. Hughes et al proposed a local element preconditioning matrix P_e as follows:

$$P_e = I + D^{-1/2}(A_e - D_e)D^{-1/2},$$

in which D denotes the diagonal of A, and D_e denotes the diagonal of A_e. It is easy to construct D from the A_e, and it is not necessary to assemble A completely for this. The idea is that

$$P \equiv \prod_{e=1}^{n_e} P_e,$$

may be viewed as an approximation for the scaled matrix $D^{-1/2}AD^{-1/2}$, and this matrix P is taken as the element by element preconditioner. In iterative solvers, we then have to solve systems like $Pw = z$, and this is simply done by computing w from

$$w = \prod_{e=n_e}^{1} P_e^{-1} z,$$

that is, a small system with P_e is solved for each relevant section of the right-hand side z (in fact the components of z corresponding to the e^{th} element).

For symmetric positive definite problems, there is a problem, since the product of symmetric matrices is not necessarily symmetric, so that P cannot be used in combination with, for instance, Conjugate Gradients. Hughes et al suggest circumventing this problem by first making a Cholesky decomposition of P_e:

$$P_e = L_e L_e^T.$$

The preconditioner is then taken as $\tilde{P} = LL^T$, defined by

$$L \equiv \prod_{e=1}^{n_e} L_e.$$

In van Gijzen (1994), the parallel implementation of the element by element preconditioner has been discussed. Note that we are free to choose the order in which we number the elements; although each ordering leads formally to a different preconditioner. We may use this 'freedom' to select an ordering that admits some degree of parallelism, and in van Gijzen (1994) it is proposed to subdivide the set of elements into n_g groups, each of n_{eg} nonadjacent elements. It is easily verified that the subproduct of element preconditioning matrices for each group can be written as a sum:

$$\left(\prod_{eg=1}^{n_{eg}} P_{eg}^{-1} \right) z = \sum_{eg=1}^{n_{eg}} P_{eg}^{-1} z,$$

and each term in this sum can be processed in parallel.

A slightly different idea is suggested by Gustafsson and Lindskog (1986). In that paper it is suggested, for symmetric positive-definite A, to decompose each A_e as $A_e = L_e L_e^T$, and to construct the preconditioner as $K = LL^T$, with

$$L = \sum_{e=1}^{n_e} L_e.$$

Since L is not necessarily a lower triangular matrix, this has to be forced explicitly by performing the local node numbering so that increasing local node numbers correspond to increasing global node numbers. Since the L_e may be singular, it is further suggested to improve the numerical stability of L (by increasing the values of the diagonal entries relative to the off-diagonal entries), by replacing L with L_ξ:

$$L_\xi = \frac{1}{1+\xi h} \widehat{L} + (1+\xi h) D_L,$$

where $\widehat{L}$ denotes the off-diagonal part of L, D_L represents the diagonal of L, h is a measure for the size of the finite elements, and ξ is a user defined value of order 1. Also for this approach, we can treat non-adjacent elements in parallel.

There may be other reasons for considering element by element inspired preconditioners. In many realistic models there is some local problem, for instance the formation of cracks under the influence of point forces in concrete or other materials. It may then seem logical to assemble the elements, around the place where some particular effect is expected, into some super-element. We can then form either complete or incomplete decompositions of these super-elements, depending on their size or complexity, and repeat the above procedure with the mix of super-elements and remaining regular elements. This approach bridges the gap between element by element preconditioning and the (incomplete) LU factorization of the fully assembled matrix, and it seems plausible that the effect of the preconditioning based on super-elements will more and more resemble the preconditioning based on the fully assembled matrix as the super-elements grow in size. Some promising results have been obtained by this super-element technique (see, for instance, Daydé, L'Excellent and Gould, 1997, van Gijzen, 1994), but the selection of appropriate super-elements is still an open question.

4 Vector and Parallel Implementation of Preconditioners

4.1 Partial Vectorization

A common approach for the vectorization of the preconditioning part of an algorithm is known as *partial vectorization*. In this approach, the nonvectorizable loops are split into vectorizable parts and nonvectorizable remainders. Schematically, this approach can be explained as follows. If we assume that the preconditioner is written in the form $K = \widetilde{L}\widetilde{U}$, where $\widetilde{L}$ is lower triangular and $\widetilde{U}$ is upper triangular then, as we discussed before, solving w from $Kw = r$ consists in solving $\widetilde{L}y = r$ and $\widetilde{U}w = y$ successively. Both systems lead to similar vectorization problems, and therefore we consider only the partial vectorization of the computation of y from $\widetilde{L}y = r$. The first step is to regard $\widetilde{L}$ as a block matrix with blocks of suitably chosen sizes (not all the blocks need to have equal size):

$$\widetilde{L} = \begin{pmatrix} L_{1,1} & & & & \\ L_{2,1} & L_{2,2} & & & \\ L_{3,1} & L_{3,2} & L_{3,3} & & \\ . & . & . & . & \\ . & . & . & . & \\ L_{n,1} & L_{n,2} & . & . & L_{n,n} \end{pmatrix}.$$

Next, we partition the vectors y and r conformally in subvectors y_i and r_i, so that the vector length of the ith subvector is equal to the block size of $L_{i,i}$. The subvector y_i is then the solution of

$$L_{i,i}y_i = r_i - (L_{i,1}y_1 + L_{i,2}y_2 + \cdots + L_{i,i-1}y_{i-1}).$$

Note that the amount of work for computing all the partitions of y successively is equal to the amount of work for solving $\widetilde{L}y = r$ in a straightforward manner. However, by rearranging the loops for the subblocks, we see that computations for the right-hand side, for each subvector y_i, can be vectorized.

For the five-point finite-difference matrix A, we take the block size equal to n_x, the number of grid points in the x-direction. In that case the standard incomplete decomposition of A leads to a factor $\widetilde{L}$ for which the $L_{i,i}$ are lower bidiagonal matrices, the $L_{i,i-1}$ are diagonal matrices, and all of the other $L_{i,j}$ vanish. Hence, the original nonvectorizable three-term recurrence relations are now replaced by two-term recurrence relations of length n_x, and vectorizable statements of length n_x also.

We have thus vectorized half of the work in the preconditioning step, so that the performance of this part almost doubles. In practice, the performance is often even better because, for many machines, optimized software is available for two-term recurrence relations and the Fortran compiler often automatically replaces this type of computation by the optimized code.

We illustrate the effect of partial vectorization by an example. If our five-diagonal model problem is solved by the preconditioned CG algorithm, and the vectors are of length n, the operation count per iteration step is roughly composed as follows: $6n$ flops for the three vector updates, $4n$ flops for the two inner products, $9n$ flops for the matrix-vector product, and $8n$ flops for solving $Kw = r$, if we assume that A has been scaled such that the factors of K have unit diagonal. Assuming that the first $19n$ flops are executed at a very high vector speed and that the preconditioning part is not vectorized and runs at a speed of S Mflop/s, we conclude, using *Amdahl's law*, that the Megaflop rate for one preconditioned CG iteration step is approximately given by

$$27/(8/S) \simeq 3.4S \text{ Mflop/s}.$$

Since for most existing vector computers S is rather modest, the straightforward preconditioned CG algorithm (as well as other iterative methods) has a disappointingly low performance. Note that applying Eisenstat's trick (Section 2.1) does not lower the CPU time noticeably in this case, since the preconditioning is really the bottleneck. With partial vectorization, we find that the Megaflop rate will be approximately

$$27/(4/S_1) \simeq 6.8S_1 \text{ Mflop/s},$$

where S_1 is the Megaflop rate for a two-term recursion. For many computers, S_1 can be twice as large as S. In practice, the modest block size of the subblocks $L_{i,i-1}$ will also often inhibit high Megaflop rates for the vectorized part of the

preconditioning step. Nevertheless, it is not uncommon to observe in practice that partial vectorization more than doubles the performance.

