CHAPTER

12

PARALLEL PRECONDITIONERS

This chapter covers a few alternative methods for preconditioning a linear system. These methods are suitable when the desired goal is to maximize parallelism. The simplest approach is the diagonal (or Jacobi) preconditioning. Often, this preconditioner is not very useful, since the number of iterations of the resulting iteration tends to be much larger than the more standard variants, such as ILU or SSOR. When developing parallel preconditioners, one should beware that the benefits of increased parallelism are not outweighed by the increased amount of computations. The main question to ask is whether or not it is possible to find preconditioning techniques that have a high degree of parallelism, as well as good intrinsic qualities.

12.1 INTRODUCTION

As seen in the previous chapter, a limited amount of parallelism can be extracted from the standard preconditioners such as ILU and SSOR. Fortunately, a number of alternative techniques can be developed that are specifically targeted at parallel environments. These are preconditioning techniques that would normally not be used on a standard machine, but only for parallel computers. There are at least three such types of techniques discussed in this chapter. The simplest approach is to use a Jacobi or, even better, a block Jacobi approach. In the simplest case, a Jacobi preconditioner may consist of the diagonal or a block-diagonal of A. To enhance performance, these preconditioners can themselves be accelerated by polynomial iterations, i.e., a second level of preconditioning called *polynomial preconditioning*.

A different strategy altogether is to enhance parallelism by using graph theory algorithms, such as graph-coloring techniques. These consist of coloring nodes such that two adjacent nodes have different colors. The gist of this approach is that all unknowns associated with the same color can be determined simultaneously in the forward and backward sweeps of the ILU preconditioning operation.

Finally, a third strategy uses generalizations of "partitioning" techniques, which can be put in the general framework of "domain decomposition" approaches. These will be covered in detail in the next chapter.

Algorithms are emphasized rather than implementations. There are essentially two types of algorithms, namely, those which can be termed *coarse-grain* and those which can be termed *fine-grain*. In coarse-grain algorithms, the parallel tasks are relatively big and may, for example, involve the solution of small linear systems. In fine-grain parallelism, the subtasks can be elementary floating-point operations or consist of a few such operations. As always, the dividing line between the two classes of algorithms is somewhat blurred.

12.2 BLOCK-JACOBI PRECONDITIONERS

Overlapping block-Jacobi preconditioning consists of a general block-Jacobi approach as described in Chapter 4, in which the sets $\mathcal{S}_i$ overlap. Thus, we define the index sets

$$\mathcal{S}_i = \{j \mid l_i \le j \le r_i\}$$

with

$$\begin{aligned} l_1 &= 1 \\ r_p &= n \\ r_i &> l_{i+1}, \quad 1 \le i \le p-1 \end{aligned}$$

where p is the number of blocks. Now use the block-Jacobi method with this particular partitioning, or employ the general framework of additive projection processes of Chapter 5, and use an additive projection method onto the sequence of subspaces

$$K_i = \text{span}\{V_i\}, \quad V_i = [e_{l_i}, e_{l_i+1}, \ldots, e_{r_i}].$$

Each of the blocks will give rise to a correction of the form

$$\xi_i^{(k+1)} = \xi_i^{(k)} + A_i^{-1} V_i^T (b - Ax^{(k)}). \tag{12.1}$$

One problem with the above formula is related to the overlapping portions of the x variables. The overlapping sections will receive two different corrections in general. According to the definition of "additive projection processes" seen in Chapter 5, the next iterate can be defined as

$$x_{k+1} = x_k + \sum_{i=1}^{p} V_i A_i^{-1} V_i^T r_k$$

where $r_k = b - Ax_k$ is the residual vector at the previous iteration. Thus, the corrections for the overlapping regions simply are added together. It is also possible to weigh these contributions before adding them up. This is equivalent to redefining (12.1) into

$$\xi_i^{(k+1)} = \xi_i^{(k)} + D_i A_i^{-1} V_i^T (b - Ax_k)$$

in which D_i is a nonnegative diagonal matrix of weights. It is typical to weigh a nonoverlapping contribution by one and an overlapping contribution by $1/k$ where k is the number of times the unknown is represented in the partitioning.

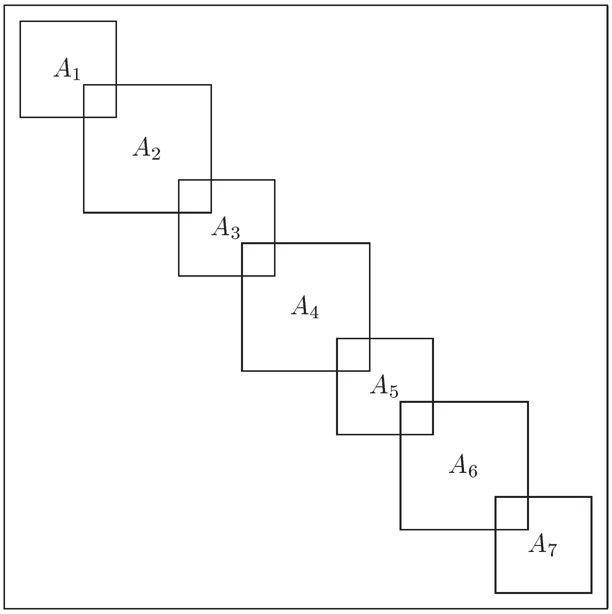


Figure 12.1 *The block-Jacobi matrix with overlapping blocks.*

The block-Jacobi iteration is often over- or under-relaxed, using a relaxation parameter ω. The iteration can be defined in the form

$$x_{k+1} = x_k + \sum_{i=1}^{p} \omega_i V_i A_i^{-1} V_i^T r_k.$$

Recall that the residual at step $k+1$ is then related to that at step k by

$$r_{k+1} = \left[I - \sum_{i=1}^{p} \omega_i A V_i \left(V_i^T A V_i \right)^{-1} V_i^T \right] r_k.$$

The solution of a sparse linear system is required at each projection step. These systems can be solved by direct methods if the subblocks are small enough. Otherwise, iterative methods may be used. The outer loop accelerator should then be a flexible variant, such as FGMRES, which can accommodate variations in the preconditioners.

12.3 POLYNOMIAL PRECONDITIONERS

In polynomial preconditioning the matrix M is defined by

$$M^{-1} = s(A)$$

where s is a polynomial, typically of low degree. Thus, the original system is replaced by the preconditioned system

$$s(A)Ax = s(A)b \tag{12.2}$$

which is then solved by a conjugate gradient-type technique. Note that $s(A)$ and A commute and, as a result, the preconditioned matrix is the same for right or left preconditioning. In addition, the matrix $s(A)$ or $As(A)$ does not need to be formed explicitly since $As(A)v$ can be computed for any vector v from a sequence of matrix-by-vector products.

