Chapter 10

Implementation Aspects

For effective use of the given iteration schemes, it is necessary that they can be implemented such that high computing speeds are achievable. It is most likely that high computing speeds will be realized only by parallel architectures and therefore we must see how well iterative methods fit to such computers.

The iterative methods only need a handful of basic operations per iteration step

- Vector updates: in each iteration step the current approximation to the solution is updated by a correction vector. Often the corresponding residual vector is also obtained by a simple update, and we have update formulas as well for the correction vector (or search direction).

- Inner products: In many methods the speed of convergence is influenced by carefully constructed iteration coefficients. These coefficients are sometimes known analytically, but more often they are computed by inner products, involving residual vectors and search directions, as in the methods discussed in the previous sections.

- Matrix vector products: In each step at least one matrix vector product has to be computed with the matrix of the given linear system. Sometimes also matrix vector products with the transpose of the given matrix are required (e.g., BiCG). Note that it is not necessary to have the matrix explicitly, it suffices to be able to generate the result of the matrix vector product.

- Preconditioning: It is common practice to precondition the given linear system by some preconditioning operator. Again it is not necessary to have this operator in explicit form, it is enough to generate the result of the operator applied to some given vector. The preconditioner is applied as often as the matrix vector multiply in each iteration step.

For problem sizes large enough the inner products, vector updates and matrix vector product are easily parallelized and vectorized. The more successful preconditionings, i.e, based upon incomplete LU decomposition, are not easily parallelizable. For that reason one is often satisfied with the use of only diagonal scaling as a preconditioner on highly parallel computers, such as the CM2 [24].

On distributed memory computers we need large grained parallelism in order to reduce synchronization overhead. This can be achieved by combining the work required for a successive number of iteration steps. The idea is to construct first in parallel a straight forward Krylov basis for the search subspace in which an update for the current solution will be determined. Once this basis has been computed, the vectors are orthogonalized, as is done in Krylov subspace methods. The construction as well as the orthogonalization can be done with large grained parallelism, and has sufficient degree of parallelism in it.
This approach has been suggested for CG in [36] and for GMRES in [37], [14] and [48]. One of the disadvantages in this approach is that a straight forward basis, of the form $y, Ay, A^2y, ..., A^iy$ is usually very ill-conditioned. This is in sharp contrast to the optimal condition of the orthogonal basis set constructed by most of the projection type methods and it puts severe limits on the number of steps that can be combined. However, in [14] and [48] ways to improve the condition of a parallel generated basis are suggested and it seems possible to take larger numbers of steps, say 25, together. In [48] the effects of this approach on the communication overhead are studied and compared with experiments done on moderately massive parallel transputer systems.

Chapter 11

Solution of singular systems

11.1 Only nonzero eigenvalues matter

11.2 Pure Neumann problems

Singular consistent linear systems arise after discretisation of, for instance, a partial differential equation like

$$-u_{xx} - u_{yy} = f(x, y), \tag{11.1}$$

over a domain Γ with Neumann boundary condition

$$\frac{\partial u}{\partial n} = 0,$$

along the entire boundary $\delta\Gamma$. This means that the solution of (11.1) is determined up to a constant and this is, after proper discretisation, reflected by singularity of the matrix A in the discretised system $Ax = b$. The vector with all one's is then an eigenvector of A with eigenvalue 0. If the linear system $Ax = b$ is consistent, in this case if b is orthogonal to the eigenvector corresponding to the zero eigenvalue, then we may solve the system with CG without much problem.

However, it might seem a good idea to fix the solution u at one boundary point and this leads to a non-singular problem and one might expect to be better off. That this is not always a good idea is shown by the next example.

A numerical example Consider the problem (11.1) over the unit square with an equidistant rectangular grid with mesh spacing $h = 1/19$. This leads, with the usual 5-point finite volume discretization (see [185, section

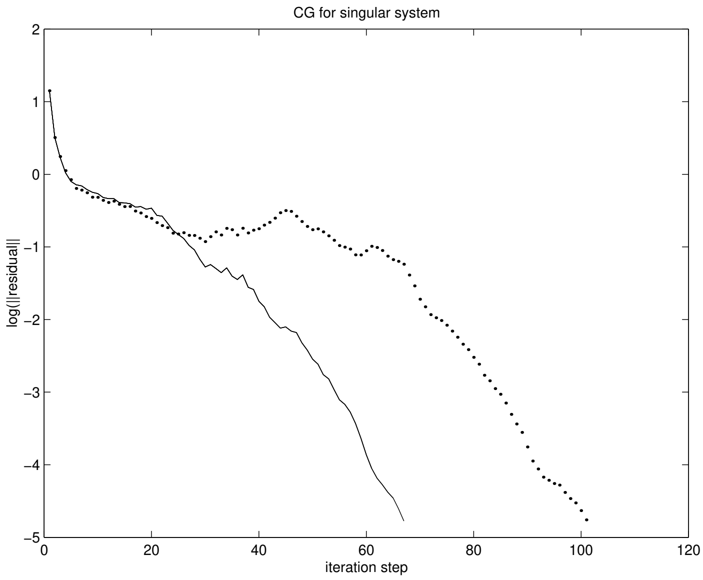

Figure 11.1: CG for a singular (—) and a near-singular system (...)

6.2]), and after multiplication with h, to a linear system $A_0x = b$. We have used the index 0 in order to underline the singularity of the system. The stencil for A_0 for an interior point in the grid is

$$\begin{array}{ccc} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{array} \quad ,$$

while for points on the boundary the stencil has no connections pointing outward of Γ and the central element 4 is then reduced by 1 for each missing connection. The matrix A_0 is of order 400 and has a block structure with blocks of dimension 20.

For our experiment we need a consistent system and we construct a meaningful right-hand side b as follows. We take the vector w with elements $w_j = \sin(j)$ and we eliminate the component in the direction of the vector with all one's. The resulting vector b is then orthogonal to the vector with alle one's (it is easy to see that that vector is the eigenvector of A_0 with eigenvalue 0).

The system can be solved with CG, because the effective eigenvalues are all positive. The condition number defined by these positive eigenvalues is $\kappa_{effect} \approx 322.9$. In Figure 11.1 we see the convergence history in terms of the 2-norms of the residuals, represented by the drawn line.

Now we fix the singularity, by prescribing one value of u at the boundary $\delta\Gamma$: in our experiment we set $b_{60} = 1$ and we set all elements of A_0, except the diagonal element, in the 60-th row and 60-th column to zero. The element $A_0(60, 60) = 1$. It can be shown that the resulting system $A_1 x = \bar{b}$ is nonsingular (using the fact that A_1 is irreducibly diagonally dominant and [185, Theorem 1.8]). The convergence behaviour of CG for this system $A_1 x = \bar{b}$ is represented by the dots in Figure 11.1. Note that we have a noticeably slower convergence now, which is explained by the condition number of A_1, which is $\kappa_2(A_1) \approx 4183.6$. Fixing the nonsingularity of A_0 has resulted in this case to shifting the eigenvalue 0 to a small nonzero eigenvalue of A_1 that is about 13 times smaller than the smallest nonzero eigenvalue of A_0.

Chapter 12

Solution of $f(A)x = b$ with Krylov subspace information

12.1 Introduction

In this chapter, we expand on an idea for exploiting Krylov subspace information obtained for the matrix A and the vector b. This subspace information can be used for the approximate solution of a linear system $f(A)x = b$, where f is some analytic function. We will make suggestions on how to use this for the case where f is the matrix *sign* function. The matrix *sign* function plays an important role in QCD computations, see for instance [131]. In the computational models one has to compute an approximate solution for linear systems of the type

$$(B + \text{sign}(A))x = b, \tag{12.1}$$

with $A, B \in \mathbb{C}^{n \times n}$, and A and B do not commute. The latter property is an important bottleneck for the efficient computation of subspaces that can be used for the reduction of both A and B.

In [172] an approach was suggested for the usage of a Krylov subspace for the computation of approximate solutions of linear systems

$$f(A)x = b.$$

The approach was motivated by the function $f(A) = A^2$, which plays a role in the solution of some biharmonic systems. The approach is easily generalized for non-symmetric complex matrices, but we may have to pay more attention to the evaluation of f for the reduced system, associated with the Krylov subspace.

In particular, we will discuss some possible approaches for using the Krylov subspace for the computation of $\mathrm{sign}(A)p$ for given vectors p. With the evaluation of the matrix *sign* function one has to be extremely careful. A popular approach, based on a Newton iteration converges fast, but is sensitive for rounding errors, especially when A is ill-conditioned. We will briefly discuss a computational method that was suggested (and analyzed) by Bai and Demmel [13]. This approach can also be combined, in principle, with the subspace reduction technique. Our early experiments, not reported here, indicate that the actual computation of approximate solutions of (12.1) is complicated because of the occurrence of the matrix B. Since this matrix does not share an eigenvector basis with A, there is little hope that the subspace generated with A can also be used efficiently with B. It seems that we have to experiment with either nested approaches or with mixed subspaces. The current state of the art is that we still have a way to go in our quest for an efficient computational technique for the very large systems that arise in QCD modeling.

12.2 Reduced Systems

With equation (2.16) we can construct approximate solutions for $Ax = b$ in the Krylov subspace $K^m(A; r_0)$. These approximate solutions can be written as $x_m = x_0 + V_m y$, with $y \in \mathbb{R}^n$, since the columns of V_m span a basis for the Krylov subspace. The Ritz-Galerkin orthogonality condition for the residual leads to

$$b - Ax_m \perp \{v_1, \ldots, v_m\},$$

or

$$V_m^H \left(b - A(x_0 + V_m y)\right) = 0.$$

Now we use that $b - Ax_0 = r_0 = \|r_0\|_2 v_1$, and with (2.16) we obtain

$$H_{m,m} y = \|r_0\| e_1, \tag{12.2}$$

with e_1 the first canonical basis vector in $\mathbb{R}^m$. If $H_{m,m}$ is not singular then we can write the approximate solution x_m as

$$x_m = \|r_0\|_2 V_m H_{m,m}^{-1} e_1. \tag{12.3}$$

Note that this expression closely resembles the expression $x = A^{-1}b$ for the exact solution of $Ax = b$. The matrix $H_{m,m}$ can be interpreted as the restriction of A with respect to $v_1, \ldots, v_m$. The vector $\|r_0\|e_1$ is the expression for the right-hand side with respect to this basis, and V_m is the operator that expresses the solution of the reduced system (in $\mathbb{R}^m$) in terms of the canonical basis for $\mathbb{R}^n$.

Let us from now on assume that $x_0 = 0$.

We can also use the above mechanism for the solution of more complicated systems of equations. Suppose that we want to find approximate solutions for $A^2x = b$, with only the Krylov subspace for A and $r_0 = b$ available. The solution of $A^2x = b$ can be realized in two steps

1. Solve z_m from $Az = b$, using the Ritz-Galerkin condition. With $z = V_m y$ and (2.16), we have that

$$z = \|b\|_2 V_m H_{m,m}^{-1} e_1.$$

2. Solve x_m from $Ax_m = z_m$, with $x_m = V_m u$. It follows that

$$AV_m u = \|b\|_2 V_m H_{m,m}^{-1} e_1,$$

$$V_{m+1} H_{m+1,m} u_m = \|b\|_2 V_m h_{m,m}^{-1}.$$

   The Ritz-Galerkin condition with respect to V_m leads to

$$H_{m,m} u_m = \|b\|_2 H_{m,m}^{-1} e_1.$$

These two steps lead to the approximate solution

$$x_m = \|b\|_2 V_m H_{m,m}^{-2} e_1. \tag{12.4}$$

If we compare (12.3) for $Ax = b$ with (12.4) for $A^2x = b$, then we see that the operation with A^2 translates to an operation with $H_{m,m}^2$ for the reduced system and that is all.

Note that this approximate solution x_m does not satisfy a Ritz-Galerkin condition for the system $A^2x = b$. Indeed, for $x_m = V_m y$, we have that

$$A^2 V_m y = AV_{m+1} H_{m+1,m} y = V_{m+2} H_{m+2,m+1} H_{m+1,m} y.$$

The Ritz-Galerkin condition with respect to V_m, for $b - Ax_m$, leads to

$$V_m^H V_{m+2} H_{m+2,m+1} H_{m+1,m} y = \|b\|_2 e_1.$$

A straight-forward evaluation of $H_{m+2,m+1} H_{m+1,m}$ and the orthogonality of the v_j's, leads to

$$V_m^H V_{m+2} H_{m+2,m+1} H_{m+1,m} = H_{m,m}^2 + h_{m+1,m} h_{m,m+1} e_m e_m^T.$$

That means that the reduced matrix for A^2, expressed with respect to the V_m basis, is given by the matrix $H_{m,m}^2$ in which the bottom right element $h_{m,m}$ is updated with $h_{m+1,m}h_{m,m+1}$. By computing x_m as in (12.4), we have ignored the factor $h_{m+1,m}h_{m,m+1}$. This is acceptable, since in generic situations the convergence of the Krylov solution process for $Ax = b$ goes hand in hand with small elements $h_{m+1,m}$.