For most parallel computers and many preconditioners, the performance of the preconditioned CG process is so low that the reduction in the number of iteration steps (because of preconditioning) is not reflected by a comparable reduction in CPU time, with respect to the unpreconditioned process. In other words, we have to seek better parallelizable or vectorizable preconditioners in order to beat the unpreconditioned CG process with respect to CPU time (see also Section 1).

4.2 Reordering the Unknowns

A standard trick for exploiting parallelism is to select all unknowns that have no direct relationship with each other and to number them first. This is repeated for the remaining unknowns. For the five-point finite-difference discretization over rectangular grids, this approach is known as a *red-black ordering*. For more complicated discretizations, graph coloring techniques can be used to decouple the unknowns in large groups. In either case, the effect is that the matrix is permuted correspondingly and can be written, after reordering, in block form as

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} & A_{1,3} & . & . & A_{1,s} \\ A_{2,1} & A_{2,2} & A_{2,3} & . & . & A_{2,s} \\ A_{3,1} & A_{3,2} & A_{3,3} & . & . & A_{3,s} \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ A_{s,1} & A_{s,2} & A_{s,3} & . & . & A_{s,s} \end{pmatrix}$$

such that all the block matrices $A_{j,j}$ are diagonal matrices, with the exception of irregularly structured problems when $A_{s,s}$ may not be diagonal. For example, for red-black ordering we have $s = 2$. Then the incomplete LU factorization K of the form

$$K = (L_A + D)D^{-1}(U_A + D)$$

with L_A and U_A equal to the strict lower and strict upper triangular part of A, respectively, leads to factors $L_A + D$ and $U_A + D$ that can be represented by the same nonzero structure, for example

$$L_A = \begin{pmatrix} D_{1,1} & & & & & \\ A_{2,1} & D_{2,2} & & & & \\ A_{3,1} & A_{3,2} & D_{3,3} & & & \\ . & . & . & . & & \\ . & . & . & . & . & \\ A_{s,1} & A_{s,2} & . & . & . & D_{s,s} \end{pmatrix}.$$

The corresponding triangular system $(L_A + D)y = r$ can be solved in an obvious way by exploiting the block structure on a vector or parallel computer. The required matrix-vector products for the subblocks $A_{i,j}$ can be optimized as in Section 4.1 (with, of course, smaller vector lengths than for the original system).

For the five-point finite-difference matrix A, the red-black ordering leads to a very vectorizable (and parallel) preconditioner. The performance of the preconditioning step is as high as the performance of the matrix-vector product. This implies that the preconditioned processes, when applying Eisenstat's trick, can be coded so that an iteration step of the preconditioned algorithm takes approximately the same amount of CPU time as for the unpreconditioned method. Hence any reduction in the number of iteration steps, resulting from the preconditioner, translates immediately to almost the same reduction in CPU time.

One should realize, however, that in general the factors of the incomplete LU factorization of the permuted matrix A are not equal to the similarly permuted incomplete factors of A itself. In other words, changing the order of the unknowns leads in general to a different preconditioner. This fact should not necessarily be a drawback, but often the reordering appears to have a rather strong effect on the number of iterations, so that it can easily happen that the parallelism or vectorizability obtained is effectively degraded by the increase in (iteration) work. Of course, it may also be the other way around—that reordering leads to a decrease in the number of iteration steps as a free bonus to the parallelism or vectorization obtained. For standard five-point finite-difference discretizations of second-order elliptic PDEs, Duff and Meurant (1989) report on experiments that show that most reordering schemes (including nested dissection and red-black orderings) lead to a considerable increase in iteration steps (and hence in computing time) compared with the standard lexicographical ordering[2]. For an analysis of these effects, see Eijkhout (1991) and Doi (1991). As noted before, this may work out differently in other situations, but one should be aware of these possible adverse effects.

For red-black ordering, it can be shown that the condition number of the preconditioned system is only about one quarter that of the *unpreconditioned* system for ILU, MILU and SSOR, with no asymptotic improvement as h tends to zero (Kuo and Chan 1990).

One way to obtain a better balance between parallelism and fast convergence is to use more colors (Doi 1991). In principle, since there is not necessarily any independence between different colors, using more colors decreases the parallelism but increases the global dependence and hence the convergence. In Doi and Hoshi (1992) up to 75 colors are used for a 76^2 grid on the NEC SX-3/14 resulting in a 2 Gflop/s performance, which is much better than that for the wavefront ordering, see Chapter 4.3. With this large number of colors the speed of convergence for the

[2]An exception seems to be a class of parallel orderings introduced by van der Vorst (1987). This will be described in Section 4.5

preconditioned process is virtually the same as with lexicographical ordering (Doi 1991).

The concept of *multi-coloring* has been generalized to unstructured problems by Jones and Plassmann (1994). They propose effective heuristics for the identification of large independent subblocks in a given matrix. For problems large enough to get sufficient parallelism in these subblocks, their approach leads to impressive speed-ups on parallel computers, in comparison with the natural ordering on one single processor.

Meier and Sameh (1988) report on the parallelization of the preconditioned CG algorithm for a multivector processor with a hierarchical memory (for example the Alliant FX series). Their approach is based on a red-black ordering in combination with forming a reduced system (Schur complement).

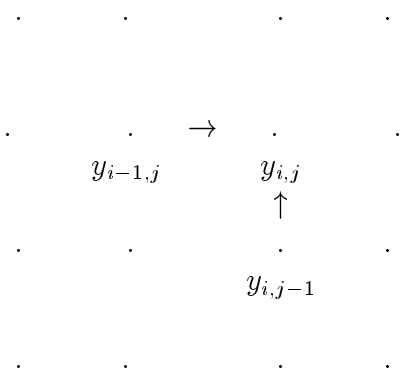
4.3 Changing the Order of Computation

In some situations it is possible to change the order of the computations (by implicitly reordering the unknowns) without changing the results. This means that bitwise the same results are produced, with the same roundoff effects. The only effect is that the order in which the results are produced may differ from the standard lexicographical ordering. A prime example is the incomplete LU preconditioner for the five-point finite-difference operator over a rectangular grid.

We now number the vector and matrix entries according to the position in the grid, that is $y_{i,j}$ refers to the component of y corresponding to the i,j-th grid point (i in x-direction and j in y-direction). The typical expression in the solution of the lower triangular system $\widetilde{L}y = r$ is

$$y_{i,j} = (r_{i,j} - a_{i-1,j,2}\, y_{i-1,j} - a_{i,j-1,3}\, y_{i,j-1})/d_{i,j},$$

where $y_{i,j}$ depends only on its previously computed neighbors in the west and south directions over the grid:



(4.1)

Hence the unknowns $y_{i,j}$ corresponding to the grid points along a diagonal of the grid, that is $i + j = constant$, depend only on the values of y corresponding to the previous diagonal. Therefore, if we compute the unknowns in the order corresponding to these diagonals over the grid, for each diagonal the y-values can be computed independently, or in vector mode.