Initially, this approach was motivated by the good performance of matrix-vector operations on vector computers for long vectors, e.g., the Cyber 205. However, the idea itself is an old one and has been suggested by Stiefel [204] for eigenvalue calculations in the mid 1950s. Next, some of the popular choices for the polynomial s are described.

12.3.1 NEUMANN POLYNOMIALS

The simplest polynomial s which has been used is the polynomial of the Neumann series expansion

$$I + N + N^2 + \cdots + N^s$$

in which

$$N = I - \omega A$$

and ω is a scaling parameter. The above series comes from expanding the inverse of ωA using the splitting

$$\omega A = I - (I - \omega A).$$

This approach can also be generalized by using a splitting of the form

$$\omega A = D - (D - \omega A)$$

where D can be the diagonal of A or, more appropriately, a block diagonal of A. Then,

$$\begin{aligned}(\omega A)^{-1} &= \left[D(I - (I - \omega D^{-1}A))\right]^{-1} \\ &= \left[I - (I - \omega D^{-1}A)\right]^{-1} D^{-1}.\end{aligned}$$

Thus, setting

$$N = I - \omega D^{-1}A$$

results in the approximate s-term expansion

$$(\omega A)^{-1} \approx M^{-1} \equiv [I + N + \cdots + N^s]\, D^{-1}. \tag{12.3}$$

Since $D^{-1}A = \omega^{-1}[I - N]$, note that

$$\begin{aligned} M^{-1}A &= [I + N + \cdots + N^s]\, D^{-1}A \\ &= \frac{1}{\omega}[I + N + \cdots + N^s](I - N) \\ &= \frac{1}{\omega}(I - N^{s+1}). \end{aligned}$$

The matrix operation with the preconditioned matrix can be difficult numerically for large s. If the original matrix is Symmetric Positive Definite, then $M^{-1}A$ is not symmetric, but it is self-adjoint with respect to the D-inner product; see Exercise 1.

12.3.2 CHEBYSHEV POLYNOMIALS

The polynomial s can be selected to be optimal in some sense, and this leads to the use of Chebyshev polynomials. The criterion that is used makes the preconditioned matrix $s(A)A$ as close as possible to the identity matrix in some sense. For example, the spectrum of the preconditioned matrix can be made as close as possible to that of the identity. Denoting by $\sigma(A)$ the spectrum of A, and by $\mathbb{P}_k$ the space of polynomials of degree not exceeding k, the following may be solved.

$$\begin{aligned} &\text{Find } s \in \mathbb{P}_k \text{ which minimizes:} \\ &\max_{\lambda \in \sigma(A)} |1 - \lambda s(\lambda)|. \end{aligned} \tag{12.4}$$

Unfortunately, this problem involves all the eigenvalues of A and is harder to solve than the original problem. Usually, problem (12.4) is replaced by the problem

$$\begin{aligned} &\text{Find } s \in \mathbb{P}_k \text{ which minimizes:} \\ &\max_{\lambda \in E} |1 - \lambda s(\lambda)|, \end{aligned} \tag{12.5}$$

which is obtained from replacing the set $\sigma(A)$ by some continuous set E that encloses it. Thus, a rough idea of the spectrum of the matrix A is needed. Consider first the particular case where A is Symmetric Positive Definite, in which case E can be taken to be an interval $[\alpha, \beta]$ containing the eigenvalues of A.

A variation of Theorem 6.4 is that for any real scalar γ such with $\gamma \le \alpha$, the minimum

$$\min_{p \in \mathbb{P}_k, p(\gamma)=1} \ \max_{t \in [\alpha,\beta]} |p(t)|$$

is reached for the shifted and scaled Chebyshev polynomial of the first kind,

$$\hat{C}_k(t) \equiv \frac{C_k\left(1 + 2\frac{\alpha - t}{\beta - \alpha}\right)}{C_k\left(1 + 2\frac{\alpha - \gamma}{\beta - \alpha}\right)}.$$

Of interest is the case where $\gamma = 0$ which gives the polynomial

$$T_k(t) \equiv \frac{1}{\sigma_k} C_k\left(\frac{\beta+\alpha-2t}{\beta-\alpha}\right) \quad \text{with} \quad \sigma_k \equiv C_k\left(\frac{\beta+\alpha}{\beta-\alpha}\right).$$

Denote the center and mid-width of the interval $[\alpha, \beta]$, respectively, by

$$\theta \equiv \frac{\beta+\alpha}{2}, \quad \delta \equiv \frac{\beta-\alpha}{2}.$$

Using these parameters instead of α, β, the above expressions then become

$$T_k(t) \equiv \frac{1}{\sigma_k} C_k\left(\frac{\theta-t}{\delta}\right) \quad \text{with} \quad \sigma_k \equiv C_k\left(\frac{\theta}{\delta}\right).$$

The three-term recurrence for the Chebyshev polynomials results in the following three-term recurrences:

$$\sigma_{k+1} = 2\,\frac{\theta}{\delta}\sigma_k - \sigma_{k-1},\ k = 1, 2\ \ldots,$$

with

$$\sigma_1 = \frac{\theta}{\delta}, \quad \sigma_0 = 1,$$

and

$$\begin{aligned} T_{k+1}(t) &\equiv \frac{1}{\sigma_{k+1}}\left[2\,\frac{\theta-t}{\delta}\sigma_k T_k(t) - \sigma_{k-1}T_{k-1}(t)\right] \\ &= \frac{\sigma_k}{\sigma_{k+1}}\left[2\,\frac{\theta-t}{\delta}T_k(t) - \frac{\sigma_{k-1}}{\sigma_k}T_{k-1}(t)\right], \quad k \geq 1, \end{aligned}$$

with

$$T_1(t) = 1 - \frac{t}{\theta}, \quad T_0(t) = 1.$$

Define

$$\rho_k \equiv \frac{\sigma_k}{\sigma_{k+1}}, \quad k = 1, 2, \ldots. \tag{12.6}$$

Note that the above recurrences can be put together as

$$\rho_k = \frac{1}{2\sigma_1 - \rho_{k-1}} \tag{12.7}$$

$$T_{k+1}(t) = \rho_k\left[2(\sigma_1 - \frac{t}{\delta})T_k(t) - \rho_{k-1}T_{k-1}(t)\right], \quad k \geq 1. \tag{12.8}$$

Observe that formulas (12.7–12.8) can be started at $k = 0$ provided we set $T_{-1} \equiv 0$ and $\rho_{-1} \equiv 0$, so that $\rho_0 = 1/(2\sigma_1)$.