We can go one step further, and try to solve

$$(A^2 + \alpha A + \beta I)x = b,$$

with Krylov subspace information obtained for $Ax = b$ (with $x_0 = 0$). The Krylov subspace $K^m(A, r_0)$ is shift invariant, that is

$$K^m(A, r_0) = K^m(A - \sigma I, r_0),$$

for any scalar $\sigma \in \mathbb{C}$. The matrix polynomial $A^2 + \alpha A + \beta I$ can be factored into

$$A^2 + \alpha A + \beta I = (A - \omega_1 I)(A - \omega_2 I).$$

Proceeding as for A^2, that is solving the given system in two steps, and imposing a Ritz-Galerkin condition for each step, leads to the approximate solution

$$x_m = \|b\|_2 V_m (H_{m,m}^2 + \alpha H_{m,m} + \beta I_m)^{-1} e_1.$$

The generalization to higher degree polynomial systems

$$p_n(A)x \equiv \left(A^n + \alpha_{n-1}A^{n-1} + \ldots + \alpha_0 I\right) x = b$$

is straight forward and leads to an approximate solution of the form

$$x_m = \|b\|_2 V_m p_n(H_{m,m})^{-1} e_1.$$

If f is an analytic function, then we can compute the following approximate solution x_m for the solution of $f(A)x = b$:

$$x_m = \|b\|_2 V_m f(H_{m,m})^{-1} e_1. \tag{12.5}$$

All these approximations are equal to the exact solution if $h_{m+1,m} = 0$. Because $h_{n+1,n} = 0$, we have the exact solution after at most n iterations. The hope is, of course, that the approximate solutions are sufficiently good after $m \ll n$ iterations. There is little to control the residual for the approximate solutions, since in general $f(A)$ may be an expensive function. We use the Krylov subspace reduction in order to avoid expensive evaluation of $f(A)p$ for $p \in \mathbb{R}^n$. A possibility is to compare successive approximations x_m and to base a stopping criterion on this comparison.

12.3 Computation of the inverse of $f(H_{m,m})$

The obvious way of computing $f(H_{m,m})^{-1}$ is to reduce $H_{m,m}$ first to some convenient canonical form, for instance to diagonal form. If $H_{m,m}$ is symmetric (in that case $H_{m,m}$ is tridiagonal) then it can be orthogonally transformed to diagonal form:

$$Q_m^H H_{m,m} Q_m = D_m,$$

with Q_m an m by m orthogonal matrix and D_m a diagonal matrix. We then have that

$$Q^H f(H_{m,m})^{-1} Q = f(D_m)^{-1},$$

and this can be used for an efficient and stable computation of x_m. If $H_{m,m}$ is neither symmetric nor (close to) normal (that is $H_{m,m}^H H_{m,m} = H_{m,m} H_{m,m}^H$, then the transformation to diagonal form cannot be done by an orthogonal operator. If $H_{m,m}$ has no Jordan blocks, the transformation can be done by

$$X_m^{-1} H_{m,m} X_m = D_m.$$

This decomposition is not advisable if the condition number of X_m is much larger than 1. In that case it is much better to reduce the matrix $H_{m,m}$ to Schur form:

$$Q_m^H H_{m,m} Q_m = U_m,$$

with U_m an upper triangular matrix. The eigenvalues of $H_{m,m}$ appear along the diagonal of U_m. If A is real, then the computations can be kept in real arithmetic if we use the property that $H_{m,m}$ can be orthogonally transformed to generalized Schur form. In a generalized Schur form, the matrix U_m may have two by two non-zero blocks along the diagonal (but its strict lower triangular part is otherwise zero). These two by two blocks represent complex conjugate eigenpairs of $H_{m,m}$. For further details on Schur forms, generalized Schur forms, and their computation, see [90].

12.4 Numerical examples

Our numerical examples have been taken from [172]. These experiments have been carried out for diagonal real matrices A, which does not mean a loss of generality. In exact arithmetic the Krylov subspace generated with A, and v, coincides with the Krylov subspace generated with $Q^H AQ$ and $Q^H v$, in the sense that

$$K^m(A; v) = QK^m(Q^H AQ, Q^H v),$$

if Q is an orthogonal matrix ($Q^HQ = I$). In other words, transformation with an orthogonal Q leads to another orientation of the orthogonal basis, but the Krylov subspace method leads to the same approximate solutions. Of course, round-off patterns may be different, but round-off does not have dramatic effects on Krylov subspace methods other than an occasional small delay in the number of iterations. Therefore, we may expect similar behavior for more general matrices with the same eigenvalue distribution.

m	$\|b - A^2x_m^{new}\|_2$	$\|b - A^2x_m^{old}\|_2$
0	$0.21E2$	$0.21E2$
10	0.18	0.15
20	$0.27E-2$	$0.16E-1$
30	$0.53E-5$	$0.63E-2$
40	$0.16E-8$	$0.28E-2$
50		$0.36E-2$
60		$0.10E-2$
70		$0.49E-4$
80		$0.18e-5$
100		$0.21e-8$

Table 12.1: Residual norms for approaches A and B

The diagonal matrix A is of order 900. Its eigenvalues are 0.034, 0.082, 0.127, 0.155, 0.190. The remaining 895 eigenvalues are uniformly distributed over the interval $[0.2, 1.2]$. This type of eigenvalue distribution is more or less what one might get with preconditioned Poisson operators. Now suppose that we want to solve $A^2x = b$, with b a vector with all ones. We list the results for two different approaches:

A We generate the Krylov subspace with A and b and determine the approximate solution x_m^{new} as in (12.4).

B We generate the Krylov subspace for the operator A^2 and the vector b (the 'classical' approach). This leads to approximations denoted as x_m^{old}.

In Table 12.1 we have listed the norms of the residuals for the two approaches for some values of m. The analysis in [172] shows that the much faster convergence for the new approach could have been expected. Note that the new approach also has the advantage that there are only m matrix vector products with A for the new approach. For the classical approach we need $2m$ matrix vector products with A, assuming that vectors like A^2p are computed by applying A twice. Usually, the matrix vector product is the CPU-dominating factor in the computations, since they operate in $\mathbb{R}^n$. The oprations with $H_{m,m}$ are carried out in $\mathbb{R}^m$, and in typical applications $m \ll n$.

In [172] also an example is given for a more complicated function of A, namely the solution of

$$e^A x = b,$$

with A the same diagonal matrix as in the previous example, and b again the vector with all ones. This is a type of problem that one encounters in the solution of linear systems of ODEs. With the Krylov subspace for A and r_0 of dimension 20, a residual

$$||r_m\|_2 \equiv \|b - e^A x_m\|_2 \approx 8.7E - 12$$

was observed, working in 48 bits floating point precision.

Others have also suggested to work with the reduced system for the computation of, for instance, the exp function of a matrix, as part of solution schemes for (parabolic) systems of equations. See, e.g. [108, 83, 123].

12.5 Matrix sign function

The matrix sign function $sign(A)$ for a nonsingular matrix A, with no eigenvalues on the imaginary axis, is defined as follows [13, 146]. Let

$$A = X\,\mathrm{diag}(J_+, J_-)X^{-1}$$

denote the decomposition of $A \in \mathbb{C}^{n\times n}$. The eigenvalues of J_+ lie in the right half plane, and those of J_- are in the left half plane. Let I_+ denote the identity matrix with the same dimensions as J_+, and I_- the identity matrix corresponding to J_-. Then

$$\mathrm{sign}(A) \equiv X\,\mathrm{diag}(I_+, -I_-)X^{-1}.$$

The sign function can be used, amongst others, to compute invariant subspaces, for instance those corresponding to the eigenvalues of A with positive real parts. It plays also an important role in QCD (cf [131]). The Jordan decomposition of A is not a useful vehicle for the computation of this function.

It can be shown that $\mathrm{sign}(A)$ is the limit of the Newton iteration

$$A_{k+1} = \frac{1}{2}(A_k + A_k^{-1}), \quad \text{for} \quad k = 0, 1, \ldots, \quad \text{with} \quad A_0 = A.$$

see [13]. Unfortunately, the Newton iteration is also not suitable for accurate computation if A is ill-conditioned. Bai and Demmel consider more accurate ways of computation, which rely on the (block) Schur form of A:

$$B = Q^H A Q = \begin{bmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{bmatrix}.$$

The matrix Q is orthonormal, and it can easily be shown that

$$\mathrm{sign}(A) = Q\,\mathrm{sign}(B) Q^H.$$

Let this decomposition be such that B_{11} contains the eigenvalues of A with positive real part. Then let R be the solution of the Sylvester equation

$$B_{11}R - RB_{22} = -B_{12}.$$

This Sylvester equation can also be solved by a Newton iteration process. Then Bai and Demmel proved that

$$\mathrm{sign}(A) = Q \begin{bmatrix} I & -2R \\ 0 & -I \end{bmatrix} Q^H.$$

See [13] for further details, stability analysis, and examples of actual computations.

Lippert [1] has experimented with alternative schemes for the computation of $\mathrm{sign}(A)$, in particular the Schultz iteration

$$A_{k+1} = \frac{1}{2} A_k (3I - A_k^2).$$

We do not know of any stability analysis for this approach; our own preliminary experiments suggest a rather slow convergence.

Suppose that we want to solve $\mathrm{sign}(A)x = b$. Then, in view of the previous section, we suggest to start with constructing the Krylov subspace $K^m(A; b)$, and to compute the sign function for the reduced matrix $H_{m,m}$. This leads to the following approximation for the solution of $\mathrm{sign}(A) = b$:

$$x_m = \|b\|_2 V_m \mathrm{sign}(H_{m,m})^{-1} e_1. \tag{12.6}$$

[1] Private communication

Our preliminary experiments with this approach are encouraging. The actual problem in QCD, however, is often to solve an essentially different equation

$$(C + \operatorname{sign}(A))x = b.$$

See, for instance, [131]. We have not yet succeeded in identifying efficient computational schemes for this shifted equation. An obvious way to solve this equation is to apply a nested Krylov solution process. In the outer process one forms the Krylov subspace for $C + \operatorname{sign}(A)$ and b, and for each matrix vector product involved in this process one computes approximations for $\operatorname{sign}(A)p$, for given vector v, using the process that we have outlined above. Our experiments, not reported here, show that this is a very expensive process. We are currently experimenting with nested Newton iterations for the inverse of $C + \operatorname{sign}(A)$ and the sign function. The idea is to carry out these Newton processes in an inexact way and to accelerate the process, in a way that has been described in [77].

Chapter 13

Preconditioning

13.1 Introduction

There are many occasions and applications where iterative methods fail to converge or converge very slowly. The usual remedy is to apply a preconditioner, that is instead of $Ax = b$, one solves $KAx = Kb$ or a spectrally equivalent system, for example, $AKy = b$. The general problem of finding an efficient preconditioner, is to identify a linear operator K (the *preconditioner*) 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 $Ky = z$ is much easier to solve than the original system.

By efficient, we mean that the iteration method converges much faster, in terms of CPU time, for the preconditioned system.

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. We refer the reader to [10, 33, 150] for further discussions on this. In this section, we will not go in details about preconditioning but rather give a sketch of important ideas for the construction of parallel preconditioners. For more details on implementation for high performance computers, see [57].

Originally, preconditioners were based on direct solution methods in which part of the computation is skipped. This leads to the notion of *Incomplete LU (or ILU) factorization* [125, 10, 150],

13.2 Incomplete LU factorizations

Standard Gaussian elimination is equivalent to factoring the matrix A as $A = LU$, where L is lower triangular and U is upper triangular. In actual computations these factors are explicitly constructed. The main problem in sparse matrix computations is that the factors of A are often a good deal less sparse than A, which makes solution expensive, The basic idea in the point ILU preconditioner is to modify Gaussian elimination to allow fill-ins at only a restricted set of positions in the LU factors. Let the allowable fill-in positions be given by the index set S, i.e.

$$\begin{aligned} l_{i,j} &= 0 \quad \text{if} \quad j > i \quad \text{or} \quad (i,j) \notin S \\ u_{i,j} &= 0 \quad \text{if} \quad i > j \quad \text{or} \quad (i,j) \notin S. \end{aligned} \tag{13.1}$$

A commonly used strategy is to define S by:

$$S = \{(i,j) | \quad a_{i,j} \neq 0\}. \tag{13.2}$$

That is, the only non-zeros allowed in the LU factors are those for which the corresponding entries in A are non-zero. Before we proceed with different strategies for the construction of effective incomplete factorizations we consider the question whether these factorizations exist. It can be shown that they exist for so-called M-matrices[1]. It can be shown [185] that the diagonal elements of an M-matrix are positive.