A number of complications arise, however. If we do not wish to rearrange the unknowns in memory explicitly, then a non-unit stride is involved in the vector operations; typically this non-unit stride is $n_x - 1$, where n_x denotes the number of grid points in x-direction. For some computers a non-unit stride is not attractive (for instance, for machines with a relatively small cache), while on others one might encounter a severe degradation in performance because of memory bank conflicts.

The other problem is that the vectorizable length for the preconditioning part is only $\min(n_x, n_y)$ at most, and many of the loops are shorter. Thus, the average vector length may be quite small, and it really depends on the $n_{1/2}$ value whether the diagonal approach is profitable on a given architecture. Moreover, there are typically more diagonals in the grid than there are grid lines, which means that there are about $n_x + n_y - 1$ vector loops, in contrast to only $\min(n_x,n_y)$ (unvectorized) loops in the standard lexicographical approach. Some computers have well optimized code for recursions over one grid line. Again, it then depends on the situation whether the additional overhead for the increased number of loops offsets the advantage of having (relatively short) vector loops. Therefore, it may be advisable to explicitly reorder the unknowns corresponding to grid diagonals.

In three-dimensional problems, there are even more possibilities to obtain vectorizable or parallel code. For the standard seven-point finite-difference approximation of elliptic PDEs over a rectangular regular grid, the obvious extension to the diagonal approach in two dimensions is known as the hyperplane ordering. We now explain this in more detail. From now on, the unknowns as well as the matrix coefficients will be indicated by three indices i, j, k, so that i refers to the index of the corresponding grid point in the x-direction, and j and k likewise in the y and z-directions. The typical relation for solving the lower triangular system in three dimensions is as follows:

$$\begin{aligned} y_{i,j,k} = (r_{i,j,k} &- a_{i-1,j,k,2}\, y_{i-1,j,k} - a_{i,j-1,k,3}\, y_{i,j-1,k} \\ &- a_{i,j,k-1,4}\, y_{i,j,k-1})/d_{i,j,k}. \end{aligned} \tag{4.2}$$

The hyperplane H^m is defined as the collection of grid points for which the triples (i, j, k) have equal sum $i + j + k = m$. Then it is obvious that all unknowns corresponding to H^m can be computed independently (that is in vector mode or in parallel) from those corresponding to H^{m-1}. This approach leads to vector lengths of $\mathcal{O}(N^{2/3})$, where the grid is $N \times N \times N$, but the difficulty is that the unknowns required for two successive hyperplanes are not located as vectors in memory, and indirect addressing is the standard technique to identify the unknowns over the hyperplanes.

On all supercomputers, indirect addressing degrades the performance of the computation. Sometimes this is because of the overhead in computing indices, in other cases it is because of the fact that cache memories cannot be used effectively. In van der Vorst (1989*b*), the reported performance for ICCG(0) in three dimensions shows that this method, with the hyperplane ordering, can hardly compete with standard conjugate gradient applied to the diagonally scaled system, because of the adverse effects of indirect addressing. However, in Schlichting and van der Vorst (1989) and van der Vorst (1989*a*) ways are presented that may help to circumvent these degradations in performance. The main idea is to rearrange the unknowns explicitly in memory, corresponding to the hyperplane ordering, where the ordering within each hyperplane is chosen suitably. The more detailed description that follows has been taken from Schlichting and van der Vorst (1989).

With respect to the hyperplane ordering, equation (4.2) is replaced by the set of equations in Figure 4.3.

for $m = 4, 5, 6, ..., n_x + n_y + n_z$
  for $(i, j, k) \in H^m$:
(a) $y_{i,j,k} = r_{i,j,k} - a_{i,j,k-1,4}\, y_{i,j,k-1}$
(b) $y_{i,j,k} = y_{i,j,k} - a_{i,j-1,k,3}\, y_{i,j-1,k}$
(c) $y_{i,j,k} = y_{i,j,k} - a_{i-1,j,k,2}\, y_{i-1,j,k}$
(d) $y_{i,j,k} = y_{i,j,k}/d_{i,j,k}$

Figure 4.3: Hyperplane dependencies

We have separated step (c). In practical implementations, it is advisable to scale the given linear system such that $d_{i,j,k} = 1$ for all (i, j, k). We shall discuss only part (c); the others can be vectorized similarly. Part (c) is rewritten as in Figure 4.4. This scheme defines the ordering of the unknowns within one hyperplane. The obvious way to implement the algorithm in Figure 4.4 is to store $y_{i,j,k}$ and $a_{i-1,j,k,2}$ in the order in which they are required over H^m. This has to be done only once at the start of the iteration process. Of course, this suggests that we have to store the matrices twice, but this is not really necessary, as we have shown by using Eisenstat's trick, in Section 2.1.

Although the entries of y and $a(\star, 2)$ have been reordered, indirect addressing is still required for the entries $y_{i-1,j,k}$ corresponding to H^{m-1}. Schematically the algorithm in Figure 4.4 can be implemented by the following steps:

1. The required entries $y_{i-1,j,k}$ are gathered into an array V, in the order in which they are required to update the $y_{i,j,k}$ over H^m. The "gaps" in V, when H^m is larger than H^{m-1}, are left zero.

```
for i = max(2, m − n_y − n_z), ..., min(n_x, m − 2)
    for j = max(1, m − i − n_z), ..., min(n_y, m − i − 1)
        k = m − i − j
        y_{i,j,k} = y_{i,j,k} − a_{i−1,j,k,2} y_{i−1,j,k}
    end j
end i
```

Figure 4.4: Hyperplane dependencies, step (c)

2. The entries of V are multiplied by the $a_{i-1,j,k,2}$, which are already in the desired order.

3. The result from the previous step is subtracted component-wise from the $y_{i,j,k}$ corresponding to H^m.

It was reported by Schlichting and van der Vorst (1989) and van der Vorst (1989*a*) that this approach can lead to a satisfactory performance; such a performance has been demonstrated for machines that gave a bad Megaflop rate for the standard hyperplane approach with indirect addressing. For the CM-5 computer a similar approach was developed by Berryman, Saltz, Gropp and Mirchandaney (1990). The hyperplane approach can be viewed as a special case of the more general *wavefront ordering*, for general sparse matrices, discussed by Radicati di Brozolo and Vitaletti (1986). The success of the wavefront ordering approaches depends very much on how well a given computer can handle indirect addressing. In general, the wavefront ordering approach gives too little control for obtaining efficient forms of parallelism.

4.4 Some Other Vectorizable Preconditioners

Of course, many suggestions have been made for the construction of vectorizable preconditioners. The simplest is diagonal scaling, where the matrix A is scaled symmetrically so that the diagonal of the scaled matrix has unit entries. This is known to be quite effective, since it helps to reduce the condition number (Forsythe and Strauss 1955, van der Sluis 1969) and often has a beneficial influence on the convergence behavior. On some vector computers, the computational speed of the resulting iterative method (without any further preconditioning) is so high that it is often competitive with many of the approaches that have been suggested in previous sections (Hayami and Harada 1985, van der Vorst 1989*b*).