The goal is to obtain an iteration that produces a residual vector of the form $r_{k+1} = T_{k+1}(A)r_0$ where T_k is the polynomial defined by the above recurrence. The difference between two successive residual vectors is given by

$$r_{k+1} - r_k = (T_{k+1}(A) - T_k(A))r_0.$$

The identity $1 = (2\sigma_1 - \rho_{k-1})\rho_k$ and the relations (12.8) yield

$$T_{k+1}(t) - T_k(t) = T_{k+1}(t) - (2\sigma_1 - \rho_{k-1})\rho_k T_k(t)$$

$$= \rho_k \left[-\frac{2t}{\delta} T_k(t) + \rho_{k-1}(T_k(t) - T_{k-1}(t)) \right].$$

As a result,

$$\frac{T_{k+1}(t) - T_k(t)}{t} = \rho_k \left[\rho_{k-1} \frac{T_k(t) - T_{k-1}(t)}{t} - \frac{2}{\delta} T_k(t) \right]. \tag{12.9}$$

Define

$$d_k \equiv x_{k+1} - x_k,$$

and note that $r_{k+1} - r_k = Ad_k$. If $x_{k+1} = x_0 + s_k(A)r_0$, then $r_{k+1} = (I - As_k(A))r_0$, and $d_k = A^{-1}(T_{k+1}(A) - T_k(A))r_0$. Therefore the relation (12.9) translates into the recurrence,

$$d_k = \rho_k \left[\rho_{k-1} d_{k-1} - \frac{2}{\delta} r_k \right].$$

Finally, the following algorithm is obtained.

ALGORITHM 12.1: Chebyshev Acceleration

1. $r_0 = b - Ax_0; \sigma_1 = \delta/\theta;$
2. $\rho_0 = 1/\sigma_1; d_0 = \frac{1}{\theta} r_0;$
3. *For* $k = 0, \ldots,$ *until convergence Do:*
4. $\quad x_{k+1} = x_k + d_k$
5. $\quad r_{k+1} = r_k - Ad_k$
6. $\quad \rho_{k+1} = (2\sigma_1 - \rho_k)^{-1};$
7. $\quad d_{k+1} = \rho_{k+1}\rho_k d_k - \frac{2\rho_{k+1}}{\delta} r_{k+1}$
8. *EndDo*

Lines 7 and 4 can also be recast into one single update of the form

$$x_{k+1} = x_k + \rho_k \left[\rho_{k-1}(x_k - x_{k-1}) - \frac{2}{\delta}(b - Ax_k) \right].$$

It can be shown that when $\alpha = \lambda_1$ and $\beta = \lambda_N$, the resulting preconditioned matrix minimizes the condition number of the preconditioned matrices of the form $As(A)$ over all polynomials s of degree $\leq k - 1$. However, when used in conjunction with the Conjugate Gradient method, it is observed that the polynomial which minimizes the total number of Conjugate Gradient iterations *is far from being the one which minimizes the condition number.* If instead of taking $\alpha = \lambda_1$ and $\beta = \lambda_N$, the interval $[\alpha, \beta]$ is chosen to be slightly inside the interval $[\lambda_1, \lambda_N]$, a much faster convergence might be achieved. The true optimal parameters, i.e., those that minimize the number of iterations of the polynomial preconditioned Conjugate Gradient method, are difficult to determine in practice.

There is a slight disadvantage to the approaches described above. The parameters α and β, which approximate the smallest and largest eigenvalues of A, are usually not available beforehand and must be obtained in some dynamic way. This may be a problem mainly because a software code based on Chebyshev acceleration could become quite complex.

To remedy this, one may ask whether the values provided by an application of Gershgorin's theorem can be used for α and β. Thus, in the symmetric case, the parameter α, which *estimates* the smallest eigenvalue of A, may be nonpositive even when A is a positive definite matrix. However, when $\alpha \leq 0$, the problem of minimizing (12.5) is not well defined, since it does not have a unique solution due to the non strict-convexity of the uniform norm. An alternative uses the L_2-norm on $[\alpha, \beta]$ with respect to some weight function $w(\lambda)$. This "least-squares" polynomials approach is considered next.

12.3.3 LEAST-SQUARES POLYNOMIALS

Consider the inner product on the space $\mathbb{P}_k$:

$$\langle p, q \rangle = \int_{\alpha}^{\beta} p(\lambda)q(\lambda)w(\lambda)d\lambda \tag{12.10}$$

where $w(\lambda)$ is some non-negative weight function on (α, β). Denote by $\|p\|_w$ and call w-norm, the 2-norm induced by this inner product.

We seek the polynomial s_{k-1} which minimizes

$$\|1 - \lambda s(\lambda)\|_w \tag{12.11}$$

over all polynomials s of degree $\leq k-1$. Call s_{k-1} the *least-squares iteration polynomial*, or simply the least-squares polynomial, and refer to $R_k(\lambda) \equiv 1 - \lambda s_{k-1}(\lambda)$ as the least-squares residual polynomial. A crucial observation is that the least squares polynomial is well defined for arbitrary values of α and β. Computing the polynomial $s_{k-1}(\lambda)$ is not a difficult task when the weight function w is suitably chosen.

Computation of the least-squares polynomials There are three ways to compute the least-squares polynomial s_k defined in the previous section. The first approach is to use an explicit formula for R_k, known as the kernel polynomials formula,

$$R_k(\lambda) = \frac{\sum_{i=0}^{k} q_i(0)q_i(\lambda)}{\sum_{i=0}^{k} q_i(0)^2} \tag{12.12}$$

in which the q_i's represent a sequence of polynomials orthogonal with respect to the weight function $w(\lambda)$. The second approach generates a three-term recurrence satisfied by the residual polynomials $R_k(\lambda)$. These polynomials are orthogonal with respect to the weight function $\lambda w(\lambda)$. From this three-term recurrence, we can proceed exactly as for the Chebyshev iteration to obtain a recurrence formula for the sequence of approximate solutions x_k. Finally, a third approach solves the Normal Equations associated with the minimization of (12.11), namely,

$$\langle 1 - \lambda s_{k-1}(\lambda), \lambda Q_j(\lambda) \rangle = 0, j = 0, 1, 2, \ldots, k-1$$

where $Q_j, j = 1, \ldots, k-1$ is any basis of the space P_{k-1} of polynomials of degree $\leq k-1$.

These three approaches can all be useful in different situations. For example, the first approach can be useful for computing least-squares polynomials of low degree explicitly. For high-degree polynomials, the last two approaches are preferable for their better numerical behavior. The second approach is restricted to the case where $\alpha \geq 0$, while the third is more general.