The theory of M-matrices for iterative methods is very well covered by Varga [185]. These matrices occur frequently after discretisation of PDE's, and for M-matrices one can identify all sorts of approximating matrices K for which the basic splitting leads to a convergent iteration (2.1). For preconditioners for Krylov subspace methods it is not important that the basic iteration converges; primarily we want reduced condition numbers and/or better eigenvalue distributions for the preconditioned matrices. These latter properties are very difficult to prove. In fact, we have been able to prove some of these effects for very special model problems.

We will now first show how the M-matrix theory functions in the construction of incomplete LU factorizations. We consider first the complete

[1] The nonsingular matrix A is an M-matrix if $a_{i,j} \leq 0$, for $i \neq j$, and $A^{-1} \geq 0$

Gaussian elimination process, for which the following extension of a theorem by K.Fan [73, p.44] is useful [125].

Theorem 2 *Gaussian elimination preserves the M-matrix property.*

Proof: We consider one step in the elimination process: the elimination of the sub-diagonal elements in the first column of A. In matrix notation this leads to

$$A^{(1)} = L^{(1)}, \tag{13.3}$$

with

$$L^{(1)} = \begin{pmatrix} 1 & & & \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & & \\ \vdots & & \ddots & \\ -\frac{a_{n,1}}{a_{1,1}} & & & 1 \end{pmatrix} \geq 0 \ .$$

The elements of $A^{(1)}$ are given by $a_{i,j}^{(1)} = a_{i,j} - \frac{a_{1,j}}{a_{1,1}} a_{i,1}$, for $i > 1$, and $a_{1,j}^{(1)} = a_{1,j}$.

Note that $a_{i,j}^{(1)} \leq 0$ for $i \neq j$.

Remains to be shown that $(A^{(1)})^{-1} \geq 0$. We consider the i-th column

$$(A^{(1)})^{-1} e_i = A^{-1} (L^{(1)})^{-1} e_i.$$

For $i = 1$ it follows that $A^{-1}(L^{(1)})^{-1} e_1 = \frac{1}{a_{1,1}} e_1$. For $i \neq 1$ we have that $(A^{(1)})^{-1} e_i = A^{-1} e_i \geq 0$. □

After an elimination step we ignore certain fill-in elements in off-diagonal positions. Now the following (almost trivial) extension of a theorem of Varga [185, Th.3.12] is helpful [125, Th.2.2].

Theorem 3 *If $A = (a_{i,j})$ be a real $n \times n$ $M-$matrix and $C = (c_{i,j})$ is a real $n \times n$ matrix with $c_{i,i} \geq a_{i,i}$ and $a_{i,j} \leq c_{i,j} \leq 0$, for $i \neq j$, then C is also an M-matrix.*

Exercise 13.1 *Suppose that we replace an off-diagonal element in the second row of $A^{(1)}$ by zero. Let the result after the elimination of the second column of the remaining matrix be denoted by $\widetilde{A}^{(2)}$. Show that*

$$\widetilde{a}_{i,i}^{(2)} \geq a_{i,i}^{(2)}, \tag{13.4}$$

for all i, and

$$a_{i,j}^{(2)} \leq \widetilde{a}_{i,j}^{(2)} \leq 0, \tag{13.5}$$

for $i \neq j$.

Theorem 2 says that after a Gaussian elimination step on an M-matrix A, we obtain a reduced matrix that is still an M-matrix. The relations (13.4) and (13.5), together with Theorem 3, show that even after ignoring off-diagonal elements in the reduced matrix we still have an M-matrix.

Exercise 13.2 *Suppose that A is an M-matrix. Consider the gaussian elimination corrections to the diagonal of A after one step of Gaussian elimination (that is, consider the diagonal elements of $A^{(1)} - A$). Show that after ignoring some of these diagonal corrections the remaining matrix has still the M-matrix property.*

After $n-1$ steps of Gaussian elimination we obtain an upper triangular matrix

$$U \equiv A^{(n-1)} = L^{(n-1)} L^{(n-2)} \cdots L^{(1)} A, \tag{13.6}$$

with

$$L^{-1} \equiv L^{(n-1)} L^{(n-2)} \cdots L^{(1)}. \tag{13.7}$$

This gives the decomposition $A = LU$. Repeated application of Theorem 2 leads to the observation that U is an M-matrix. Because of the above given arguments, we conclude that U is still an M-matrix after ignoring Gaussian elimination corrections in various stages of the elimination process.

Now we consider the elimination matrices $L^{(i)}$. Obviously the elements of these matrices are non-negative.

The following exercise involves the proof of a very well-known fact (see, for example, [90]).

Exercise 13.3 *Show that the inverse of $L^{(i)}$ is obtained by simply multiplying the off-diagonal elements with -1. Show also that $(L^{(i)})^{-1}$ is an M-matrix, if A is an M-matrix.*

Exercise 13.4 *Prove that L (cf (13.7) is an M-matrix if A is an M-matrix.*

Exercise 13.5 *Show that if we ignore arbitrary off-diagonal elements in the $L^{(i)}$, the remaining lower trinagular matrix L is still an M-matrix (if A is an M-matrix).*

We have so arrived at an incomplete LU decomposition of the M-matrix A. Obviously, the incomplete factors exist, but there is more. We have seen that ignoring gaussian elimination elements in L and U has the effect that diagonal elements of the upper triangular factor U become larger and that the off-diagonal elements of the factors L and U become smaller in absolute value (but remain negative).

The following theorem collects the above results for the situation where we determine a priori the positions of the elements that we wish to ignore during the Gaussian elimination process. Note that this is not a serious restriction because we may also neglect elements during the process according to certain criteria and this defines the positions implicitly. The indices of the elements to be ignored are collected in a set $\mathcal{S}$:

$$S \subset S_n \equiv \{(i,j) | i \neq j, 1 \leq i \leq n, 1 \leq j \leq n\}. \tag{13.8}$$

We can now formulate the Theorem that guarantees the existence of incomplete decommpositions for the M-matrix A (cf. [125, Th.2.3]).

Theorem 4 *Let $A = (a_{i,j})$ be an $n \times n$ M-matrix, then there exists for every $S \subset S_n$ a lower triangular matrix $\widetilde{L} = (\ell_{i,j})$, with $\ell_{i,i} = 1$, an upper triangular matrix $\widetilde{U} = (u_{i,j})$, and a matrix $N = (n_{i,j})$ with*

- *$\ell_{i,j} = 0$, $u_{i,j} = 0$, if $(i,j) \in S$*
- *$n_{i,j} = 0$ if $(i,j) \notin S$,*

such that the splitting $A = \widetilde{L}\widetilde{U} - N$ leads to a convergent iteration (2.1). The factors $\widetilde{L}$ and $\widetilde{U}$ are uniquely defined by S.

One can make, of course, variations on these incomplete splittings in which the diagonal of $\widetilde{U}$ is taken as a separate factor, or where the diagonals of $\widetilde{L}$ and $\widetilde{U}$ are equal. The latter leads to incomplete Choleski decompositions.

A commonly used strategy is to define S by:

$$S = \{(i,j) | \quad a_{i,j} \neq 0\}. \tag{13.9}$$

That is, the only non-zeros allowed in the LU factors are those for which the corresponding entries in A are non-zero. Let the preconditioner M be defined by the product of the resulting LU factors, i.e. $M = LU$. For M to be a good preconditioner, it must be a good approximation to A in some measure. A typical strategy is to require the entries of M to match those of A on the set S:

$$m_{i,j} = a_{i,j} \quad \text{if} \quad (i,j) \in S. \tag{13.10}$$

Even though the conditions (13.1) and (13.10) together are sufficient (for certain classes of matrices) to determine the non-zero entries of L and U directly, it is more natural and simpler to compute these entries based on a simple modification of the Gaussian elimination algorithm; see Fig. 13.1. The main difference from the usual Gaussian elimination algorithm is in the inner-most j-loop where an update to $a_{i,j}$ is computed only if it is allowed by the constraint set S.

ILU for an n by n matrix A (cf [10]):
for $k = 1, 2, ..., n-1$
    $d1/a_{k,k}$
    for $i = k+1, k+2, ..., n$
      if $(i, k) \in S$
        $e = da_{i,k}; a_{i,k} = e$
        for $j = k+1, ..., n$
          if $(i, j) \in S$ and $(k, j) \in S$
            $a_{i,j} = a_{i,j} - ea_{k,j}$
          end if
        end j
      end if
    end i
end k

Figure 13.1: ILU for a general matrix A

After the completion of the algorithm, the incomplete LU factors are stored in the corresponding lower and upper triangular parts of the array A. It can be shown that the computed LU factors satisfy (13.10).

The incomplete factors $\widetilde{L}$ and $\widetilde{U}$ define the preconditioner $K = (\widetilde{L}\widetilde{U})^{-1}$. In the context of an iterative solver, this means that we have to evaluate expressions like $z = (\widetilde{L}\widetilde{U})^{-1}y$ for any given vector y. This is done in two steps: first obtain w from the solution of $\widetilde{L}w = y$ and then compute z from $\widetilde{U}z = w$. Straightforward implementation of these processes leads to recursions, for which vector and parallel computers are not ideally suited. This sort of observation has led to reformulations of the preconditioner, for example, with reordering techniques or with blocking techniques. It has also led to different types of preconditioners, including diagonal scaling, polynomial preconditioning, and truncated Neumann series. These approaches may be useful in certain circumstances, but they tend to increase the computational complexity, because they often require more iteration steps or make each iteration step more expensive. Diagonal scaling can be done explicitly, without any further complications for parallel processing. This can be done before and in addition to the construction of another preconditioning.

13.2.1 An Example of Incomplete Decompositions

We shall illustrate the above sketched process for a popular preconditioner for sparse positive definite symmetric matrices, namely, the *Incomplete Cholesky factorization* [90, 125, 126, 184] 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 two-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_xth co-diagonal, then the ith row of the symmetric matrix A can be represented as in (13.11).

$$A = \begin{pmatrix} \ddots & & & \ddots & \ddots & \ddots & & \ddots & \\ & a_{i-n_x,3} & & & a_{i-1,2} & a_{i,1} & a_{i,2} & & a_{i,3} \\ & \ddots & & & \ddots & \ddots & \ddots & & \ddots \end{pmatrix} \tag{13.11}$$

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

$$\begin{array}{cccccccc}
\cdot & \cdot & \cdot & & \cdot & & \cdot & \cdot \\
\\
\cdot & \cdot & \cdot & & \cdot & & \cdot & \cdot \\
 & & & & \downarrow & & & \\
\cdot & \cdot & \cdot & \rightarrow & \cdot & \leftarrow & \cdot & \cdot \\
 & & & & \uparrow & & & \\
\cdot & \cdot & \cdot & & \cdot & & \cdot & \cdot \\
\\
\cdot & \cdot & \cdot & & \cdot & & \cdot & \cdot \\
\\
\cdot & \cdot & \cdot & & \cdot & & \cdot & \cdot
\end{array}$$

If we write A as $A = L_A + \mathrm{diag}(A) + {L_A}^T$, 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 nonzero entries in the incomplete elimination process for A and if we ignore all fill-in outside the nonzero 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 nonzero 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 5-diagonal A, this leads to the following relations for the d_i:

$$d_i = a_{i,1} - a_{i-1,2}^2/d_{i-1} - a_{i-n_x,3}^2/d_{i-n_x}. \tag{13.12}$$

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* [66, 96] follow from the requirement that

$$\mathrm{rowsum}(K) = \mathrm{rowsum}(A) + ch^2. \tag{13.13}$$

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 [67]. In our context, the rowsum requirement in (13.13) amounts to an additional correction to the diagonal entries d_i, compared to those computed in (13.12).

Axelsson and Lindskog [11] describe a relaxed form of this modified incomplete decomposition that, for the 5-diagonal A, leads to the following relations for the d_i:

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

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 [96]. 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 [175]. 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 nonzero 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 [104], and SIP [162]. 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.