Nevertheless, in many situations more powerful preconditioners are needed, and many vectorizable variants of these have been proposed. One of the first suggestions

was to approximate the inverse of A by a truncated Neumann series (Dubois et al. 1979). When A is diagonally dominant and scaled such that $\mathrm{diag}(A) = I$, then it can be written as $A = I - B$, and A^{-1} can be evaluated in a Neumann series as

$$A^{-1} = (I - B)^{-1} = I + B + B^2 + B^3 + \cdots. \tag{4.3}$$

Dubois et al. Dubois et al. (1979) suggest taking a truncated Neumann series as the preconditioner, that is approximating A^{-1} by

$$K^{-1} = I + B + B^2 + \cdots + B^p. \tag{4.4}$$

By observing that this preconditioner, K^{-1}, can be written as a pth degree polynomial P in A, it is obvious that all the iterative methods now lead to iteration vectors x_i in the Krylov subspace that is formed with $P(A)A$ (instead of A, as for the unpreconditioned methods). That is, after m iteration steps we arrive at a Krylov subspace of restricted form: it contains only powers of $P(A)A$ times the starting residual. This is in contrast with the regular Krylov subspace that is obtained after $m(p+1)$ iteration steps with the unpreconditioned method, and that contains also all intermediate powers of A. In both cases, the amount of work spent in matrix-vector multiplications is the same; hence, at the cost of more iterations, the unpreconditioned process can lead, in theory, to a better approximation for the solution, since it has a larger subspace at its disposal. Therefore polynomial preconditioning will seldom lead to significant savings. Any possible gain is due to the fact that the overhead in the polynomial preconditioned case may be smaller.

More sophisticated polynomial preconditioners are obtained when arbitrary coefficients are allowed in the polynomial expansion for A^{-1} (Johnson et al. 1983, Saad 1985). They still suffer from the same disadvantage in that they generate approximate solutions in Krylov subspaces of a restricted form, at the cost of the same number of matrix-vector products for which the unpreconditioned method generates a "complete" Krylov subspace. However, they can certainly be of advantage in a parallel environment, since they reduce the effect of synchronization points in the method. Another advantage is that they may lead to an "effective" Krylov subspace (that is, containing only the powers of A that really matter) in fewer iteration steps with less loss of orthogonality. As far as we know, this point has not yet been investigated.

Obviously, the inverse of A is better approximated by a truncated Neumann series of a fixed degree when A is more diagonally dominant. This is the idea behind a truncated Neumann series approach suggested by van der Vorst (1982*b*). First, an incomplete factorization of A is constructed. To simplify the description, we assume that A has been scaled such that the diagonal entries in the factors of the incomplete decomposition are equal to 1 (see Section 2.1):

$$K = (\widetilde{L} + I)(I + \widetilde{U}). \tag{4.5}$$

Then the factors are written in some suitable block form, as in Section 4.1, viz.

$$(\tilde{L}+I)=\begin{pmatrix} L_{1,1} & & & & & \\ L_{2,1} & L_{2,2} & & & & \\ L_{3,1} & L_{3,2} & L_{3,3} & & & \\ . & . & . & . & & \\ . & . & . & . & . & \\ L_{m,1} & L_{m,2} & . & . & . & L_{m,m} \end{pmatrix}.$$

If the computation of $L_{i,i}^{-1}r_i$ were an efficient vectorizable operation, then the complete process of solving $(\tilde{L}+I)z=r$ could be vectorized, because segments z_i of z, of dimension equal to the order of $L_{i,i}$, are obtained from the equation:

$$z_i = L_{i,i}^{-1}(r_{i,i} - L_{i,i-1}z_{i-1} - ... - L_{i,1}z_1) \tag{4.6}$$

(assuming that the operations $L_{i,j}z_j$ are vectorizable operations). In many relevant situations it happens that the factors $\tilde{L}+I$ and $I+\tilde{U}$ are diagonally dominant when A is diagonally dominant, and one may then expect that the subblocks $L_{i,i}$ are even more diagonally dominant. van der Vorst (1982*b*) has proposed using truncated Neumann series only for the inversion of these diagonal blocks of $\tilde{L}$ (and $\tilde{U}$). He has shown both theoretically and experimentally that only a few terms in the Neumann series, say two or three, suffice to get an efficient (and vectorizable) process, for problems that come from five-point finite-difference approximations in two dimensions. In most situations, there is a price to be paid for this vectorization, in that the number of iteration steps increases slightly and also the number of floating-point operations per iteration step increases by $4N$ (for the 2-term truncated variant). van der Vorst (1982*b*) has shown for a model problem that the increase in iteration steps is modest when only two terms in the Neumann series are used. In van der Vorst (1989*b*), this approach is extended to the three-dimensional situation, where $I+\tilde{L}$ can be viewed as a nested block form. The resulting method, which has the name "nested truncated Neumann series", leads to rather long vector lengths and can be attractive for some special classes of problem on some parallel computers.

Finally, we comment on a vectorizable preconditioner that has been suggested by Meurant (1984*a*). The starting point is a so-called block preconditioner, that is a preconditioner of the form

$$K=(\tilde{L}+B)B^{-1}(B+\tilde{U}), \tag{4.7}$$

in which B itself is a block diagonal matrix. This type of preconditioner has been suggested by many authors (Concus et al. 1985, Kettler 1987, Meijerink 1983). Most of these block preconditioners differ in the choice of B. They are reported to be quite effective in two-dimensions (in which case A is block tridiagonal) in that they significantly reduce the number of iteration steps for many problems. However, in three-dimensions, experience leads to less favorable conclusions (see, for example,

Kettler, 1987). Moreover, for vector computers, they share the drawback that the inversion of the diagonal blocks of B (which are commonly tridiagonal matrices) can lead to rather poor performance. Meurant (1984*a*) has proposed a variant to a block preconditioner introduced by Concus et al. (1985), in which he approximates the inverses of these tridiagonal blocks of B by some suitably chosen band matrices. He reports on results for some vector computers (CRAY-1, CRAY X-MP, and CYBER 205) and shows that this approach leads often to lower CPU times than the truncated Neumann series approach.

4.5 Parallel Aspects of Reorderings

By reordering the unknowns, a matrix structure can be obtained that allows for parallelism in the triangular factors representing the incomplete decomposition. The red-black ordering, for instance, leads to such a highly parallel form. As has been mentioned before, this reordering often leads to an increase in the number of iteration steps, with respect to the standard lexicographical ordering.

Of more interest is the effort that has been put into constructing a parallel preconditioner, since these attempts are also relevant for the other iterative methods.

Let us write the triangular factors of K in block bidiagonal form:

$$\widetilde{L} = \begin{pmatrix} L_{1,1} & & & & & & \\ L_{2,1} & L_{2,2} & & & & & \\ & L_{3,2} & L_{3,3} & & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & L_{p,p-1} & L_{p,p} \end{pmatrix}$$

For p not too small, Seager (1986) suggests setting some of the off-diagonal blocks of $\widetilde{L}$ to zero (and to do so in a symmetric way in the upper triangular factor $\widetilde{U}$). Then the back substitution process is decoupled into a set of independent back substitution processes. The main disadvantage of this approach is that often the number of iteration steps increases, especially when more off-diagonal blocks are discarded.