Since the degrees of the polynomial preconditioners are often low, e.g., not exceeding 5 or 10, we will give some details on the first formulation. Let $q_i(\lambda), i = 0, 1, \ldots, n, \ldots$, be the *orthonormal* polynomials with respect to $w(\lambda)$. It is known that the least-squares residual polynomial $R_k(\lambda)$ of degree k is determined by the kernel polynomials formula (12.12). To obtain $s_{k-1}(\lambda)$, simply notice that

$$\begin{aligned} s_{k-1}(\lambda) &= \frac{1 - R_k(\lambda)}{\lambda} \\ &= \frac{\sum_{i=0}^{k} q_i(0) t_i(\lambda)}{\sum_{i=0}^{k} q_i(0)^2}, \quad \text{with} \qquad (12.13) \\ t_i(\lambda) &= \frac{q_i(0) - q_i(\lambda)}{\lambda}. \qquad (12.14) \end{aligned}$$

This allows s_{k-1} to be computed as a linear combination of the polynomials $t_i(\lambda)$. Thus, we can obtain the desired least-squares polynomials from the sequence of orthogonal polynomials q_i which satisfy a three-term recurrence of the form:

$$\beta_{i+1} q_{i+1}(\lambda) = (\lambda - \alpha_i) q_i(\lambda) - \beta_i q_{i-1}(\lambda), i = 1, 2, \ldots.$$

From this, the following recurrence for the t_i's can be derived:

$$\beta_{i+1} t_{i+1}(\lambda) = (\lambda - \alpha_i) t_i(\lambda) - \beta_i t_{i-1}(\lambda) + q_i(0), i = 1, 2, \ldots.$$

The weight function w is chosen so that the three-term recurrence of the orthogonal polynomials q_i is known explicitly and/or is easy to generate. An interesting class of weight functions that satisfy this requirement is considered next.

Choice of the weight functions This section assumes that $\alpha = 0$ and $\beta = 1$. Consider the Jacobi weights

$$w(\lambda) = \lambda^{\mu - 1}(1 - \lambda)^{\nu}, \text{where } \mu > 0 \text{ and } \nu \geq -\frac{1}{2}. \qquad (12.15)$$

For these weight functions, the recurrence relations are known explicitly for the polynomials that are orthogonal with respect to $w(\lambda)$, $\lambda w(\lambda)$, or $\lambda^2 w(\lambda)$. This allows the use of any of the three methods described in the previous section for computing $s_{k-1}(\lambda)$. Moreover, it has been shown [129] that the preconditioned matrix $A s_k(A)$ is Symmetric Positive Definite when A is Symmetric Positive Definite, provided that $\mu - 1 \geq \nu \geq -\frac{1}{2}$.

The following explicit formula for $R_k(\lambda)$ can be derived easily from the explicit expression of the Jacobi polynomials and the fact that $\{R_k\}$ is orthogonal with respect to the weight $\lambda w(\lambda)$:

$$R_k(\lambda) = \sum_{j=0}^{k} \kappa_j^{(k)} (1 - \lambda)^{k-j} (-\lambda)^j \qquad (12.16)$$

$$\kappa_j^{(k)} = \binom{k}{j} \prod_{i=0}^{j-1} \frac{k - i + \nu}{i + 1 + \mu}.$$

Using (12.13), the polynomial $s_{k-1}(\lambda) = (1 - R_k(\lambda))/\lambda$ can be derived easily "by hand" for small degrees; see Exercise 4.

Example 12.1 As an illustration, we list the least-squares polynomials s_k for $k = 1, \ldots,$ 8, obtained for the Jacobi weights with $\mu = \frac{1}{2}$ and $\nu = -\frac{1}{2}$. The polynomials listed are for the interval $[0, 4]$ as this leads to integer coefficients. For a general interval $[0, \beta]$, the best polynomial of degree k is $s_k(4\lambda/\beta)$. Also, each polynomial s_k is rescaled by $(3+2k)/4$ to simplify the expressions. However, this scaling factor is unimportant if these polynomials are used for preconditioning.

	1	λ	λ^2	λ^3	λ^4	λ^5	λ^6	λ^7	λ^8
s_1	5	− 1							
s_2	14	−7	1						
s_3	30	− 27	9	− 1					
s_4	55	− 77	44	− 11	1				
s_5	91	− 182	156	− 65	13	− 1			
s_6	140	− 378	450	− 275	90	− 15	1		
s_7	204	− 714	1122	− 935	442	− 119	17	− 1	
s_8	285	− 1254	2508	− 2717	1729	− 665	152	− 19	1

We selected $\mu = \frac{1}{2}$ and $\nu = -\frac{1}{2}$ only because these choices lead to a very simple recurrence for the polynomials q_i, which are the Chebyshev polynomials of the first kind.

Theoretical considerations An interesting theoretical question is whether the least-squares residual polynomial becomes small in some sense as its degree increases. Consider first the case $0 < \alpha < \beta$. Since the residual polynomial R_k minimizes the norm $\|R\|_w$ associated with the weight w, over all polynomials R of degree $\leq k$ such that $R(0) = 1$, the polynomial $(1 - (\lambda/\theta))^k$ with $\theta = (\alpha + \beta)/2$ satisfies

$$\|R_k\|_w \leq \left\| \left(1 - \frac{\lambda}{c}\right)^k \right\|_w \leq \left\| \left[\frac{b-a}{b+a}\right]^k \right\|_w = \kappa \left[\frac{\beta - \alpha}{\beta + \alpha}\right]^k$$

where κ is the w-norm of the function unity on the interval $[\alpha, \beta]$. The norm of R_k will tend to zero geometrically as k tends to infinity, provided $\alpha > 0$.

Consider now the case $\alpha = 0$, $\beta = 1$ and the Jacobi weight (12.15). For this choice of the weight function, the least-squares residual polynomial is known to be $p_k(\lambda)/p_k(0)$ where p_k is the k^{th} degree Jacobi polynomial associated with the weight function $w'(\lambda) = \lambda^\mu(1-\lambda)^\nu$. It can be shown that the 2-norm of such a residual polynomial with respect to this weight is given by

$$\|p_k/p_k(0)\|^2_{w'} = \frac{\Gamma^2(\mu+1)\Gamma(k+\nu+1)}{(2k+\mu+\nu+1)(\Gamma(k+\mu+\nu+1)} \frac{\Gamma(k+1)}{\Gamma(k+\mu+1)}$$

in which Γ is the Gamma function. For the case $\mu = \frac{1}{2}$ and $\nu = -\frac{1}{2}$, this becomes

$$\|p_k/p_k(0)\|^2_{w'} = \frac{[\Gamma(\frac{3}{2})]^2}{(2k+1)(k+\frac{1}{2})} = \frac{\pi}{2(2k+1)^2}.$$

Therefore, the w'-norm of the least-squares residual polynomial converges to zero like $1/k$ as the degree k increases (a much slower rate than when $\alpha > 0$). However, note that the condition $p(0) = 1$ implies that the polynomial must be large in some interval around the origin.