13.2.2 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 [69] has proposed an efficient implementation for preconditioned iterative methods, when the preconditioner K can be represented as

$$K = (L_A + D)D^{-1}(D + U_A), \tag{13.14}$$

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 [126]; 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 two dimensions, this is equivalent to the incomplete factorizations with no fill-in, since in these situations there are no Gaussian elimination operations on nonzero off-diagonal entries.

The first step to make the preconditioning more efficient is to eliminate the diagonal D in (13.14). We rescale the original linear system $Ax = b$ to obtain

$$D^{-1/2}AD^{-1/2}\widetilde{x} = D^{-1/2}b, \tag{13.15}$$

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

$$\widetilde{K} = (L_{\widetilde{A}} + I)(I + U_{\widetilde{A}}). \tag{13.16}$$

Note that the corresponding triangular systems, like $(L_{\widetilde{A}}+I)r = w$, 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 $\mathrm{diag}(\widetilde{A}) = I$.

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

$$(L_{\widetilde{A}} + I)^{-1}\widetilde{A}(I + U_{\widetilde{A}})^{-1}y = (L_{\widetilde{A}} + I)^{-1}\widetilde{b}, \tag{13.17}$$

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

$$\widetilde{A} = L_{\widetilde{A}} + I + \mathrm{diag}(\widetilde{A}) - 2I + I + U_{\widetilde{A}}. \tag{13.18}$$

This expression, as well as the special form of the preconditioner given by (13.16), is used to compute the vector Pz for a given z by

$$Pz = (L_{\widetilde{A}}+I)^{-1}\widetilde{A}(I+U_{\widetilde{A}})^{-1}z = (L_{\widetilde{A}}+I)^{-1}(z+(\mathrm{diag}(\widetilde{A})-2I)t)+t, \tag{13.19}$$

with

$$t = (I + U_{\widetilde{A}})^{-1}z. \tag{13.20}$$

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.

13.3 Changing the Order of Computation

In some situations, it is possible to change the order of the computations without changing the results. A prime example is the ILU preconditioner for the 5-point finite-difference operator over a rectangular m by m grid. Suppose that we have indexed the unknowns according to their positions in the grid, lexicographically as $x_{1,1}$, $x_{1,2}$, $\ldots$, $x_{1,m}$, $x_{2,1}$, $\ldots$, $x_{m,m}$. Then, for the standard ILU(0) preconditioner, in which all fill-ins are discarded, it is easily verified that the computations for the unknowns $x_{i,j}$ can be done independently of each other along diagonals of the grid (grid points for which the sum of the indices is constant). This leads to vector code but, because there is only independence along each diagonal, the parallelism is too fine-grained. In three-dimensional problems, there are more possibilities for obtaining vectorizable or parallel code. For the standard 7-point finite-difference approximation of elliptic PDEs over a regular rectangular grid, the equivalent of the diagonal in two dimensions is known as the hyperplane: a set of grid points for which the sum of the three indices is constant. It was reported in [152, 174] that this approach can lead to satisfactory performance on vector computers. For the CM-5 computer, a similar approach was developed in [24]. The obvious extension of hyperplanes (or diagonals) to irregular sparse matrices, defines the *wavefront ordering*, discussed in [145]. The success of a wavefront ordering depends very much on how well a given computer can handle indirect addressing. In general, the straightforward wavefront ordering approach gives too little opportunity for efficient parallelization.

Vuik, van Nooyen and Wesseling [187], generalize the wavefront approach to a block wavefront approach, using ideas that were originally proposed for parallel multigrid in [19]. They present results of experiments on a 128-processor CRAY 3TD. Van Duin [182, Chapter 3], uses graph concepts for the detection of parallelism. He attempts to identify strongly connected components for which independent ILU factorizations can be made. A drop tolerance strategy is used to create a large enough number of such components. This leads to the concept of MuliILU.

13.4 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. For the 5-point finite-difference discretization over rectangular grids, this approach is known as a *red-black ordering*. For elliptic PDEs, this leads to very parallel preconditioners. The performance of the preconditioning step is as high as the performance of the matrix-vector product. However, changing the order of the unknowns leads in general to a different preconditioner. Duff and Meurant [64] report on experiments that show that most reordering schemes (for example, the red-black ordering) lead to a considerable increase in iteration steps (and hence in computing time) compared with the standard lexicographical ordering. For the red-black ordering associated with the discretized Poisson equation, 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 the gridsize h tends to zero [117].

One way to obtain a better balance between parallelism and fast convergence, is to use more colours [53]. In principle, since there is not necessarily any independence between different colours, using more colours decreases the parallelism but increases the global dependence and hence the convergence. In [54], up to 75 colours 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 for the wavefront ordering. With this large number of colours the speed of convergence for the preconditioned process is virtually the same as with a lexicographical ordering [53].

The concept of *multi-colouring* has been generalized to unstructured problems by Jones and Plassmann [114]. They propose effective heuristics for the identification of large independent subblocks of a given matrix. For problems large enough to get sufficient parallelism in these subblocks, their approach leads to impressive speedups compared to the natural ordering on a single processor.

Meier and Sameh [124] 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).

Another approach, suggested by Meurant [128], exploits the idea of the two-sided (or twisted) Gaussian elimination procedure for tridiagonal matrices. This is generalized for the incomplete factorization. Van der Vorst [173] has shown how this procedure can be done in a nested way. For 3D finite-difference problems, twisting can be used for each dimension, which gives an increase in parallelism by a factor of two per dimension. This leads, without further computational overhead, to an incomplete decompositions, as well as triangular solves, that can be done in eight parallel parts (2 in each dimension). For a discussion of these techniques see [57].

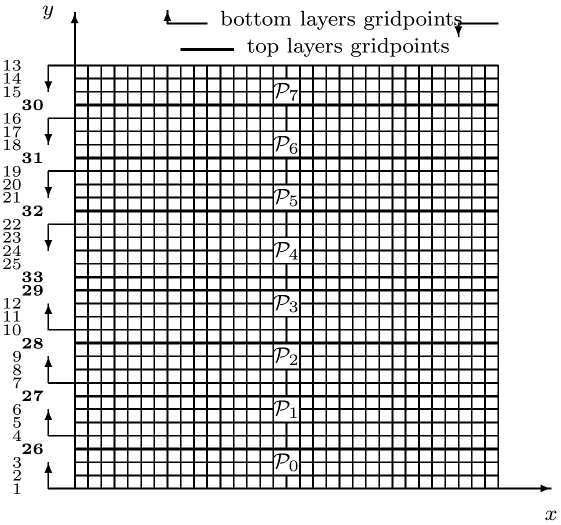

Figure 13.2: Decomposition of the grid into stripes,and assignment of subdomains to processors for $p = 8$. Arrows indicate the progressing direction of the line numbering per subdomain. Numbers along the y-axis give an example of global (line) ordering, which satisfy all the required conditions. Within each horizontal line, gridpoints are ordered lexicographically.

Meurant [129] reports on timing results obtained on a CRAY Y-MP/832, using an incomplete repeated twisted block factorization for two-dimensional problems. For this approach for preconditioned CG, Meurant reports a speedup of nearly 6 on an 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 best non-parallel code is only about 4.

A more sophisticated approach that combines ideas from twisting, domain decomposition with overlap, and reordering, was proposed by in [121, 122, 120]. We will explain this idea for the special situation of a discretized second order elliptic PDE over a rectangular domain. The discretization has been carried out with the standard 5-point central difference stencil which, over a rectangular grid with lexicographical ordering leads to the familiar block matrix with 5 nonzero diagonals.
The first step is to split the domain in blocks, as in domain decomposition methods, and to order the unknowns lexicographically per block. This has

been indicated, for the case of 8 horizontal blocks, in Figure 13.2. Per block we start counting from one side ("the bottom layer"); the points on the last line ("the top layer") are ordered after all subdomains, as is indicated in Fig. 13.3. For instance, the lines 1, 2, 3, and 26, all belong to the block stored with processor $\mathcal{P}_0$, but in the matrix interpretation the first 3 lines are ordered first and line 26 appears in the matrix only after all other "interior" lines. This means that the matrix has the following nonzero structure (we give only a relevant part of the matrix). Note that we have already introduced another element in our ordering, namely the idea of twisting: the lines of the subdomains are ordered from bottom to top and from top to bottom in Fig. 13.2.

$$
\begin{array}{c}
\begin{matrix}
1 & & & & & 2 & & & & 3 & & & & 4 & & & & \cdot & & \cdot & & \cdot & & & & 26 & &
\end{matrix} \\
\begin{bmatrix}
* & * & & & * \\
* & * & * & & & * \\
& * & * & * & & & * \\
& & * & * & & & & * \\
* & & & & * & * & & & * \\
& * & & & * & * & * & & & * \\
& & * & & & * & * & * & & & * \\
& & & * & & & * & * & & & & * \\
& & & & * & & & & * & * & & & & & & & & & & & & & & & & * \\
& & & & & * & & & * & * & * & & & & & & & & & & & & & & & * \\
& & & & & & * & & & * & * & * & & & & & & & & & & & & & & & * \\
& & & & & & & * & & & * & * & & & & & & & & & & & & & & & & * \\
& & & & & & & & & & & & * & * & & & * & & & & & & & & * \\
& & & & & & & & & & & & * & * & * & & & * & & & & & & & & * \\
& & & & & & & & & & & & & * & * & * & & & * & & & & & & & & * \\
& & & & & & & & & & & & & & * & * & & & & * & & & & & & & & * \\
& & & & & & & & & & & & * \\
& & & & & & & & & & & & & * \\
& & & & & & & & & & & & & & *
\end{bmatrix}
\end{array}
$$

Figure 13.3: The structure of the reordered matrix

Now imagine what happens if we carry out an incomplete LU factorization with zero fill. That would create level-1 fill in the error matrix. Note that, in particular, we would introduce fill in the subblock of the matrix that connects line 26 with line 5, and note also that we would not have seen this level-1 fill if we would have selected all points lexicographically.

Exercise 13.6 *Complete the structure in the matrix of Fig 13.3 and check where fill-in occurs due to incomplete pivoting. Identify the fill positions that lead to connections of unknowns that would not have occurred with lexicographical ordering.*

This means that if we want the block ordering to be at least as effective as the standard ordering, we have to remove this additional fill. This can be interpreted as permitting level-1 fill in a small overlap, and this is the reason for the name *pseudo-overlap"* for this way of ordering. It is obvious how to generalize this idea for more arbitrary matrices: one compares the new ordering with the standard given one and one includes the possibly additional level-1 fill in the preconditioner. The idea can also be easily applied to preconditioners with a higher level fill.

In [121, 120] it is suggested to increase the pseudo-overlap and to include also higher levels of fill that are introduced by the new block-wise ordering. For high dimensional problems and relatively low numbers of processors this leads to almost negligible overhead. It is shown by analysis in [122] and by experiments [121, 120] that the block ordering with pseudo-overlap may lead to parallelizable incomplete decompositions that are almost perfectly scalable if the number of processors p is less than $\sqrt(n)$, where n denotes the order of the given linear system (the reported experiments include experiments for 16 processors, for $n \approx 260\,000$).

13.5 Hybrid Techniques

In the classical incomplete decompositions one ignores fill-in right from the start of the decomposition process. However, it might be a good idea to delay this until the matrix becomes too dense. This leads to a hybrid combination of direct and iterative techniques. One of such approaches has been described in [26]; we will describe it here in some detail.

We first permute the given matrix of the linear system $Ax = b$ to a doubly bordered block diagonal form:

$$\widetilde{A} = P^T A P = \begin{bmatrix} A_{00} & 0 & \cdots & 0 & A_{0m} \\ 0 & A_{11} & \ddots & \vdots & A_{1m} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & A_{m-1m-1} & \vdots \\ A_{m0} & A_{m1} & \cdots & \cdots & A_{mm} \end{bmatrix}. \tag{13.21}$$

Of course, the parallelism in the eventual method depends on the value of m, and some problems lend themselves more to this than others. Many circuit simulation problems can be rewritten in an effective way, as a circuit is often composed of components that are only locally coupled to others.

We permute the right-hand side b as well to $\widetilde{b} = P^T b$, which leads to the system

$$\widetilde{A}\widetilde{x} = \widetilde{b}, \tag{13.22}$$

with $x = Px$.

The parts of $\widetilde{b}$ and $\widetilde{x}$ that correspond to the block ordering, will be denoted by $\widetilde{b}_i$ and $\widetilde{x}_i$. The first step in the (parallelizable) algorithm will be to eliminate the unknown parts $\widetilde{x}_0, \cdots, \widetilde{x}_{m-1}$, which is done by the Algorithm in Fig. 13.4.