Another approach, suggested by Meurant (1984*b*), exploits the idea of the two-sided (or twisted) Gaussian elimination procedure for tridiagonal matrices (Babuska 1972, van der Vorst 1987). This is generalized for the incomplete factorization. In this approach, K is written as ST, where S takes the (twisted) form

$$\begin{pmatrix} S_{1,1} & & & & & & & & \\ S_{2,1} & S_{2,2} & & & & & & & \\ & S_{3,2} & S_{3,3} & & & & & & \\ & & \cdot & \cdot & & & & & \\ & & & S_{p-1,p} & S_{p,p} & S_{p,p+1} & & & \\ & & & & & S_{p+1,p+1} & S_{p+1,p+2} & & \\ & & & & & & \cdot & \cdot & \\ & & & & & & & S_{q,q} & \end{pmatrix}$$

(and T has a block structure similar to S^T). This approach can be viewed as starting the (incomplete) factorization process simultaneously at both ends of the matrix A. This factorization is discussed in Chapters 7.4 and 7.5 of the LINPACK Users' Guide (Dongarra, Bunch, Moler and Stewart 1979) where it is attributed to Jim Wilkinson. It is colloquially referred to as the BABE algorithm (for **B**urn **A**t **B**oth **E**nds).

van der Vorst (1987) has shown how this procedure can be done in a nested way for the diagonal blocks of S (and T). For the two-dimensional five-point finite-difference discretization over a rectangular grid, the first approach comes down to reordering the unknowns (and the corresponding equations) as

$$\begin{array}{cccccc} 1 & 3 & 5 & 7 & 9 & 11 \\ 13 & 15 & 17 & 19 & 21 & 23 \\ \rightarrow & & & & & \\ & \cdot & \cdot & \cdot & \cdot & \\ & & & & & \leftarrow \\ 24 & 22 & 20 & 18 & 16 & 14 \\ 12 & 10 & 8 & 6 & 4 & 2 \end{array} \tag{4.8}$$

while the nested (twisted) approach is equivalent to reordering the unknowns as

$$\begin{array}{cccc|cccc} 1 & 5 & 9 & 13 & 14 & 10 & 6 & 2 \\ 17 & 21 & 25 & 29 & 30 & 26 & 22 & 18 \\ 33 & 37 & 41 & 45 & 46 & 42 & 38 & 34 \\ \hline 35 & 39 & 43 & 47 & 48 & 44 & 40 & 36 \\ 19 & 23 & 27 & 31 & 32 & 28 & 24 & 20 \\ 3 & 7 & 11 & 15 & 16 & 12 & 8 & 4 \end{array} \tag{4.9}$$

That is, we start numbering from the four corners of the grid in an alternating manner. It is obvious that the original twisted approach leads to a process that can be carried out almost entirely in two parallel parts, while the nested form can be done almost entirely in four parallel parts. Similarly, in three dimensions, the incomplete

decomposition, as well as the triangular solves, can be done almost entirely in eight parallel parts. Van der Vorst (1987,1989*a*) reports a slight decrease in the number of iteration steps for these parallel versions, with respect to the lexicographical ordering. Duff and Meurant (1989) have compared the preconditioned conjugate gradient method for a large number of different orderings, such as nested dissection and red-black, zebra, lexicographical, Union Jack, and nested parallel orderings. The nested parallel orderings are among the most efficient; thus, they are good candidates even for serial computing, and parallelism here comes as a free bonus.

At first sight, there might be some problems for parallel vector processors because, in the orderings we have just sketched, the subsystems are in lexicographical order and hence not completely vectorizable. Of course, these subgroups could be reordered diagonally:

$$\begin{array}{cccc|cccc} 1 & 5 & 13 & 25 & 28 & 16 & 7 & 2 \\ 6 & 14 & 26 & 37 & 39 & 29 & 17 & 8 \\ 15 & 27 & 38 & 45 & 46 & 40 & 30 & 18 \\ \hline 21 & 33 & 42 & 47 & 48 & 44 & 36 & 24 \\ 10 & 20 & 32 & 41 & 43 & 35 & 23 & 12 \\ 3 & 9 & 19 & 31 & 34 & 22 & 11 & 4 \end{array} \tag{4.10}$$

which then leads to vector code as shown in Section 4.3. The disadvantage is that in practical situations the vector lengths will only be small on average. In van der Vorst (1989*a*), alternative orderings are suggested, based on carrying out the twisted factorization in a diagonal fashion. For example, in the two-dimensional situation the ordering could be

$$\begin{array}{cccccccccc} 1 & 3 & 7 & 13 & . & . & . & . & . & . \\ 5 & 9 & 15 & . & . & . & . & . & . & . \\ 11 & 17 & . & . & . & . & . & . & . & . \\ 19 & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & 20 \\ . & . & . & . & . & . & . & . & 18 & 12 \\ . & . & . & . & . & . & . & 16 & 10 & 6 \\ . & . & . & . & . & . & 14 & 8 & 4 & 2 \end{array} \tag{4.11}$$

which leads to a process that can be done almost entirely in parallel (except for the grid diagonal in the middle, which is coupled to both groups), and each group can be done in vector mode, just as shown in Section 4.3. Of course, this can be generalized to three dimensions, leading to four parallel processes, each vectorizable. The twisted factorization approach can also be combined with the hyperplane approach in Section 4.3.

In van der Vorst (1987), it has been mentioned that these twisted incomplete factorizations can be implemented in the efficient manner proposed by Eisenstat

(1981) (see also Section 2.1), since they satisfy the requirement that the entries in corresponding locations in A be equal to the off-diagonal entries of $\tilde{S}$ and $\tilde{T}$ in

$$K = ((\tilde{S} + D)D^{-1})(D + \tilde{T}), \tag{4.12}$$

with $\tilde{S} + D = SD^{-1/2}$, $D^{-1/2}T = D + \tilde{T}$.

4.6 Experiences with Parallelism

Although the problem of finding efficient parallel preconditioners has not been fully solved, it may be helpful to discuss some experimental results for some of the previously discussed approaches. All of the results have been reported for the nicely structured systems coming from finite-difference discretizations of elliptic PDEs over two-dimensional and three-dimensional rectangular grids.

Radicati di Brozolo and Robert (1989) suggest partitioning the given matrix A in (slightly) overlapping blocks along the main diagonal, as in Figure 4.5.

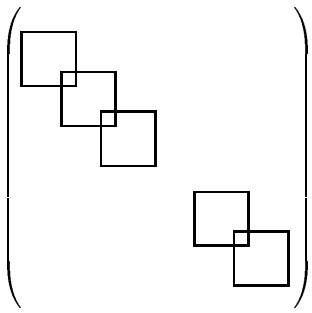


Figure 4.5: Overlapping blocks in A

Note that a given nonzero entry of A is not necessarily contained in one of these blocks. However, experience suggests that this approach is more successful if these blocks cover all the nonzero entries of A. The idea is to compute in parallel local preconditioners for all of the blocks, for example

$$A_i = L_i D_i^{-1} U_i - R_i. \tag{4.13}$$

Then, when solving $Kw = r$ in the preconditioning step, we partition r in (overlapping) parts r_i, according to A_i, and we solve the systems $L_i D_i^{-1} U_i w_i = r_i$ in parallel. Finally we define the components of w to be equal to corresponding components of the w_i's in the nonoverlapping parts and to the average of them in the overlapped parts.

Radicati di Brozolo and Robert (1989) report on timing results obtained on an IBM 3090-600E/VF for GMRES preconditioned by overlapped incomplete LU

decomposition for a two-dimensional system of order 32,400 with a bandwidth of 360. For p processors ($1 \leq p \leq 6$) they subdivide A into p overlapping parts, the overlap being so large that these blocks cover all the nonzero entries of A. They found experimentally that an overlap of about 360 entries was optimal for their problem. This approach led to a speedup of roughly p. In some cases the parallel preconditioner was slightly more effective than the standard one, so that this method is also of interest for applications on serial computers.