12.3.4 THE NONSYMMETRIC CASE

Given a set of approximate eigenvalues of a nonsymmetric matrix A, a simple region E can be constructed in the complex plane, e.g., a disk, an ellipse, or a polygon, which encloses the spectrum of the matrix A. There are several choices for E. The first idea uses an ellipse E that encloses an approximate convex hull of the spectrum. Consider an ellipse centered at θ, and with focal distance δ. Then as seen in Chapter 6, the shifted and scaled Chebyshev polynomials defined by

$$T_k(\lambda) \;=\; \frac{C_k\left(\frac{\theta-\lambda}{\delta}\right)}{C_k\left(\frac{\theta}{\delta}\right)}$$

are nearly optimal. The use of these polynomials leads again to an attractive three-term recurrence and to an algorithm similar to Algorithm 12.1. In fact, the recurrence is identical, except that the scalars involved can now be complex to accommodate cases where the ellipse has foci not necessarily located on the real axis. However, when A is real, then the symmetry of the foci with respect to the real axis can be exploited. The algorithm can still be written in real arithmetic.

An alternative to Chebyshev polynomials over ellipses employs a polygon H that contains $\sigma(A)$. Polygonal regions may better represent the shape of an arbitrary spectrum. The best polynomial for the infinity norm is not known explicitly but it may be computed by an algorithm known in approximation theory as the Remez algorithm. It may be simpler to use an L_2-norm instead of the infinity norm, i.e., to solve (12.11) where w is some weight function defined on the boundary of the polygon H.

Now here is a sketch of an algorithm based on this approach. We use an L_2-norm associated with Chebyshev weights on the edges of the polygon. If the contour of H consists of k edges each with center θ_i and half-length δ_i, then the weight on each edge is defined by

$$w_i(\lambda) = \frac{2}{\pi}\, |\delta_i - (\lambda-\theta_i)^2|^{-1/2}, \;\; i = 1, \ldots, k. \tag{12.17}$$

Using the power basis to express the best polynomial is not a safe practice. It is preferable to use the Chebyshev polynomials associated with the ellipse of smallest area containing H. With the above weights or any other Jacobi weights on the edges, there is a finite procedure *which does not require numerical integration* to compute the best polynomial. To do this, each of the polynomials of the basis (namely, the Chebyshev polynomials associated with the ellipse of smallest area containing H) must be expressed as a linear combination of the Chebyshev polynomials associated with the different intervals $[\theta_i - \delta_i, \theta_i + \delta_i]$. This redundancy allows exact expressions for the integrals involved in computing the least-squares solution to (12.11).

Next, the main lines of a preconditioned GMRES algorithm are described based on least-squares polynomials. Eigenvalue estimates are obtained from a GMRES step at the beginning of the outer loop. This GMRES adaptive corrects the current solution and the eigenvalue estimates are used to update the current polygon H. Correcting the solution at this stage is particularly important since it often results in a few orders of magnitude improvement. This is because the polygon H may be inaccurate and the residual vector is dominated by components in one or two eigenvectors. The GMRES step will immediately annihilate those dominating components. In addition, the eigenvalues associated with these components will now be accurately represented by eigenvalues of the Hessenberg matrix.

ALGORITHM 12.2: Polynomial Preconditioned GMRES

Start or Restart:
    Compute current residual vector $r := b - Ax$.
Adaptive GMRES step:
    Run m_1 *steps of GMRES for solving* $Ad = r$.
    Update x *by* $x := x + d$.
    Get eigenvalue estimates from the eigenvalues of the
        Hessenberg matrix.
Compute new polynomial:
    Refine H *from previous hull* H *and new eigenvalue estimates.*
    Get new best polynomial s_k.
Polynomial Iteration:
    Compute the current residual vector $r = b - Ax$.
    Run m_2 *steps of GMRES applied to* $s_k(A)Ad = s_k(A)r$.
    Update x *by* $x := x + d$.
    Test for convergence.
    If solution converged then Stop; else GoTo 1.

Example 12.2 Table 12.1 shows the results of applying GMRES(20) with polynomial preconditioning to the first four test problems described in Section 3.7.

Matrix	Iters	Kflops	Residual	Error
F2DA	56	2774	0.22E-05	0.51E-06
F3D	22	7203	0.18E-05	0.22E-05
ORS	78	4454	0.16E-05	0.32E-08
F2DB	100	4432	0.47E-05	0.19E-05

Table 12.1 *A test run of ILU(0)-GMRES accelerated with polynomial preconditioning.*

See Example 6.1 for the meaning of the column headers in the table. In fact, the system is preconditioned by ILU(0) before polynomial preconditioning is applied to it. Degree 10 polynomials (maximum) are used. The tolerance for stopping is 10^{-7}. Recall that *Iters* is the number of matrix-by-vector products rather than the number of GMRES iterations. Notice that, for most cases, the method does not compare well with the simpler ILU(0) example seen in Chapter 10. The notable exception is example F2DB for which the method converges fairly fast in contrast with the simple ILU(0)-GMRES; see Example 10.2. An attempt to use the method for the fifth matrix in the test set, namely, the FIDAP matrix FID, failed because the matrix has eigenvalues on both sides of the imaginary axis and the code tested does not handle this situation.

It is interesting to follow the progress of the algorithm in the above examples. For the first example, the coordinates of the vertices of the upper part of the first polygon H are

$\Re e(c_i)$	$\Im m(c_i)$
0.06492	0.00000
0.17641	0.02035
0.29340	0.03545
0.62858	0.04977
1.18052	0.00000

This hull is computed from the 20 eigenvalues of the 20×20 Hessenberg matrix resulting from the first run of GMRES(20). In the ensuing GMRES loop, the outer iteration converges in three steps, each using a polynomial of degree 10, i.e., there is no further adaptation required. For the second problem, the method converges in the 20 first steps of GMRES, so polynomial acceleration was never invoked. For the third example, the initial convex hull found is the interval $[0.06319, 1.67243]$ of the real line. The polynomial preconditioned GMRES then convergences in five iterations. Finally, the initial convex hull found for the last example is

$\Re e(c_i)$	$\Im m(c_i)$
0.17131	0.00000
0.39337	0.10758
1.43826	0.00000

and the outer loop converges again without another adaptation step, this time in seven steps.

12.4 MULTICOLORING

The general idea of multicoloring, or graph coloring, has been used for a long time by numerical analysts. It was exploited, in particular, in the context of relaxation techniques both for understanding their theory and for deriving efficient algorithms. More recently, these techniques were found to be useful in improving parallelism in iterative solution techniques. This discussion begins with the 2-color case, called *red-black* ordering.

12.4.1 RED-BLACK ORDERING

The problem addressed by multicoloring is to determine a coloring of the nodes of the adjacency graph of a matrix such that any two adjacent nodes have different colors. For a 2-dimensional finite difference grid (5-point operator), this can be achieved with two colors, typically referred to as “red” and “black.” This red-black coloring is illustrated in Figure 12.2 for a 6×4 mesh where the black nodes are represented by filled circles.