Parallel_for $i = 0, 1, ..., m-1$
    Decompose A_{ii}: $A_{ii} = L_{ii}U_{ii}$
    $L_{mi} = A_{mi}U_{ii}^{-1}$
    $U_{im} = L_{ii}^{-1}A_{im}$
    $y_i = L_{ii}^{-1}\widetilde{b}_i$
    $S_i = L_{mi}U_{im}$
    $z_i = L_{mi}y_i$
end
$S = A_{mm} - \sum_{i=0}^{m-1} S_i$
$y_m = \widetilde{b}_m - \sum_{i=0}^{m-1} z_i$
Solve $Sx_m = y_m$
Parallel_for $i = 0, 1, ..., m-1$
    $x_i = U_{ii}^{-1}(y_i - U_{im}x_m)$
end

Figure 13.4: Parallel elimination

Note that S in Fig. 13.4 denotes the Schur complement after the elimination of the blocks $0, 1, \ldots, m-1$. In many relevant situations, direct solution of the reduced system $Sx_m = y_m$ requires the dominating part of the total computational costs, and this is where we bring in the iterative component of the algorithm.

Exercise 13.7 *Suppose that we solve the reduced system $Sx_m = y_m$ with an iterative method and that after termination we have the approximated solution $\widehat{x}_m$, with $r_m = S\widehat{x}_m - y_m$. When we take this approximated solution for the computation of the x_i in Fig. 13.4, then this leads to an approximated*

solution $\widehat{x}$ for the system $Ax = b$.
Show that, in exact computation,

$$\|A\widehat{x} - b\|_2 = \|r_m\|_2. \tag{13.23}$$

The next step is to construct a preconditioner for the reduced system. This is based on discarding small elements in S. The elements larger than some threshold value define the preconditioner C:

$$c_{ij} = \begin{cases} s_{ij} & \text{if } |s_{ij} > t|s_{ii}| \text{ or } |s_{ij}| > t|s_{jj}| \\ 0 & elsewhere \end{cases} \tag{13.24}$$

with a parameter $0 \leq t < 1$. In the experiments, reported in [26] the value $t = 0.02$ turned out to be satisfactory, but this may need some experimentation for specific problems.
When we take C as the preconditioner, then we have to solve systems like $Cv = w$, and this requires decomposition of C. In order to prevent too much fill-in, it is suggested to reorder C with a minimum degree ordering. The system $Sx_m = y_m$ is then solved with, for instance, GMRES with preconditioner C. For the examples described in [26] it turns out that the convergence of GMRES was not very sensitive to the choice of t. The preconditioned iterative solution approach for the reduced system offers also opportunities for parallelism, although in [26] it is shown that even in serial mode the iterative solution (too sufficiently high precision) is often more efficient than direct solution of the reduced system.
Note that, because of (13.23), it is not necessary to iterate on the complete system.

In [26] heuristics are described for the decision on when the switch from direct to iterative should take place. These heuristics are based on mild assumptions on the speed of convergence of GMRES. The paper also reports on a number of experiments for linear systems, not only from circuit simulation, but also for some matrix problems taken from Matrix Market[2]. These experiments indicate that attractive savings in computational costs can be achieved, even in serial computation mode.

13.6 Element by Element Preconditioners

In finite-element problems, it is not always possible or sensible to assemble the entire matrix, and it is as easy to form products of the matrix with

[2]Collection of testmatrices available at
`ftp://ftp.cise.ufl.edu/cis/tech-reports/tr98/tr98-016.ps`

vectors as when it is held in assembled form. Furthermore it is easy to distribute such matrix multiplications to exploit parallelism. Hence preconditioners are required that can be constructed at the element level. Hughes et al. [109] were the first to propose such *element by element* preconditioners.

A parallel variant is suggested in [97]. For symmetric positive-definite A, they decompose each element matrix A_e as $A_e = L_e L_e^T$, and construct the preconditioner as $K = LL^T$, with

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

In this approach, non-adjacent elements can be treated in parallel. An overview and discussion of parallel element by element preconditioners is given in [183]. To our knowledge, the effectiveness of element by element preconditioners is limited, in the sense that it does not often give a substantial improvement of the CPU time.

13.7 Polynomial Preconditioning

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). By doing so, all implementation tricks for the matrix-vector product can easily be exploited. The main problem is to find effective low degree polynomials $p_k(A)$, so that the iterative solver can be applied to $p_k(A)Ax = p_k(A)b$. With m steps of a Krylov solver, this leads to a Krylov subspace

$$K^m(p_k(A)A; r_0) = \mathrm{span}(r_0, p_k(A)Ar_0, \ldots, (p_k(A)A)^{m-1}r_0),$$

and this is a subspace of the Krylov subspace $K^{(k+1)(m-1)+1}(A; r_0)$. The point is, that we have arrived in a high dimensional subspace (with "holes"), for the overhead costs of only m iteration steps. The hope is that for clever choices of p_k, this high dimensional subspace, with holes, will contain almost the same good approximation to the solution as the full Krylov subspace. If so, then we have saved ourselves all the overhead associated with the $(k+1)(m-1)$ iteration steps, that are needed to create the full subspace.

A big problem with polynomial preconditioning is that the aforementioned "holes" can cause one to miss important directions and so often many more iterations are required. Thus this form of preconditioning is usually only beneficial on a platform where inner products are expensive and for methods rich in inner products, like GMRES.

One approach for obtaining a polynomial preconditioner, reported in [61], 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 in [61] 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, [7, 113, 147]). These polynomials are usually shifted Chebyshev polynomials over intervals that are estimated from the iteration parameters of a few steps of the unpreconditioned solver, or from other spectral information.

13.8 Sparse Approximate Inverse (SPAI)

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 factorization techniques will produce a dense matrix even if some of its entries are actually zero [65]. However, this need not be a problem if we follow the flavour 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[3]

$$\min_K ||I - AK||_F, \tag{13.25}$$

where K has some fully or partially prescribed sparsity structure. This problem can be expressed as n independent least-squares problems for each of the n columns of K. Each of these least-squares problems only involves a few variables and, because they are independent, they can be solved in parallel. With these techniques it is possible to successively increase the density of the approximation to reduce the value of (13.25) and so, in principle, ensure convergence of the preconditioned iterative method [42]. The small least-squares subproblems can be solved by standard (dense) QR factorizations [42, 91, 95]. In a further attempt to increase sparsity and reduce the computational cost of the solution of the subproblems, it has been suggested to use a few steps of GMRES to solve the subsystems [35]. A recent study indicates that the computed approximate inverse may be a good alternative to ILU [91], 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

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

normally only attractive to use this technique if the computational costs for the construction can be amortized by using the preconditioner for several 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. Partly to avoid this, an approach that approximates the triangular factors of the inverse has been proposed [116]. The nonsingularity of the factors can be easily controlled and, if necessary, the sparsity pattern of the factors may also be controlled. Following this approach, sparse approximations to an A-biconjugate set of vectors using drop tolerances can be generated [21, 23]. 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 [22]. Van Duin [182, Chapter 5] shows how to compute (*sparsified*) inverses for incomplete Choleski factors and Zhang [195] has developed a parallel preconditioning using incomplete triangular factors of the inverse.

Although almost every paper on approximate inverse preconditioners states that the authors are working on a parallel implementation, it is only quite recently that papers on this have appeared [16, 17, 20]. For highly structured matrices, some experiences have been reported in [94]. Gustafsson and Lindskog [98] have implemented a fully parallel preconditioner based on truncated Neumann expansions [171] 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 problems in electromagnetism [1]. 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 [3]. An incomplete form of this factorization for use as a preconditioner was proposed in [2].

13.9 Preconditioning by Blocks or Domains

Other 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 can be done in *domain decomposition*, where problems on subdomains can be solved by a direct method but the interaction between the subproblems is handled by an iterative technique.

Domain decomposition methods were motivated by parallel computing, but it now appears that the approach can also be used with success for the construction of global preconditioners. This is usually done for linear systems that arise from the discretization of a PDE. The idea is to split the given domain into subdomains, and to compute an approximation for the solution on each subdomain. If all connections between subdomains are ignored, this then leads to a *Block Jacobi* preconditioner. Chan and Goovaerts [32] 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.

In order to make the preconditioner more successful, one has to couple the domains, that is one has to find proper boundary conditions along the interior boundaries of the subdomains. From a linear algebra point of view, this amounts to adapting the diagonal blocks in order to compensate for the neglected off-diagonal blocks. This is only successful if the matrix comes from a PDE problem and if certain smoothness conditions on the solution are assumed. If, for instance, the solution were constant, then one could remove the off-diagonal block entries adding them to the diagonal block entries without changing the solution. Likewise, if the solution is assumed to be fairly smooth along a domain interface, one might expect this technique of diagonal block correction to be effective. Domain decomposition is used in an iterative fashion and usually the interior boundary conditions (in matrix language: the corrections to diagonal blocks) are based upon information from the approximate solutions on the neighbouring subdomains that are available from a previous iteration step.

Tang [166] has proposed the concept of matrix enhancement. The idea is to introduce additional unknowns, which gives elegant possibilities for the formulation of effective domain decomposition of the underlying problem. For hyperbolic systems, this technique was further refined by Tan in [164] and by Tan and Borsboom [165]. We will explain the idea for the situation of two domains.

13.9.1 Canonical enhancement of a linear system

We start with the linear nonsingular system

$$\mathbf{B}\,\mathbf{y} = \mathbf{d}, \tag{13.26}$$

that results from discretization of a given PDE over some domain. Now, we partition the matrix $\mathbf{B}$, and the vectors $\mathbf{y}$ and $\mathbf{d}$ correspondingly,

$$\begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{1\ell} & \mathbf{B}_{1r} & \mathbf{B}_{12} \\ \mathbf{B}_{\ell 1} & B_{\ell\ell} & B_{\ell r} & \mathbf{B}_{\ell 2} \\ \mathbf{B}_{r1} & B_{r\ell} & B_{rr} & \mathbf{B}_{r2} \\ \mathbf{B}_{21} & \mathbf{B}_{2\ell} & \mathbf{B}_{2r} & \mathbf{B}_{22} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{y}_1 \\ y_\ell \\ y_r \\ \mathbf{y}_2 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} \mathbf{d}_1 \\ d_\ell \\ d_r \\ \mathbf{d}_2 \end{bmatrix}. \tag{13.27}$$

The labels are not chosen arbitrarily: we associate with label 1 (and 2, respectively) elements/operations of the linear system corresponding to subdomain 1 (2, respectively) and with label ℓ (resp. r) elements/operations corresponding to the left (resp. right) of the interface between the two subdomains. The central blocks $B_{\ell\ell}$, $B_{\ell r}$, $B_{r\ell}$ and B_{rr} are square matrices of equal size, say, n_i by n_i. They correspond to the unknowns on the interface. Since the number of unknowns on the interface will typically be much smaller than the total number of unknows, n_i will be much smaller than n, the size of $\mathbf{B}$.

For a typical discretization, the matrix $\mathbf{B}$ is banded and the unknowns are only locally coupled. Therefore it is not unreasonable to assume that $\mathbf{B}_{r1}, \mathbf{B}_{21}, \mathbf{B}_{12}$ and $\mathbf{B}_{\ell 2}$ are zero. For this situation, we define the "canonical enhancement" $\underline{\mathbf{B}}$ of $\mathbf{B}$, $\widetilde{\mathbf{y}}$ of $\mathbf{y}$, and $\underline{\mathbf{d}}$ of $\mathbf{d}$, by

$$\underline{\mathbf{B}} \equiv \left[\begin{array}{ccc|ccc} \mathbf{B}_{11} & \mathbf{B}_{1\ell} & \mathbf{B}_{1r} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_{\ell 1} & B_{\ell\ell} & B_{\ell r} & 0 & 0 & \mathbf{0} \\ \mathbf{0} & I & 0 & -I & 0 & \mathbf{0} \\ \hline \mathbf{0} & 0 & -I & 0 & I & \mathbf{0} \\ \mathbf{0} & 0 & 0 & B_{r\ell} & B_{rr} & \mathbf{B}_{r2} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{2\ell} & \mathbf{B}_{2r} & \mathbf{B}_{22} \end{array}\right], \quad \widetilde{\mathbf{y}} \equiv \begin{bmatrix} \mathbf{y}_1 \\ y_\ell \\ \widetilde{y}_r \\ \widetilde{y}_\ell \\ y_r \\ \mathbf{y}_2 \end{bmatrix}, \text{ and } \quad \underline{\mathbf{d}} \equiv \begin{bmatrix} \mathbf{d}_1 \\ d_\ell \\ 0 \\ 0 \\ d_r \\ \mathbf{d}_2 \end{bmatrix}. \tag{13.28}$$

One easily verifies that $\mathbf{B}$ is also nonsingular and that $\underline{\mathbf{y}}$ is the unique solution of

$$\underline{\mathbf{B}}\,\widetilde{\mathbf{y}} = \underline{\mathbf{d}}, \tag{13.29}$$

with $\underline{\mathbf{y}} \equiv (\mathbf{y}_1^T, y_\ell^T, y_r^T, y_\ell^T, y_r^T, \mathbf{y}_2^T)^T$.