Meurant (1989) reports on timing results obtained on a CRAY Y-MP/832, using an incomplete repeated twisted block factorization for two-dimensional problems. In his experiments, the $\widetilde{L}$ of the incomplete factorization has a block structure as shown below.

$$\begin{pmatrix}
X & & & & & & & & & & & & & & & & & & & & & & & \\
X & X & & & & & & & & & & & & & & & & & & & & & & \\
 & X & X & & & & & & & & & & & & & & & & & & & & & \\
 & & & X & X & & & & & & & & & & & & & & & & & & & \\
 & & & & X & X & & & & & & & & & & & & & & & & & & \\
 & & & & & X & & & & & & & & & & & & & & & & & & \\
 & & & & & & X & & & & & & & & & & & & & & & & & \\
 & & & & & & X & X & & & & & & & & & & & & & & & & \\
 & & & & & & & X & X & & & & & & & & & & & & & & & \\
 & & & & & & & & & X & X & & & & & & & & & & & & & \\
 & & & & & & & & & & X & X & & & & & & & & & & & & \\
 & & & & & & & & & & & X & & & & & & & & & & & & \\
 & & & & & & & & & & & & X & & & & & & & & & & & \\
 & & & & & & & & & & & & X & X & & & & & & & & & & \\
 & & & & & & & & & & & & & X & X & & & & & & & & & \\
 & & & & & & & & & & & & & & & X & X & & & & & & & \\
 & & & & & & & & & & & & & & & & X & X & & & & & & \\
 & & & & & & & & & & & & & & & & & & X & X & & & & \\
 & & & & & & & & & & & & & & & & & & & X & X & & & \\
 & & & & & & & & & & & & & & & & & & & & X & & &
\end{pmatrix}$$

Specifically, $\widetilde{L}$ has alternatingly a lower block diagonal, an upper one, a lower one, and (finally) an upper one. For this approach Meurant reports a speedup, for preconditioned CG, close to 6 on the 8-processor CRAY Y-MP. This speedup has been measured relative to the same repeated twisted factorization process executed on a single processor. Meurant also reports an increase in the number of iteration steps as a result of this repeated twisting. This increase implies that the effective speedup with respect to the non-parallel code is only about 4. In De Sturler (1994), Meurant's approach has been combined with the approach of Radicati and Robert.

For three-dimensional problems on a two-processor system we have used the blockwise twisted approach in the z-direction. That is, the (x, y)-planes in the grid were treated in parallel from the bottom and top inwards. Over each plane we used ordering by diagonals, in order to achieve high vector speeds on each processor.

On a CRAY X-MP/2 this led, for a preconditioned CG, to a reduction by a factor of close to 2 in wall clock time with respect to the CPU time for the non-parallel code on a single processor. For the microtasked code, the wall clock time on the 2-processor system was measured for a dedicated system, whereas for the non-parallel code the CPU time was measured on a moderately loaded system. In some situations we even observed a reduction in wall clock time by a factor of slightly more than two, because of the better convergence properties of the twisted incomplete preconditioner.

As suggested before, we can apply the twisted incomplete factorization in a nested way. For three-dimensional problems this can be exploited by twisting also the blocks corresponding to (x, y) planes in the y-direction. Over the resulting blocks, corresponding to half (x, y) planes, we may apply diagonal ordering in order to fully vectorize the four parallel parts. With this approach we have been able to reduce the wall clock time by a factor of 3.3, for a preconditioned CG, on the 4-processor Convex C-240. In this case, the total CPU time used by all of the processors is roughly equal to the CPU time required for single-processor execution. For more details on these and some other experiments, see van der Vorst (1990*b*).

As has been shown in Section 4.3, the hyperplane ordering can be used to realize long vector lengths in three-dimensional situations—at the expense, however, of indirect addressing. A similar approach has been followed by Berryman et al. (1990) for parallelizing standard ICCG on a Thinking Machines CM-2. For a 4K-processor machine they report a computational speed of 52.6 Mflop/s for the (sparse) matrix-vector product, while 13.1 Mflop/s has been realized for the preconditioner with the hyperplane approach. This reduction in speed by a factor of 4 makes it attractive to use only diagonal scaling as a preconditioner, in certain situations, for massively parallel machines like the CM-2. The latter approach has been followed by Mathur and Johnsson (1989) for finite-element problems.

We have used the hyperplane ordering for preconditioned CG on an Alliant FX/4, for three-dimensional systems with dimensions $n_x = 40$, $n_y = 39$, and $n_z = 30$. For 4 processors this led to a speedup of 2.61, compared with a speedup of 2.54 for the CG process with only diagonal scaling as a preconditioner. The fact that both speedups are far below the optimal value of 4 must be attributed to cache effects. These cache effects can be largely removed by using the reduced system approach suggested by Meier and Sameh (1988). However, for the three-dimensional systems that we have tested, the reduced system approach led, on average, to about the same CPU times as for the hyperplane approach on Alliant FX/8 and FX/80 computers.

Acknowledgment

We would like to thank Michele Benzi of Los Alamos and Nick Gould of the Rutherford Appleton Laboratory for helpful comments on an earlier draft.

References

Alleon, G., Benzi, M. and Giraud, L. (1997), 'Sparse approximate inverse preconditioning for dense linear systems arising in computational electromagnetics', *Numerical Algorithms* **16**(1), 1–15.

Alvarado, F. and Dağ, H. (1994), Incomplete partitioned inverse preconditioners, Technical report, Department of Electrical and Computer Engineering, University of Wisconsin, Madison.

Alvarado, F. and Schreiber, R. (1993), 'Optimal parallel solution of sparse triangular systems', *SIAM J. Scientific Computing* **14**, 446–460.

Arioli, M., Duff, I. S., Noailles, J. and Ruiz, D. (1992), 'A block projection method for sparse matrices', *SIAM J. Scientific and Statistical Computing* **13**, 47–70.

Ashby, S. F. (1991), 'Minimax polynomial preconditioning for Hermitian linear systems', *SIAM J. Matrix Analysis and Applications* **12**, 766–789.

Ashcraft, C. and Grimes, R. (1988), 'On vectorizing incomplete factorizations and SSOR preconditioners', *SIAM J. Scientific and Statistical Computing* **9**, 122–151.

Axelsson, O. (1994), *Iterative Solution Methods*, Cambridge University Press, Cambridge.

Axelsson, O. and Barker, V. (1984), *Finite Element Solution of Boundary Value Problems. Theory and Computation*, Academic Press, New York, NY.

Axelsson, O. and Lindskog, G. (1986), 'On the eigenvalue distribution of a class of preconditioning methods', *Numerische Mathematik* **48**, 479–498.

Axelsson, O. and Munksgaard, N. (1983), Analysis of incomplete factorizations with fixed storage allocation, *in* D. Evans, ed., 'Preconditioning Methods - Theory and Applications', Gordon and Breach, New York, pp. 265–293.

Babuska, I. (1972), 'Numerical stability in problems of linear algebra', *SIAM J. Numerical Analysis* **9**, 53–77.

Barnard, S. T. and Clay, R. L. (1997), A portable MPI implementation of the SPAI preconditioner in ISIS++, *in* M. Heath, V. Torczon, G. Astfalk, P. E. Björstad, A. H. Karp, C. H. Koebel, V. Kumar, R. F. Lucas, L. T. Watson and D. E. Womble, eds, 'Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing', SIAM Press, pp. xxx–yyy.

Barnard, S. T., Bernardo, L. M. and Simon, H. D. (1997), An MPI implementation of the SPAI preconditioner on the T3E, Technical Report LBNL-40794 UC405, Lawrence Berkeley National Laboratory.

Benzi, M. and Tůma, M. (1998*a*), 'Numerical experiments with two sparse approximate inverse preconditioners', *BIT* **38**, 234–241.