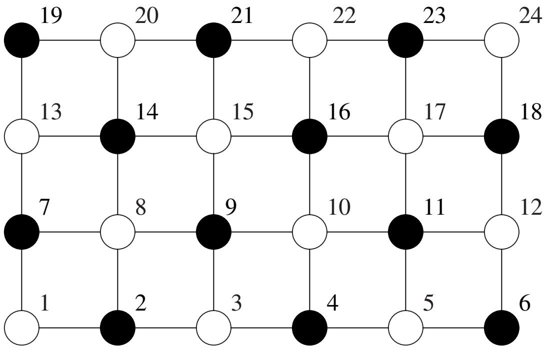


Figure 12.2 *Red-black coloring of a* 6×4 *grid. Natural labeling of the nodes.*

Assume that the unknowns are labeled by listing the red unknowns first together, followed by the black ones. The new labeling of the unknowns is shown in Figure 12.3.

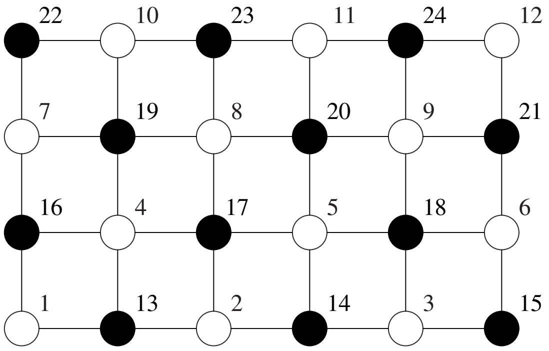


Figure 12.3 *Red-black coloring of a* 6×4 *grid. Red-black labeling of the nodes.*

Since the red nodes are not coupled with other red nodes and, similarly, the black nodes are not coupled with other black nodes, the system that results from this reordering will have the structure

$$\begin{pmatrix} D_1 & F \\ E & D_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}, \tag{12.18}$$

in which D_1 and D_2 are diagonal matrices. The reordered matrix associated with this new labeling is shown in Figure 12.4.

Two issues will be explored regarding red-black ordering. The first is how to exploit this structure for solving linear systems. The second is how to generalize this approach for systems whose graphs are not necessarily 2-colorable.

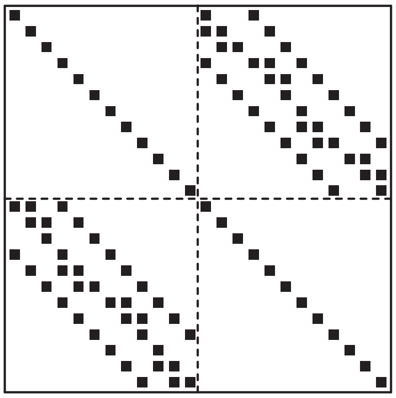


Figure 12.4 *Matrix associated with the red-black reordering of Figure 12.3.*

12.4.2 SOLUTION OF RED-BLACK SYSTEMS

The easiest way to exploit the red-black ordering is to use the standard SSOR or ILU(0) preconditioners for solving the block system (12.18) which is derived from the original system. The resulting preconditioning operations are highly parallel. For example, the linear system that arises from the forward solve in SSOR will have the form

$$\begin{pmatrix} D_1 & O \\ E & D_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}.$$

This system can be solved by performing the following sequence of operations:

1. Solve $D_1 x_1 = b_1$.
2. Compute $\hat{b}_2 := b_2 - E x_1$.
3. Solve $D_2 x_2 = \hat{b}_2$.

This consists of two diagonal scalings (operations 1 and 3) and a sparse matrix-by-vector product. Therefore, the degree of parallelism, is at least $n/2$ if an atomic task is considered to be any arithmetic operation. The situation is identical with the ILU(0) preconditioning. However, since the matrix has been reordered before ILU(0) is applied to it, the resulting LU factors are not related in any simple way to those associated with the original matrix. In fact, a simple look at the structure of the ILU factors reveals that many more elements are dropped with the red-black ordering than with the natural ordering. The result is that the number of iterations to achieve convergence can be much higher with red-black ordering than with the natural ordering.

A second method that has been used in connection with the red-black ordering solves the reduced system which involves only the black unknowns. Eliminating the red unknowns from (12.18) results in the reduced system:

$$(D_2 - E D_1^{-1} F) x_2 = b_2 - E D_1^{-1} b_1.$$

Note that this new system is again a sparse linear system with about half as many unknowns. In addition, it has been observed that for "easy problems," the reduced system can often be solved efficiently with only diagonal preconditioning. The computation of the reduced system is a highly parallel and inexpensive process. Note that it is not necessary to form the reduced system. This strategy is more often employed when D_1 is not diagonal, such as in domain decomposition methods, but it can also have some uses in other situations. For example, applying the matrix to a given vector x can be performed using nearest-neighbor communication, and this can be more efficient than the standard approach of multiplying the vector by the Schur complement matrix $D_2 - ED_1^{-1}F$. In addition, this can save storage, which may be more critical in some cases.

12.4.3 MULTICOLORING FOR GENERAL SPARSE MATRICES

Chapter 3 discussed a general greedy approach for multicoloring a graph. Given a general sparse matrix A, this inexpensive technique allows us to reorder it into a block form where the diagonal blocks are diagonal matrices. The number of blocks is the number of colors. For example, for six colors, a matrix would result with the structure shown in Figure 12.5 where the D_i's are diagonal and E, F are general sparse. This structure is obviously a generalization of the red-black ordering.

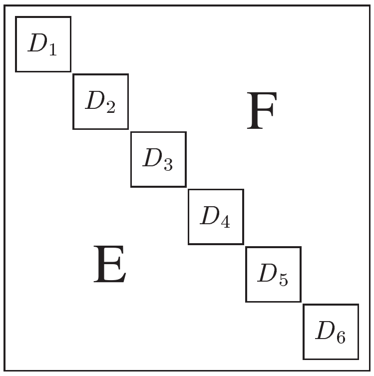


Figure 12.5 *A six-color ordering of a general sparse matrix.*

Just as for the red-black ordering, ILU(0), SOR, or SSOR preconditioning can be used on this reordered system. The parallelism of SOR/SSOR is now of order n/p where p is the number of colors. A loss in efficiency may occur since the number of iterations is likely to increase.

A Gauss-Seidel sweep will essentially consist of p scalings and $p-1$ matrix-by-vector products, where p is the number of colors. Specifically, assume that the matrix is stored in the well known Ellpack-Itpack format and that the block structure of the permuted matrix is defined by a pointer array $iptr$. The index $iptr(j)$ is the index of the first row in the j-th block. Thus, the pair $A(n1 : n2, *), JA(n1 : n2, *)$ represents the sparse matrix consisting of the rows $n1$ to $n2$ in the Ellpack-Itpack format. The main diagonal of A is assumed to be stored separately in inverted form in a one-dimensional array $diag$. One single step of the multicolor SOR iteration will then take the following form.