With this linear system we can associate a simple iterative scheme for the two coupled subblocks:

$$\begin{bmatrix} \mathbf{B}_{11} & \mathbf{B}_{1\ell} & \mathbf{B}_{1r} \\ \mathbf{B}_{\ell 1} & B_{\ell\ell} & B_{\ell r} \\ \mathbf{0} & I & 0 \end{bmatrix} \begin{bmatrix} \mathbf{y}_1^{(i+1)} \\ y_\ell^{(i+1)} \\ \widetilde{y_r}^{(i+1)} \end{bmatrix} = \begin{bmatrix} \mathbf{d}_1 \\ d_\ell \\ \widetilde{y_\ell}^{(i)} \end{bmatrix},$$

$$\begin{bmatrix} 0 & I & \mathbf{0} \\ B_{r\ell} & B_{rr} & \mathbf{B}_{r2} \\ \mathbf{B}_{2\ell} & \mathbf{B}_{2r} & \mathbf{B}_{22} \end{bmatrix} \begin{bmatrix} \widetilde{y_\ell}^{(i+1)} \\ y_r^{(i+1)} \\ \mathbf{y}_2^{(i+1)} \end{bmatrix} = \begin{bmatrix} \widetilde{y_r}^{(i)} \\ d_r \\ \mathbf{d}_2 \end{bmatrix}. \tag{13.30}$$

These systems can be solved in parallel and we recognize this as nothing else than a simple additive Schwartz iteration (with no overlap and Dirichlet-Dirichlet coupling). The extra unknowns $\widetilde{y_\ell}$ and $\widetilde{y_r}$, in the enhanced vector $\widetilde{\mathbf{y}}$, will serve for communication between the subdomains during the iterative solution process of the linear system. After termination of the iterative process, we have to undo the enhancement. We could simply skip the values of the additional elements, but since these carry also information one of the alternatives could be the following one.
With an approximate solution

$$\widetilde{\mathbf{y}}^{(i)} = (\mathbf{y}_1^{(i)\,T}, y_\ell^{(i)\,T}, (\widetilde{y_r}^{(i)})^T, (\widetilde{y}_\ell^{(i)})^T, y_r^{(i)\,T}, \mathbf{y}_2^{(i)\,T})^T$$

of (13.30), we may associate the approximate solution $\mathbf{R}\widetilde{\mathbf{y}}$ of (13.26) given by

$$\mathbf{R}\widetilde{\mathbf{y}} \equiv (\mathbf{y}_1^{(i)\,T}, \frac{1}{2}(y_\ell^{(i)} + \widetilde{y}_\ell^{(i)})^T, \frac{1}{2}(y_r^{(i)} + \widetilde{y}_r^{(i)})^T, \mathbf{y}_2^{(i)\,T})^T, \tag{13.31}$$

that is, we simply average the two sets of unknowns that should have been equal to each other at full convergence.

13.9.2 Interface coupling matrix

From (13.28) we see that the interface unknowns and the additional interface unknowns are coupled in a straightforward way by

$$\begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} y_\ell \\ \widetilde{y_r} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & -I \end{bmatrix} \begin{bmatrix} \widetilde{y_\ell} \\ y_r \end{bmatrix}, \tag{13.32}$$

but, of course, we may replace the coupling matrix by any other nonsingular interface coupling matrix C:

$$C \equiv \left[\begin{array}{cc} C_{\ell\ell} & -C_{\ell r} \\ -C_{r\ell} & C_{rr} \end{array} \right]. \tag{13.33}$$

This leads to the following block system

$$\mathbf{B}_C \widetilde{\mathbf{y}} = \left[\begin{array}{ccc|ccc} \mathbf{B}_{11} & \mathbf{B}_{1\ell} & \mathbf{B}_{1r} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{B}_{\ell 1} & B_{\ell\ell} & B_{\ell r} & 0 & 0 & \mathbf{0} \\ \mathbf{0} & C_{\ell\ell} & C_{\ell r} & -C_{\ell\ell} & -C_{\ell r} & \mathbf{0} \\ \hline \mathbf{0} & -C_{r\ell} & -C_{rr} & C_{r\ell} & C_{rr} & \mathbf{0} \\ \mathbf{0} & 0 & 0 & B_{r\ell} & B_{rr} & \mathbf{B}_{r2} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{B}_{2\ell} & \mathbf{B}_{2r} & \mathbf{B}_{22} \end{array} \right] \left[\begin{array}{c} \mathbf{y}_1 \\ y_\ell \\ \widetilde{y}_r \\ \widetilde{y}_\ell \\ y_r \\ \mathbf{y}_2 \end{array} \right] = \underline{\mathbf{d}}. \tag{13.34}$$

In a domain decomposition context, we will have for the approximate solution $\mathbf{y}$ that $\widetilde{y}_r \approx y_r$ and $\widetilde{y}_\ell \approx y_\ell$. If we know some analytic properties about the local behavior of the true solution $\mathbf{y}$ accross the interface, for instance, smoothness up to some degree, then we may try to identify a convenient coupling matrix C that takes advantage of this knowledge. We want preferably a C so that

$$\begin{aligned} -C_{\ell\ell}\widetilde{y}_\ell - C_{\ell r} y_r &\approx -C_{\ell\ell} y_\ell - C_{\ell r} y_r \approx 0 \\ \text{and} \quad - C_{r\ell} y_\ell - C_{rr}\widetilde{y}_r &\approx -C_{r\ell} y_\ell - C_{rr} y_r \approx 0. \end{aligned}$$

In that case (13.34) is almost decoupled into two independent smaller linear systems (identified by the two boxes).

The matrix consisting of the two blocks is then used as a preconditioner for the enhanced system. Tan [164, Chapter 2&3] (see also [165]) 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 in [164]. Some attention is paid to the case where the solutions for the subdomains are obtained with only modest accuracy per iteration step.

13.9.3 Other approaches

Washio and Hayami [189] 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. 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 in [189] for a block modified ILU preconditioner. Experimental results have been reported for a 32-processor NEC Cenju distributed memory computer.

Radicati and Robert [144] 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 taken as the average of the two values computed by the overlapping ILU factors. The approach of Radicati and Robert has been further refined by de Sturler [49], 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 [49] (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 a twisted 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% (this 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). Since twisting leads to more parallelism, one can use bigger blocks (which usually means a better approximation). This helps to explain the good results.

Haase [101] suggests constructing an incomplete Choleski decomposition on each subdomain and modifying the decomposition using information from neighbouring subdomains. His results, for the discretized Poisson equation in 3D, show that an increase in the number of domains scarcely affects the effectiveness of the preconditioner. Experimental results for a realistic finite-element model, on a 16-processor Parsytec Xplorer, show very good scalability of the Conjugate Gradient method with this preconditioner.

Heisse and Jung [105] attempt to improve the effectiveness of a domain decomposition preconditioner by using a multigrid V-cycle with only one pre- and one post-smoothing step of a parallel variant of Gauss-Seidel type to solve a coarse grid approximation to the problem. With the usual domain decomposition technique, effects of local changes in a domain that lead to global changes in the solution travel only one layer of neigbouring domains far per iteration. The coarse grid corrections are used to get this globally relevant information quicker to all domains. The combination with Conjugate Gradients, which is the underlying method used for the local subproblems, leads to good results on a variety of platforms, including a 64-processor

For general systems, one could apply a block Jacobi preconditioning to the normal equations which would result in the block Cimmino algorithm [4]. A similar relationship exists between a block SOR preconditioning and the block Kaczmarz algorithm [27]. Block preconditioning for symmetric systems is discussed in [39]; in [41] 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 [34]. In fact, it is possible to have a significant effect on the convergence of these methods just by permuting the matrix to put large entries on the diagonal and then scaling it to reduce the magnitude of off-diagonal entries [63].

13.10 General Remarks

The history of parallel preconditioners is one with ups and downs. It is very difficult to construct efficient preconditioners for a given class of problems and even more so if they also have to be parallel. Initially, the attempts for parallelism were focussed on minimal changes to existing effective preconditioners (like ILU), or to exploiting the matrix A itself (polynomial preconditioners). These attempts have sometimes led to preconditioners that could be vectorized (hyperplane, etc), but for parallelism they are too fine grained. Polynomial preconditioning reduces the proportion of inner product and vector operations but at the penalty of increasing the number of iterations. It is thus usually only useful for overhead rich methods like GMRES on platforms where operations like inner products are relatively expensive with respect to the matrix-vector product. More recently there has been a shift of focus towards methods that offer more coarse-grained parallelism: domain decomposition methods and sparse approximate inverses. Despite some partial success, the sparse approximate inverses are still in their infancy. The domain decomposition techniques have proven to be successful for large classes of PDE related problems.

Bibliography

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

[2] F. L. Alvarado and H. Da g. Incomplete partitioned inverse preconditioners. Technical report, Department of Electrical and Computer Engineering, University of Wisconsin, Madison, 1994.

[3] F. L. Alvarado and R. Schreiber. Optimal parallel solution of sparse triangular systems. *SIAM J. Sci. Comput.*, 14:446–460, 1993.

[4] M. Arioli, I. S. Duff, J. Noailles, and D. Ruiz. A block projection method for sparse equations. *SIAM J. Sci. Statist. Comput.*, 13:47–70, 1992.

[5] M. Arioli, I. S. Duff, and D. Ruiz. Stopping criteria for iterative solvers. *SIAM J. Matrix Anal. Applic.*, 13:138–144, 1992.

[6] W. E. Arnoldi. The principle of minimized iteration in the solution of the matrix eigenproblem. *Quart. Appl. Math.*, 9:17–29, 1951.

[7] S. F. Ashby. Minimax polynomial preconditioning for Hermitian linear systems. *SIAM J. Matrix Anal. Applic.*, 12:766–789, 1991.

[8] O. Axelsson. Solution of linear systems of equations: iterative methods. In V. A. Barker, editor, *Sparse Matrix Techniques*, pages 1–51, Berlin, 1977. Copenhagen 1976, Springer Verlag.

[9] O. Axelsson. Conjugate gradient type methods for unsymmetric and inconsistent systems of equations. *Linear Alg. Appl.*, 29:1–16, 1980.

[10] O. Axelsson. *Iterative Solution Methods.* Cambridge University Press, Cambridge, 1994.

[11] O. Axelsson and G. Lindskog. On the eigenvalue distribution of a class of preconditioning methods. *Numer. Math.*, 48:479–498, 1986.

[12] O. Axelsson and P. S. Vassilevski. A black box generalized conjugate gradient solver with inner iterations and variable-step preconditioning. *SIAM J. Matrix Anal. Applic.*, 12(4):625–644, 1991.

[13] Z. Bai and J. W. Demmel. Using the matrix *sign* function to compute invariant subspaces. *SIAM J. Matrix Anal. Applic.*, 1998:205–225, 1998.

[14] Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. *IMA J. Numer. Anal.*, 14:563–581, 1991.

[15] R. E. Bank and T. F. Chan. An analysis of the composite step biconjugate gradient method. *Numer. Math.*, 66:295–319, 1993.

[16] S. T. Barnard, L. M. Bernardo, and H. D. Simon. An MPI implementation of the SPAI preconditioner on the t3E. Technical Report Technical Report LBNL-40794 UC405, Lawrence Berkeley National Laboratory, 1997.

[17] S. T. Barnard and R. L. Clay. A portable MPI implementation of the SPAI preconditioner in ISIS++. In M. Heath et al, editor, *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, 1997.

[18] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods.* SIAM, Philadelphia, PA, 1994.

[19] P. Bastian and G. Horton. Parallelization of robust multigrid methods: ILU factorization and frequency decomposition method. *SIAM J. Sci. Statist. Comput.*, 6:1457–1470, 1991.

[20] M. Benzi, J. Marín, and M. Tůma. A two-level parallel preconditioner based on sparse approximate inverses. In D. R. Kincaid and A. C. Elster, editors, *Iterative Methods in Scientific Computing, II*, pages 167–178. IMACS, 1999.

[21] M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17:1135–1149, 1996.

[22] M. Benzi and M. Tůma. Numerical experiments with two sparse approximate inverse preconditioners. *BIT*, 38:234–241, 1998.

[23] M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for non-symmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994, 1998.