Benzi, M. and Tůma, M. (1998*b*), 'A sparse approximate inverse preconditioner for nonsymmetric linear systems', *SIAM J. Scientific Computing* **19**(3), 968–994.

Benzi, M., Meyer, C. D. and Tůma, M. (1996), 'A sparse approximate inverse preconditioner for the conjugate gradient method', *SIAM J. Scientific Computing* **17**, 1135–1149.

Benzi, M., Szyld, D. B. and van Duin, A. C. N. (1997), Orderings for incomplete factorization preconditioning of nonsymmetric problems, Technical Report 97-91, Temple University, Department of Mathematics, Philadelphia, PA.

Berryman, H., Saltz, J., Gropp, W. and Mirchandaney, R. (1990), 'Krylov methods preconditioned with incompletely factored matrices on the CM-2', *J. Par. Dist. Comp.* **8**, 186–190.

Bramley, R. and Sameh, A. (1992), 'Row projection methods for large nonsymmetric linear systems', *SIAM J. Scientific and Statistical Computing* **13**, 168–193.

Chan, T. F. (1991), 'Fourier analysis of relaxed incomplete factorization procedures', *SIAM J. Scientific and Statistical Computing* **12**, 668–680.

Chan, T. F. and Goovaerts, D. (1990), 'A note on the efficiency of domain decomposed incomplete factorizations', *SIAM J. Scientific and Statistical Computing* **11**, 794–803.

Chan, T. F. and van der Vorst, H. A. (1997), Approximate and incomplete factorizations, *in* D. Keyes, A. Sameh and V. Venkatakrishnan, eds, 'Parallel Numerical Algorithms', ICASE/LaRC Interdisciplinary Series in Science and Engineering, Kluwer, Dordrecht, pp. 167–202.

Choi, H. and Szyld, D. (1996), Threshold ordering for preconditioning nonsymmetric problems with highly varying coefficients, Technical Report 96-51, Department of Mathematics, Temple University, Philadelphia.

Chow, E. and Saad, Y. (1994), Approximate inverse preconditioners for general sparse matrices, Technical Report Research Report UMSI 94/101, University of Minnesota Supercomputing Institute, Minneapolis, Minnesota.

Concus, P. and Meurant, G. (1986), 'On computing INV block preconditionings for the conjugate gradient method', *BIT* pp. 493–504.

Concus, P., Golub, G. H. and Meurant, G. (1985), 'Block preconditioning for the conjugate gradient method', *SIAM J. Scientific and Statistical Computing* **6**, 220–252.

Cosgrove, J. D. F., Diaz, J. C. and Griewank, A. (1992), 'Approximate inverse preconditionings for sparse linear systems', *Int J. Computer Mathematics* **44**, 91–110.

Daydé, M. J., L'Excellent, J.-Y. and Gould, N. I. M. (1997), 'Element-by-element preconditioners for large partially separable optimization problems', *SIAM J. Scientific Computing* **18**(6), 1767–1787.

D'Azevedo, E. F., Forsyth, P. A. and Tang, W. P. (1992), 'Drop tolerance preconditioning for incompressible viscous flow', *Int J. Computer Mathematics* **44**, 301–312.

De Sturler, E. (1994), Iterative methods on distributed memory computers, PhD thesis, Delft University of Technology, Delft, the Netherlands.

Doi, S. (1991), 'On parallelism and convergence of incomplete LU factorizations', *Applied Numerical Math.* **7**, 417–436.

Doi, S. and Hoshi, A. (1992), 'Large numbered multicolor MILU preconditioning on SX-3/14', *Int J. Computer Mathematics* **44**, 143–152.

Dongarra, J. J., Bunch, J., Moler, C. and Stewart, G. (1979), *LINPACK Users' Guide*, SIAM Pub., Philadelphia.

Dubois, P. F., Greenbaum, A. and Rodrigue, G. H. (1979), 'Approximating the inverse of a matrix for use in iterative algorithms on vector processors', *Computing* **22**, 257–268.

Duff, I. S. and Meurant, G. A. (1989), 'The effect of ordering on preconditioned conjugate gradient', *BIT* **29**, 635–657.

Duff, I. S., Erisman, A. M., Gear, C. W. and Reid, J. K. (1988), 'Sparsity structure and Gaussian elimination', *SIGNUM Newsletter* **23**(2), 2–8.

Dupont, T., Kendall, R. P. and Rachford Jr., H. H. (1968), 'An approximate factorization procedure for solving self-adjoint elliptic difference equations', *SIAM J. Numerical Analysis* **5(3)**, 559–573.

Dutto, L. (1993), 'The effect of ordering on preconditioned GMRES algorithm for solving the Navier-Stokes equations', *Int J. Numerical Methods in Engineering* **36**, 457–497.

Eijkhout, V. (1991), 'Analysis of parallel incomplete point factorizations', *Linear Algebra and its Applications* **154-156**, 723–740.

Eijkhout, V. (1992), Beware of unperturbed modified incomplete point factorizations, *in* R. Beauwens and P. de Groen, eds, 'Iterative Methods in Linear Algebra', IMACS Int. Symp., Brussels, Belgium, 2-4 April, 1991, North-Holland, Amsterdam, pp. 583–591.

Eisenstat, S. C. (1981), 'Efficient implementation of a class of preconditioned conjugate gradient methods', *SIAM J. Scientific and Statistical Computing* **2(1)**, 1–4.

Elman, H. C. (1989), 'Relaxed and stabilized incomplete factorizations for non-self-adjoint linear systems', *BIT* **29**, 890–915.

Forsythe, G. E. and Strauss, E. G. (1955), 'On best conditioned matrices', *Proc.Amer.Math.Soc.* **6**, 340–345.

Golub, G. H. and Van Loan, C. F. (1996), *Matrix Computations. Third Edition*, The Johns Hopkins University Press, Baltimore.

Gould, N. I. M. and Scott, J. A. (1998), 'Sparse approximate-inverse preconditioners using norm-minimization techniques', *SIAM J. Scientific Computing* **19**(2), 605–625.

Grote, M. and Huckle, T. (1997), 'Parallel preconditionings with sparse approximate inverses', *SIAM J. Scientific Computing* **18**, 838–853.

Grote, M. and Simon, H. (1993), Parallel preconditioning and approximate inverses on the connection machine, *in* R. Sincovec, D. Keyes, M. Leuze, L. Petzold and D. Reed, eds, 'Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing', SIAM, Philadelphia, pp. 519–523.

Gustafsson, I. (1978), 'A class of first order factorization methods', *BIT* **18**, 142–156.

Gustafsson, I. and Lindskog, G. (1986), 'A preconditioning technique based on element matrix factorizations', *Comput. Methods Appl. Mech. Eng.* **55**, 201–220.

Gustafsson, I. and Lindskog, G. (1995), 'Completely parallelizable preconditioning methods', *Numerical Linear Algebra with Applications* **2**, 447–465.

Hageman, L. A. and Young, D. M. (1981), *Applied Iterative Methods*, Academic Press, New York.

Hayami, K. and Harada, N. (1985), The scaled conjugate gradient method on vector processors, *in* K. P. Kartashev and I. S. Kartashev, eds, 'Supercomputing Systems', St. Petersburg, FL.

Heroux, M., Vu, P. and Yang, C. (1991), 'A parallel preconditioned conjugate gradient package for solving sparse linear systems on a CRAY Y-MP', *Applied Numerical Math.* **8**, 93–115.