ALGORITHM 12.3: Multicolor SOR Sweep in the Ellpack Format

```
 1. Do col = 1, ncol
 2.    n1 = iptr(col)
 3.    n2 = iptr(col+1) – 1
 4.    y(n1:n2) = rhs(n1:n2)
 5.    Do j = 1, ndiag
 6.       Do i = n1, n2
 7.          y(i) = y(i) – a(i,j)*y(ja(i,j))
 8.       EndDo
 9.    EndDo
10.    y(n1:n2) = diag(n1:n2) * y(n1:n2)
11. EndDo
```

In the above algorithm, $ncol$ is the number of colors. The integers $n1$ and $n2$ set in lines 2 and 3 represent the beginning and the end of block col. In line 10, $y(n1 : n2)$ is multiplied by the diagonal D^{-1} which is kept in inverted form in the array $diag$. The outer loop, i.e., the loop starting in line 1, is sequential. The loop starting in line 6 is vectorizable/parallelizable. There is additional parallelism which can be extracted in the combination of the two loops starting in lines 5 and 6.

12.5 MULTI-ELIMINATION ILU

The discussion in this section begins with the Gaussian elimination algorithm for a general sparse linear system. Parallelism in sparse Gaussian elimination can be obtained by finding unknowns that are independent at a given stage of the elimination, i.e., unknowns that do not depend on each other according to the binary relation defined by the graph of the matrix. A set of unknowns of a linear system which are independent is called an independent set. Thus, independent set orderings can be viewed as permutations to put the original matrix in the form

$$\begin{pmatrix} D & E \\ F & C \end{pmatrix} \tag{12.19}$$

in which D is diagonal, but C can be arbitrary. This amounts to a less restrictive form of multicoloring, in which a set of vertices in the adjacency graph is found so that no equation in the set involves unknowns from the same set. A few algorithms for finding independent set orderings of a general sparse graph were discussed in Chapter 3.

The rows associated with an independent set can be used as pivots simultaneously. When such rows are eliminated, a smaller linear system results, which is again sparse. Then we can find an independent set for this reduced system and repeat the process of reduction. The resulting second reduced system is called the second-level reduced system. The process can be repeated recursively a few times. As the level of the reduction increases, the reduced systems gradually lose their sparsity. A direct solution method would continue the reduction until the reduced system is small enough or dense enough to switch to a dense Gaussian elimination to solve it. This process is illustrated in Figure 12.6. There exists a number of sparse direct solution techniques based on this approach.

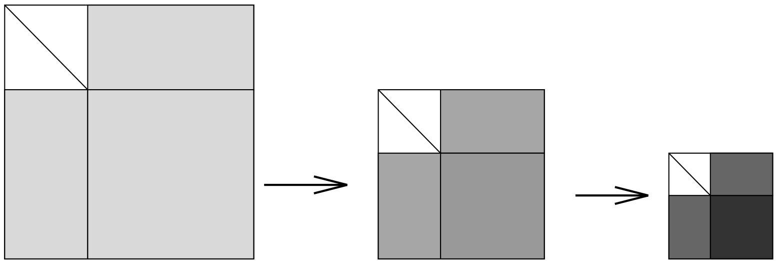


Figure 12.6 *Illustration of two levels of multi-elimination for sparse linear systems.*

After a brief review of the direct solution method based on independent set orderings, we will explain how to exploit this approach for deriving incomplete LU factorizations by incorporating drop tolerance strategies.

12.5.1 MULTI-ELIMINATION

We start by a discussion of an *exact* reduction step. Let A_j be the matrix obtained at the j-th step of the reduction, $j = 0, \ldots, nlev$ with $A_0 = A$. Assume that an independent set ordering is applied to A_j and that the matrix is permuted accordingly as follows:

$$P_j A_j P_j^T = \begin{pmatrix} D_j & F_j \\ E_j & C_j \end{pmatrix} \tag{12.20}$$

where D_j is a diagonal matrix. Now eliminate the unknowns of the independent set to get the next reduced matrix,

$$A_{j+1} = C_j - E_j D_j^{-1} F_j. \tag{12.21}$$

This results, implicitly, in a block LU factorization

$$P_j A_j P_j^T = \begin{pmatrix} D_j & F_j \\ E_j & C_j \end{pmatrix} = \begin{pmatrix} I & O \\ E_j D_j^{-1} & I \end{pmatrix} \times \begin{pmatrix} D_j & F_j \\ O & A_{j+1} \end{pmatrix}$$

with A_{j+1} defined above. Thus, in order to solve a system with the matrix A_j, both a forward and a backward substitution need to be performed with the block matrices on the right-hand side of the above system. The backward solution involves solving a system with the matrix A_{j+1}.

This block factorization approach can be used recursively until a system results that is small enough to be solved with a standard method. The transformations used in the elimination process, i.e., the matrices $E_j D_j^{-1}$ and the matrices F_j must be saved. The permutation matrices P_j can also be saved. Alternatively, the matrices involved in the factorization at each new reordering step can be permuted explicitly.

12.5.2 ILUM

The successive reduction steps described above will give rise to matrices that become more and more dense due to the fill-ins introduced by the elimination process. In iterative methods, a common cure for this is to neglect some of the fill-ins introduced by using a simple dropping strategy as the reduced systems are formed. For example, any fill-in element introduced is dropped, whenever its size is less than a given tolerance times the 2-norm of the original row. Thus, an "approximate" version of the successive reduction steps can be used to provide an approximate solution $M^{-1}v$ to $A^{-1}v$ for any given v. This can be used to precondition the original linear system. Conceptually, the modification leading to an "incomplete" factorization replaces (12.21) by

$$A_{j+1} = (C_j - E_j D_j^{-1} F_j) - R_j \tag{12.22}$$

in which R_j is the matrix of the elements that are dropped in this reduction step. Globally, the algorithm can be viewed as a form of incomplete block LU with permutations.

Thus, there is a succession of block ILU factorizations of the form

$$\begin{aligned} P_j A_j P_j^T &= \begin{pmatrix} D_j & F_j \\ E_j & C_j \end{pmatrix} \\ &= \begin{pmatrix} I & O \\ E_j D_j^{-1} & I \end{pmatrix} \times \begin{pmatrix} D_j & F_j \\ O & A_{j+1} \end{pmatrix} + \begin{pmatrix} O & O \\ O & R_j \end{pmatrix} \end{aligned}$$

with A_{j+1} defined by (12.22). An independent set ordering for the new matrix A_{j+1} will then be found and this matrix is reduced again in the same manner. It is not necessary to save the successive A_j matrices, but only the last one that is generated. We need also to save the sequence of sparse matrices

$$B_{j+1} = \begin{pmatrix} D_j & F_j \\ E_j D_j^{-1} & O \end{pmatrix} \tag{12.23}$$

which contain the transformation needed at level j of the reduction. The successive permutation matrices P_j can be discarded if they are applied to the previous B_i matrices as soon as these permutation matrices are known. Then only the global permutation is needed, which is the product of all these successive permutations.