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

[25] A. Björck and T. Elfving. Accelerated projection methods for computing pseudo-inverse solutions of systems of linear equations. *BIT*, 19:145–163, 1979.

[26] C. W. Bomhof and H. A. van der Vorst. A parallel linear system solver for circuit-simulation problems. *Num. Lin. Alg. Appl.*, 7:649–665, 2000.

[27] R. Bramley and A. Sameh. Row projection methods for large nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:168–193, 1992.

[28] C. Brezinski and M. Redivo-Zaglia. Treatment of near breakdown in the CGS algorithm. *Numerical Algorithms*, 7:33–73, 1994.

[29] A.M. Bruaset. *A Survey of Preconditioned Iterative Methods.* Longman Scientific & and Technical, Harlow, UK, 1995.

[30] G. Brussino and V. Sonnad. A comparison of direct and preconditioned iterative techniques for sparse unsymmetric systems of linear equations. *Int. J. for Num. Methods in Eng.*, 28:801–815, 1989.

[31] B. L. Buzbee, G. H. Golub, and C W. Nielson. On direct methods for solving Poisson's equations. *SIAM J. Numer. Anal.*, 7:627–656, 1970.

[32] T. F. Chan and D. Goovaerts. A note on the efficiency of domain decomposed incomplete factorizations. *SIAM J. Sci. Statist. Comput.*, 11:794–803, 1990.

[33] T. F. Chan and H. A. van der Vorst. Approximate and incomplete factorizations. In D. E. Keyes, A. Sameh, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms*, ICASE/LaRC Interdisciplinary Series in Science and Engineering, pages 167–202. Kluwer, Dordrecht, 1997.

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

[35] E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM J. Sci. Comput.*, 19:995–1023, 1998.

[36] A. T. Chronopoulos and C. W. Gear. s-Step iterative methods for symmetric linear systems. *J. Comp. and Appl. Math.*, 25:153–168, 1989.

[37] A. T. Chronopoulos and S. K. Kim. s-Step Orthomin and GMRES implemented on parallel computers. Technical Report 90/43R, UMSI, Minneapolis, 1990.

[38] P. Concus and G. H. Golub. A generalized Conjugate Gradient method for nonsymmetric systems of linear equations. Technical Report STAN-CS-76-535, Stanford University, Stanford, CA, 1976.

[39] P. Concus, G. H. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 6:220–252, 1985.

[40] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In J. R. Bunch and D. J. Rose, editors, *Sparse Matrix Computations*. Academic Press, New York, 1976.

[41] P. Concus and G. Meurant. On computing INV block preconditionings for the conjugate gradient method. *BIT*, pages 493–504, 1986.

[42] J. D. F. Cosgrove, J. C. Diaz, and A. Griewank. Approximate inverse preconditionings for sparse linear systems. *Intern. J. Computer Math.*, 44:91–110, 1992.

[43] G. C. (Lianne) Crone. The conjugate gradient method on the parsytec GCel-3/512. *FGCS*, 11:161–166, 1995.

[44] L. Crone and H. van der Vorst. Communication aspects of the conjugate gradient method on distributed-memory machines. *Supercomputer*, X(6):4–9, 1993.

[45] J. Cullum and A. Greenbaum. Relations between Galerkin and norm-minimizing iterative methods for solving linear systems. *SIAM J. Matrix Anal. Applic.*, 17:223–247, 1996.

[46] J. W. Daniel, W. B. Gragg, L. Kaufmann, and G. W. Stewart. Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Math. Comp.*, 30:772–795, 1976.

[47] E. de Sturler. A parallel restructured version of GMRES(m). Technical Report 91-85, Delft University of Technology, Delft, 1991.

[48] E. de Sturler. A parallel variant of GMRES(m). In R. Miller, editor, *Proc. of the fifth Int. Symp. on Numer. Methods in Eng.*, 1991.

[49] E. de Sturler. *Iterative methods on distributed memory computers.* PhD thesis, Delft University of Technology, Delft, the Netherlands, 1994.

[50] E. de Sturler and D. R. Fokkema. Nested Krylov methods and preserving the orthogonality. In N. Duane Melson, T. A. Manteuffel, and S. F. McCormick, editors, *Sixth Copper Mountain Conference on Multigrid Methods*, volume Part 1 of *NASA Conference Publication 3324*, pages 111–126. NASA, 1993.

[51] E. de Sturler and H.A. van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Applied Numerical Mathematics*, 18:441–459, 1995.

[52] J. Demmel, M. Heath, and H. van der Vorst. Parallel numerical linear algebra. In *Acta Numerica 1993.* Cambridge University Press, Cambridge, 1993.

[53] S. Doi. On parallelism and convergence of incomplete LU factorizations. *Applied Numerical Mathematics*, 7:417–436, 1991.

[54] S. Doi and A. Hoshi. Large numbered multicolor MILU preconditioning on SX-3/14. *Int'l J. Computer Math.*, 44:143–152, 1992.

[55] J. J. Dongarra. Performance of various computers using standard linear equations software in a fortran environment. Technical Report CS-89-85, University of Tennessee, Knoxville, 1990.

[56] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers.* SIAM, Philadelphia, PA, 1991.

[57] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. SIAM, Philadelphia, PA, 1998.

[58] J. J. Dongarra, V. Eijkhout, and H. A. van der Vorst. Iterative solver benchmark. Technical report, University of Tennessee, 2000.

[59] Jack J. Dongarra and Henk A. van der Vorst. Performance of various computers using standard sparse linear equations solving techniques. *Supercomputer*, 9(5):17–29, 1992.

[60] J. Drkosova, A. Greenbaum, M. Rozloznik, and Z. Strakos. Numerical stability of GMRES. *BIT*, 35:309–330, 1995.

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

[62] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices*. Oxford University Press, London, 1986.

[63] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Applic.*, 20:889–901, 1999.

[64] I. S. Duff and G. A. Meurant. The effect of ordering on preconditioned conjugate gradient. *BIT*, 29:635–657, 1989.

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

[66] T. Dupont, R. P. Kendall, and H. H. Rachford Jr. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM J. Numer. Anal.*, 5(3):559–573, 1968.

[67] V. Eijkhout. Beware of unperturbed modified incomplete point factorizations. In R. Beauwens and P. de Groen, editors, *Iterative Methods in Linear Algebra*, pages 583–591, Amsterdam, 1992. IMACS Int. Symp., Brussels, Belgium, 2-4 April, 1991, North-Holland.

[68] T. Eirola and O. Nevanlinna. Accelerating with rank-one updates. *Linear Alg. Appl.*, 121:511–520, 1989.

[69] S. C. Eisenstat. Efficient implementation of a class of preconditioned conjugate gradient methods. *SIAM J. Sci. Statist. Comput.*, 2(1):1–4, 1981.

[70] H. C. Elman. *Iterative methods for large sparse nonsymmetric systems of linear equations*. PhD thesis, Yale University, New Haven, CT, 1982.

[71] M. Engeli, T. Ginsburg, H. Rutishauser, and E. Stiefel. *Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems*. Birkhäuser, Basel/Stuttgart, 1959.

[72] V. Faber and T. A. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM J. Numer. Anal.*, 21(2):352–362, 1984.

[73] K. Fan. Note on m-matrices. *Quart. J. Math. Oxford Ser. (2)*, 11:43–49, 1960.

[74] B. Fischer. *Orthogonal Polynomials and Polynomial Based Iteration Methods for Indefinite Linear Systems*. PhD thesis, University of Hamburg, Hamburg, Germany, 1994.

[75] B. Fischer. *Polynomial based iteration methods for symmetric linear systems*. Advances in Numerical Mathematics. Wiley and Teubner, Chichester, Stuttgart, 1996.

[76] R. Fletcher. *Conjugate gradient methods for indefinite systems*, volume 506 of *Lecture Notes Math.*, pages 73–89. Springer-Verlag, Berlin–Heidelberg–New York, 1976.

[77] D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst. Accelerated inexact Newton schemes for large systems of nonlinear equations. *SIAM J. Sci. Comput.*, 19:657–674, 1998.

[78] R. W. Freund, G. H. Golub, and N. M. Nachtigal. Iterative solution of linear systems. In *Acta Numerica 1992*. Cambridge University Press, Cambridge, 1992.

[79] R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices. *SIAM J. Sci. Comput.*, 14:137–158, 1993.

[80] R. W. Freund and N. M. Nachtigal. An implementation of the look-ahead Lanczos algorithm for non-Hermitian matrices, part 2. Technical Report 90.46, RIACS, NASA Ames Research Center, 1990.

[81] R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numer. Math.*, 60:315–339, 1991.

[82] Roland Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14:470–482, 1993.

[83] E. Gallopoulos and Y. Saad. Efficient solution of parabolic equations by Krylov approximation methods. *SIAM J. Sci. Statist. Comput.*, 13:1236–1264, 1992.

[84] C. F. Gauss. *Werke, Band IX*. Teubner, Leipzig, 1903.

[85] T. Ginsburg. Contribution I/5: The Conjugate Gradient method. In J. H. Wilkinson and C. Reinsch, editors, *Handbook of Automatic Computation, Linear Algebra, Vol II*. Springer Verlag, Berlin, etc., 1971.

[86] G. H. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *SIAM J. Numer. Anal.*, 2:205–224, 1965.

[87] G. H. Golub and D. P. O'Leary. Some history of the conjugate gradient and Lanczos algorithms: 1948-1976. *SIAM Rev.*, 31:50–102, 1989.

[88] G. H. Golub and C. F. Van Loan. *Matrix Computations*. North Oxford Academic, Oxford, 1983.

[89] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1989.

[90] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 1996.

[91] N. I. M. Gould and J. A. Scott. Sparse approximate-inverse preconditioners using norm-minimization techniques. *SIAM J. Sci. Comput.*, 19(2):605–625, 1998.

[92] A. Greenbaum. Estimating the attainable accuracy of recursively computed residual methods. *SIAM J. Matrix Anal. Applic.*, 18:535–551, 1997.

[93] A. Greenbaum. *Iterative Methods for Solving Linear Systems.* SIAM, Philadelphia, 1997.

[94] M. Grote and H. Simon. Parallel preconditioning and approximate inverses on the connection machine. In R. F. Sincovec, D. E. Keyes, M. R. Leuze, L. R. Petzold, and D. A. Reed, editors, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, pages 519–523, Philadelphia, 1993. SIAM.

[95] M. J. Grote and T. Huckle. Parallel preconditionings with sparse approximate inverses. *SIAM J. Sci. Comput.*, 18:838–853, 1997.

[96] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[97] I. Gustafsson and G. Lindskog. A preconditioning technique based on element matrix factorizations. *Comput. Methods Appl. Mech. Eng.*, 55:201–220, 1986.

[98] I. Gustafsson and G. Lindskog. Completely parallelizable preconditioning methods. *Num. Lin. Alg. Appl.*, 2:447–465, 1995.

[99] M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms, Part I. *SIAM J. Matrix Anal. Applic.*, 13:594–639, 1992.

[100] M. H. Gutknecht. Variants of BICGSTAB for matrices with complex spectrum. *SIAM J. Sci. Comput.*, 14:1020–1033, 1993.

[101] G. Haase. Parallel incomplete Cholesky preconditioners based on the nonoverlapping data distribution. *Parallel Computing*, 24:1685–1703, 1998.

[102] W. Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* Teubner, Stuttgart, 1991.

[103] W. Hackbusch. *Elliptic Differential Equations.* Springer-Verlag, Berlin, 1992.

[104] L. A. Hageman and D. M. Young. *Applied Iterative Methods.* Academic Press, New York, 1981.

[105] B. Heisse and M. Jung. Parallel solvers for nonlinear elliptic problems based on domain decomposition ideas. *Parallel Computing*, 22:1527–1544, 1997.

[106] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Natl. Bur. Stand.*, 49:409–436, 1954.

[107] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, 1996.

[108] M. Hochbruck and C. Lubich. On Krylov subspace approximations to the matrix exponential operator. *SIAM J. Numer. Anal.*, 34:1911–1925, 1997.

[109] T. J. R. Hughes, I. Levit, and J. Winget. An element-by-element solution algorithm for problems of structural and solid mechanics. *J. Comp. Methods in Appl. Mech. Eng.*, 36:241–254, 1983.

[110] C. P. Jackson and P. C. Robinson. A numerical study of various algorithms related to the preconditioned Conjugate Gradient method. *Int. J. for Num. Meth. in Eng.*, 21:1315–1338, 1985.

[111] D. A. H. Jacobs. Preconditioned Conjugate Gradient methods for solving systems of algebraic equations. Technical Report RD/L/N 193/80, Central Electricity Research Laboratories, 1981.