Hughes, T., Levit, I. and Winget, J. (1983), 'An element-by-element solution algorithm for problems of structural and solid mechanics', *J. Comp. Methods in Appl. Mech. Eng.* **36**, 241–254.

Johnson, O. G., Micchelli, C. A. and Paul, G. (1983), 'Polynomial preconditioning for conjugate gradient calculations', *SIAM J. Numerical Analysis* **20**, 363–376.

Jones, M. T. and Plassmann, P. E. (1994), The efficient parallel iterative solution of large sparse linear systems, *in* A. George, J. Gilbert and J. Liu, eds, 'Graph Theory and Sparse Matrix Computations', IMA Vol 56, Springer Verlag, Berlin.

Kettler, R. (1987), Linear multigrid methods in numerical reservoir simulation, PhD thesis, Delft University of Technology, Delft.

Kolotilina, L. Y. and Yeremin, A. Y. (1993), 'Factorized sparse approximate inverse preconditionings', *SIAM J. Matrix Analysis and Applications* **14**, 45–58.

Kuo, J. C. C. and Chan, T. F. (1990), 'Two-color fourier analysis of iterative algorithms for elliptic problems with red/black ordering', *SIAM J. Scientific and Statistical Computing* **11**, 767–793.

Manteuffel, T. A. (1980), 'An incomplete factorization technique for positive definite linear systems', *Mathematics of Computation* **31**, 473–497.

Mathur, K. K. and Johnsson, S. L. (1989), 'The finite element method on a data parallel computing system', *Int J. High Speed Computing* **1**(1), 29–44.

Meier, U. and Sameh, A. (1988), The behavior of conjugate gradient algorithms on a multivector processor with a hierarchical memory, Technical Report CSRD 758, University of Illinois, Urbana, IL.

Meijerink, J. A. (1983), Iterative methods for the solution of linear equations based on incomplete factorisations of the matrix, Technical Report Shell Publication 643, KSEPL, Rijswijk.

Meijerink, J. A. and van der Vorst, H. A. (1977), 'An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix', *Mathematics of Computation* **31**, 148–162.

Meijerink, J. A. and van der Vorst, H. A. (1981), 'Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems', *J. Comp. Phys.* **44**, 134–155.

Meurant, G. (1984*a*), 'The block preconditioned conjugate gradient method on vector computers', *BIT* **24**, 623–633.

Meurant, G. (1984*b*), Numerical experiments for the preconditioned conjugate gradient method on the CRAY X-MP/2, Technical Report LBL-18023, University of California, Berkeley, CA.

Meurant, G. (1989), The conjugate gradient method on vector and parallel supercomputers, Technical Report CTAC-89, University of Brisbane.

Munksgaard, N. (1980), 'Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method', *ACM Trans. Math. Softw.* **6**, 206–219.

Notay, Y. (1994), 'DRIC: a dynamic version of the RIC method', *Numerical Linear Algebra with Applications* **1**, 511–532.

Østerby, O. and Zlatev, Z. (1983), *Direct methods for sparse matrices*, number 157 *in* 'Lecture Notes in Computer Science', Springer Verlag, Berlin, Heidelberg, New York.

Radicati di Brozolo, G. and Robert, Y. (1989), 'Parallel conjugate gradient-like algorithms for solving sparse non-symmetric systems on a vector multiprocessor', *Parallel Computing* **11**, 223–239.

Radicati di Brozolo, G. and Vitaletti, M. (1986), Sparse matrix-vector product and storage representations on the IBM 3090 with Vector Facility, Technical Report 513-4098, IBM-ECSEC, Rome.

Saad, Y. (1985), 'Practical use of polynomial preconditionings for the conjugate gradient method', *SIAM J. Scientific and Statistical Computing* **6**, 865–881.

Saad, Y. (1994), 'ILUT: A dual threshold incomplete LU factorization', *Numerical Linear Algebra with Applications* **1**, 387–402.

Schlichting, J. J. F. M. and van der Vorst, H. A. (1989), 'Solving 3D block bidiagonal linear systems on vector computers', *J. Comput. Appl. Math.* **27**, 323–330.

Seager, M. K. (1986), 'Parallelizing conjugate gradient for the CRAY X-MP', *Parallel Computing* **3**, 35–47.

Sleijpen, G. L. G. and van der Vorst, H. A. (1995), 'Maintaining convergence properties of BICGSTAB methods in finite precision arithmetic', *Numerical Algorithms* **10**, 203–223.

Stone, H. S. (1968), 'Iterative solution of implicit approximations of multidimensional partial differential equations', *SIAM J. Numerical Analysis* **5**, 530–558.

Tan, K. H. (1995), Local coupling in domain decomposition, PhD thesis, Utrecht University, Utrecht, the Netherlands.

van der Sluis, A. (1969), 'Condition numbers and equilibration of matrices', *Numerische Mathematik* **14**, 14–23.

van der Vorst, H. A. (1981), 'Iterative solution methods for certain sparse linear systems with a non-symmetric matrix arising from PDE-problems', *J. Comp. Phys.* **44**, 1–19.

van der Vorst, H. A. (1982*a*), Preconditioning by Incomplete Decompositions, PhD thesis, Utrecht University, Utrecht, The Netherlands.

van der Vorst, H. A. (1982*b*), 'A vectorizable variant of some ICCG methods', *SIAM J. Scientific and Statistical Computing* **3**, 86–92.

van der Vorst, H. A. (1987), 'Large tridiagonal and block tridiagonal linear systems on vector and parallel computers', *Parallel Computing* **5**, 45–54.

van der Vorst, H. A. (1989*a*), 'High performance preconditioning', *SIAM J. Scientific and Statistical Computing* **10**, 1174–1185.

van der Vorst, H. A. (1989*b*), 'ICCG and related methods for 3D problems on vector computers', *Computer Physics Communications* **53**, 223–235.

van der Vorst, H. A. (1990*a*), The convergence behaviour of preconditioned CG and CG-S in the presence of rounding errors, *in* O. Axelsson and L. Y. Kolotilina, eds, 'Preconditioned Conjugate Gradient Methods', Nijmegen 1989, Springer Verlag, Berlin. Lecture Notes in Mathematics 1457.

van der Vorst, H. A. (1990*b*), 'Experiences with parallel vector computers for sparse linear systems', *Supercomputer* **37**, 28–35.

van Gijzen, M. B. (1994), Iterative solution methods for linear equations in finite element computations, PhD thesis, Delft University of Technology, Delft, the Netherlands.

Varga, R. S. (1960), Factorizations and normalized iterative methods, *in* R. E. Langer, ed., 'Boundary Problems in differential equations', University of Wisconsin Press, Madison, WI, pp. 121–142.

Washio, T. and Hayami, K. (1994), 'Parallel block preconditioning based on SSOR and MILU', *Numerical Linear Algebra with Applications* **1**, 533–553.

Watts-III, J. W. (1981), 'A conjugate gradient-truncated direct method for the iterative solution of the reservoir simulation pressure equation', *Society of Petroleum Engineers J.* **21**, 345–353.

Young, D. P., Melvin, R. G., Johnson, F. T., Bussoletti, J. E., Wigton, L. B. and Samanth, S. S. (1989), 'Application of sparse matrix solvers as effective preconditioners', *SIAM J. Scientific and Statistical Computing* **10**(6), 1186–1199.

Zlatev, Z. (1991), *Computational methods for general sparse matrices*, Kluwer Acad. Pub., Dordrecht, Boston, London.