[112] K. C. Jea and D. M. Young. Generalized conjugate-gradient acceleration of nonsymmetrizable iterative methods. *Linear Alg. Appl.*, 34:159–194, 1980.

[113] O. G. Johnson, C. A. Micheli, and G. Paul. Polynomial preconditioning for conjugate gradient calculations. *SIAM J. Numer. Anal.*, 20:363–376, 1983.

[114] M. T. Jones and P. E. Plassmann. The efficient parallel iterative solution of large sparse linear systems. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computations*, IMA Vol 56. Springer Verlag, Berlin, 1994.

[115] E. F. Kaasschieter. A practical termination criterion for the Conjugate Gradient method. *BIT*, 28:308–322, 1988.

[116] L. Yu. Kolotilina and A. Yu. Yeremin. Factorized sparse approximate inverse preconditionings. *SIAM J. Matrix Anal. Applic.*, 14:45–58, 1993.

[117] J. C. C. Kuo and T. F. Chan. Two-color Fourier analysis of iterative algorithms for elliptic problems with red/black ordering. *SIAM J. Sci. Statist. Comput.*, 11:767–793, 1990.

[118] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natl. Bur. Stand*, 45:225–280, 1950.

[119] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand*, 49:33–53, 1952.

[120] M. Magolu monga Made and H. A. van der Vorst. A generalized domain decomposition paradigm for parallel incomplete LU factorization preconditionings. *Future Generation Computer Systems*, 2001. in press.

[121] M. Magolu monga Made and H. A. van der Vorst. Parallel incomplete factorizations with pseudo-overlapped subdomains. *Parallel Computing*, 2001. in press.

[122] M. Magolu monga Made and H. A. van der Vorst. Spectral analysis of parallel incomplete factorizations with implicit pseudo-overlap. *Num. Lin. Alg. Appl.*, 2001. in press.

[123] K. Meerbergen and M. Sadkane. Using Krylov approximations to the matrix exponential operator in Davidson's method. *Applied Numerical Mathematics*, 31:331–351, 1999.

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

[125] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comp.*, 31:148–162, 1977.

[126] J. A. Meijerink and H. A. van der Vorst. Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems. *J.of Comp. Physics*, 44:134–155, 1981.

[127] G. Meurant. The block preconditioned conjugate gradient method on vector computers. *BIT*, 24:623–633, 1984.

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

[129] G. Meurant. The conjugate gradient method on vector and parallel supercomputers. Technical Report CTAC-89, University of Brisbane, July 1989.

[130] N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen. How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Applic.*, 13:778–795, 1992.

[131] H. Neuberger. The overlap Dirac operator. In B. Medeke A. Frommer, Th. Lippert and K. Schilling, editors, *Numerical Challenges in Lattice Quantum Chromodynamics*, pages 1–17, Berlin, 2000. Proceedings of the Interdisciplinary Workshop on Numerical Challenges in Lattice QCD, Wuppertal, August 22-24, 1999, Springer Verlag. Lecture notes in Computational Science and Engineering, 15.

[132] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems.* Plenum Press, New York and London, 1988.

[133] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *J. Inst. Math. Appl.*, 10:373–381, 1972.

[134] C. C. Paige. Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. *J. Inst. Math. Appl.*, 18:341–349, 1976.

[135] C. C. Paige. Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Linear Alg. Appl.*, 34:235–258, 1980.

[136] C. C. Paige, B. N. Parlett, and H. A. van der Vorst. Approximate solutions and eigenvalue bounds from Krylov subspaces. *Num. Lin. Alg. Appl.*, 2(2):115–134, 1995.

[137] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12:617–629, 1975.

[138] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8:43–71, 1982.

[139] B. N. Parlett. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, N.J., 1980.

[140] B. N. Parlett, D. R. Taylor, and Z. A. Liu. A look-ahead Lanczos algorithm for unsymmetric matrices. *Math. Comp.*, 44:105–124, 1985.

[141] C. Pommerell. *Solution of large unsymmetric systems of linear equations.* PhD thesis, Swiss Federal Institute of Technology, Zürich, 1992.

[142] C. Pommerell and W. Fichtner. PILS: An iterative linear solver package for ill-conditioned systems. In *Supercomputing '91*, pages 588–599, Los Alamitos, CA., 1991. IEEE Computer Society.

[143] A. Quarteroni and A. Valli. *Numerical Approximation of Partial Differential Equations.* Springer Verlag, Berlin, 1994.

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

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

[146] J. Roberts. Linear model reduction and solution of algebraic Riccati equations. *Inter. J. Control*, 32:677–687, 1980.

[147] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Statist. Comput.*, 6:865–881, 1985.

[148] Y. Saad. Krylov subspace methods on supercomputers. Technical report, RIACS, Moffett Field, CA, September 1988.

[149] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14:461–469, 1993.

[150] Y. Saad. *Iterative Methods for Sparse Linear Systems.* PWS Publishing Company, Boston, 1996.

[151] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[152] J. J. F. M. Schlichting and H. A. van der Vorst. Solving 3D block bidiagonal linear systems on vector computers. *J. Comp. and Appl. Math.*, 27:323–330, 1989.

[153] A. Sidi. Efficient implementation of minimal polynomial and reduced rank extrapolation methods. *J. Comp. and Appl. Math.*, 36:305–337, 1991.

[154] Horst D. Simon. Direct sparse matrix methods. In James C. Almond and David M. Young, editors, *Modern Numerical Algorithms for Supercomputers*, pages 325–444, Austin, 1989. The University of Texas at Austin, Center for High Performance Computing.

[155] G. L. G. Sleijpen and D. R. Fokkema. BICGSTAB(ℓ) for linear equations involving unsymmetric matrices with complex spectrum. *ETNA*, 1:11–32, 1993.

[156] G. L. G. Sleijpen and H. A. van der Vorst. Reliable updated residuals in hybrid Bi-CG methods. *Computing*, 56:141–163, 1996.

[157] G. L. G. Sleijpen, H. A. van der Vorst, and J. Modersitzki. The main effects of rounding errors in Krylov solvers for symmetric linear systems. Technical Report Preprint 1006, Utrecht University, Department of Mathematics, 1997.

[158] G. L. G. Sleijpen, H. A. van der Vorst, and J. Modersitzki. Differences in the effects of rounding errors in Krylov solvers for symmetric indefinite linear systems. *simax*, 22(3):726–751, 2000.

[159] G. L. G. Sleijpen, H.A. van der Vorst, and D. R. Fokkema. Bi-CGSTAB(ℓ) and other hybrid Bi-CG methods. *Numerical Algorithms*, 7:75–109, 1994.

[160] P. Sonneveld. CGS: a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 10:36–52, 1989.

[161] G. W. Stewart. *Matrix Algorithms: Vol. II Eigensystems.* SIAM, Philadelphia, 2001.

[162] H. S. Stone. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM J. Numer. Anal.*, 5:530–558, 1968.

[163] R. Sweet. A parallel and vector variant of the cyclic reduction algorithm. *Supercomputer*, 22:18–25, 1987.

[164] K. H. Tan. *Local coupling in domain decomposition.* PhD thesis, Utrecht University, Utrecht, the Netherlands, 1995.

[165] K. H. Tan and M. J. A. Borsboom. On generalized Schwarz coupling applied to advection-dominated problems. In *Domain decomposition methods in scientific and engineering computing*, pages 125–130, Providence, RI, 1993. Amer. Math. Soc.

[166] W. P. Tang. Generalized Schwarz splitting. *SIAM J. Sci. Statist. Comput.*, 13:573–595, 1992.

[167] C. H. Tong and Q. Ye. Analysis of the finite precision Bi-Conjugate Gradient algorithm for nonsymmetric linear systems. *Math. Comp.*, 69:1559–1575, 2000.

[168] A. van der Sluis and H. A. van der Vorst. The rate of convergence of conjugate gradients. *Numer. Math.*, 48:543–560, 1986.

[169] A. van der Sluis and H. A. van der Vorst. Numerical solution of large sparse linear algebraic systems arising from tomographic problems. In G. Nolet, editor, *Seismic Tomography*, chapter 3, pages 49–83. Reidel Pub. Comp., Dordrecht, 1987.

[170] A. van der Sluis and H. A. van der Vorst. SIRT- and CG-type methods for the iterative solution of sparse linear least-squares problems. *Linear Alg. Appl.*, 130:257–302, 1990.

[171] H. A. van der Vorst. A vectorizable variant of some ICCG methods. *SIAM J. Sci. Statist. Comput.*, 3:86–92, 1982.

[172] H. A. van der Vorst. An iterative solution method for solving $f(a)x = b$, using Krylov subspace information obtained for the symmetric positive definite matrix a. *J. Comp. and Appl. Math.*, 18:249–263, 1987.

[173] H. A. van der Vorst. Large tridiagonal and block tridiagonal linear systems on vector and parallel computers. *Parallel Computing*, 5:45–54, 1987.

[174] H. A. van der Vorst. High performance preconditioning. *SIAM J. Sci. Statist. Comput.*, 10:1174–1185, 1989.

[175] H. A. van der Vorst. ICCG and related methods for 3D problems on vector computers. *Computer Physics Communications*, 53:223–235, 1989.

[176] H. A. van der Vorst. The convergence behaviour of preconditioned CG and CG-S in the presence of rounding errors. In O. Axelsson and L. Yu. Kolotilina, editors, *Preconditioned Conjugate Gradient Methods*, Berlin, 1990. Nijmegen 1989, Springer Verlag. Lecture Notes in Mathematics 1457.

[177] H. A. van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13:631–644, 1992.

[178] H. A. van der Vorst. Conjugate gradient type methods for nonsymmetric linear systems. In R. Beauwens and P. de Groen, editors, *Iterative Methods in Linear Algebra*, pages 67–76, Amsterdam, 1992. IMACS Int. Symp., Brussels, Belgium, 2-4 April, 1991, North-Holland.

[179] H. A. van der Vorst and C. Vuik. The superlinear convergence behaviour of GMRES. *J. Comp. and Appl. Math.*, 48:327–341, 1993.

[180] H. A. van der Vorst and C. Vuik. GMRESR: A family of nested GMRES methods. *Num. Lin. Alg. Appl.*, 1:369–386, 1994.

[181] H. A. van der Vorst and Q. Ye. Refined residual replacement techniques for subspace iterative methods for convergence of true residuals. *SIAM J. Sci. Comput.*, 22:835–852, 2000.

[182] A. C. N. van Duin. *Parallel Sparse Matrix Computations*. PhD thesis, Utrecht University, Utrecht, The Netherlands, 1998.

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

[184] R. S. Varga. Factorizations and normalized iterative methods. In R. E. Langer, editor, *Boundary Problems in Differential Equations*, pages 121–142, Madison, WI, 1960. University of Wisconsin Press.

[185] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs N.J., 1962.

[186] P. K. W. Vinsome. ORTOMIN: an iterative method for solving sparse sets of simultaneous linear equations. In *Proc.Fourth Symposium on Reservoir Simulation*, pages 149–159. Society of Petroleum Engineers of AIME, 1976.

[187] C. Vuik, R. R. P. van Nooyen, and P. Wesseling. Parallelism in ILU-preconditioned GMRES. *Parallel Computing*, 24:1927–1946, 1998.

[188] H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Statist. Comput.*, 9:152–163, 1988.

[189] T. Washio and K. Hayami. Parallel block preconditioning based on SSOR and MILU. *Num. Lin. Alg. Appl.*, 1:533–553, 1994.

[190] R. Weiss. Error-minimizing Krylov subspace methods. *SIAM J. Sci. Comput.*, 15:511–527, 1994.

[191] R. Weiss. *Parameter-Free Iterative Linear Solvers.* Akademie Verlag, Berlin, 1996.

[192] P. Wesseling. *An Introduction to Multigrid Methods.* John Wiley and Sons, Chichester, 1992.

[193] O. Widlund. A Lanczos method for a class of nonsymmetric systems of linear equations. *SIAM J. Numer. Anal.*, 15:801–812, 1978.

[194] J. H. Wilkinson and C. Reinsch, editors. *Handbook for Automatic Computation, Vol 2, Linear Algebra.* Springer Verlag, Heidelberg – Berlin – New York, 1971.

[195] J. Zhang. A sparse approximate inverse technique for parallel preconditioning of general sparse matrices. Technical Report Technical Report 281-98, Department of Computer Science, University of Kentucky, KY, 1998.

[196] L. Zhou and H. F. Walker. Residual smoothing techniques for iterative methods. *SIAM J. Sci. Statist. Comput.*, 15:297–312, 1994.

Index