An illustration of the matrices obtained after three reduction steps is shown in Figure 12.7. The original matrix is a 5-point matrix associated with a 15×15 grid and is therefore of size $N = 225$. Here, the successive matrices B_i (with permutations applied) are shown together with the last A_j matrix which occupies the location of the O block in (12.23).

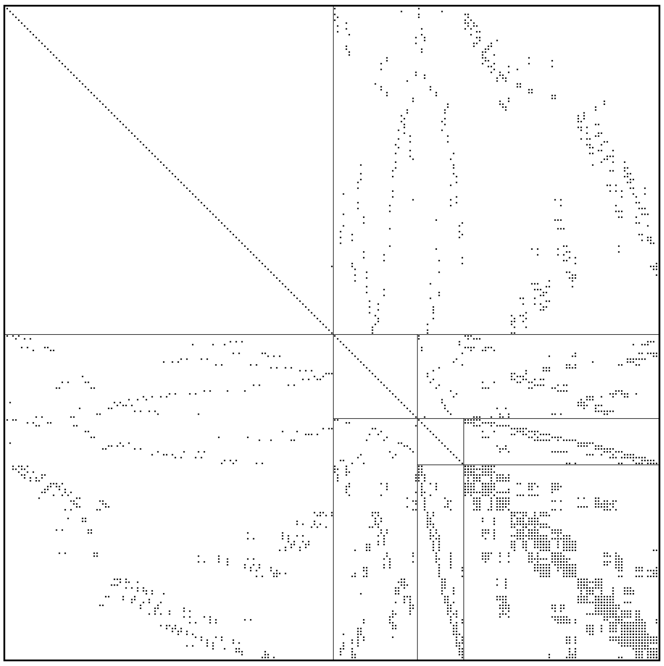


Figure 12.7 *Illustration of the processed matrices obtained from three steps of independent set ordering and reductions.*

We refer to this incomplete factorization as ILUM (ILU with Multi-Elimination). The preprocessing phase consists of a succession of $nlev$ applications of the following three steps: (1) finding the independent set ordering, (2) permuting the matrix, and (3) reducing it.

ALGORITHM 12.4: ILUM: Preprocessing Phase

1. Set $A_0 = A$.
2. For $j = 0, 1, \ldots, nlev - 1$ Do:
3. Find an independent set ordering permutation P_j for A_j;
4. Apply P_j to A_j to permute it into the form (12.20);
5. Apply P_j to $B_1, \ldots, B_j$;
6. Apply P_j to $P_0, \ldots, P_{j-1}$;
7. Compute the matrices A_{j+1} and B_{j+1} defined by (12.22) and (12.23).
8. EndDo

In the backward and forward solution phases, the last reduced system must be solved but not necessarily with high accuracy. For example, we can solve it according to the level of tolerance allowed in the dropping strategy during the preprocessing phase. Observe that if the linear system is solved inaccurately, only an accelerator that allows variations in the preconditioning should be used. Such algorithms have been discussed in Chapter 9. Alternatively, we can use a fixed number of multicolor SOR or SSOR steps or a fixed polynomial iteration. The implementation of the ILUM preconditioner corresponding to this strategy is rather complicated and involves several parameters.

In order to describe the forward and backward solution, we introduce some notation. We start by applying the "global permutation," i.e., the product

$$P_{nlev-1}, P_{nlev-2} \dots, P_0$$

to the right-hand side. We overwrite the result on the current solution vector, an N-vector called x_0. Now partition this vector into

$$x_0 = \begin{pmatrix} y_0 \\ x_1 \end{pmatrix}$$

according to the partitioning (12.20). The forward step consists of transforming the second component of the right-hand side as

$$x_1 := x_1 - E_0 D_0^{-1} y_0.$$

Now x_1 is partitioned in the same manner as x_0 and the forward elimination is continued the same way. Thus, at each step, each x_j is partitioned as

$$x_j = \begin{pmatrix} y_j \\ x_{j+1} \end{pmatrix}.$$

A forward elimination step defines the new x_{j+1} using the old x_{j+1} and y_j for $j = 0, \dots, nlev - 1$ while a backward step defines y_j using the old y_j and x_{j+1}, for $j = nlev - 1, \dots, 0$. Algorithm 12.5 describes the general structure of the forward and backward solution sweeps. Because the global permutation was applied at the beginning, the successive permutations need not be applied. However, the final result obtained must be permuted back into the original ordering.

ALGORITHM 12.5: ILUM: Forward and Backward Solutions

1. *Apply global permutation to right-hand-side* b *and copy into* x_0.
2. *For* $j = 0, 1, \dots, nlev - 1$ *Do: [Forward sweep]*
3. $\quad x_{j+1} := x_{j+1} - E_j D_j^{-1} y_j$
4. *EndDo*
5. *Solve with a relative tolerance* ϵ:
6. $\quad A_{nlev} x_{nlev} := x_{nlev}$.
7. *For* $j = nlev - 1, \dots, 1, 0$ *Do: [Backward sweep]*
8. $\quad y_j := D_j^{-1}(y_j - F_j x_{j+1})$.
9. *EndDo*
10. *Permute the resulting solution vector back to the original*
11. *ordering to obtain the solution* x.

Computer implementations of ILUM can be rather tedious. The implementation issues are similar to those of parallel direct-solution methods for sparse linear systems.

12.6 DISTRIBUTED ILU AND SSOR

This section describes parallel variants of the block Successive Over-Relaxation (BSOR) and ILU(0) preconditioners which are suitable for distributed memory environments. Chapter 11 briefly discussed *distributed sparse matrices*.. A distributed matrix is a matrix whose entries are located in the memories of different processors in a multiprocessor system. These types of data structures are very convenient for distributed memory computers and it is useful to discuss implementations of preconditioners that are specifically developed for them. Refer to Section 11.5.6 for the terminology used here. In particular, the term *subdomain* is used in the very general sense of subgraph. For both ILU and SOR, multicoloring or level scheduling can be used at the macro level, to extract parallelism. Here, macro level means the level of parallelism corresponding to the processors, or blocks, or subdomains.

12.6.1 DISTRIBUTED SPARSE MATRICES

In the ILU(0) factorization, the LU factors have the same nonzero patterns as the original matrix A, so that the references of the entries belonging to the external subdomains in the ILU(0) factorization are identical with those of the matrix-by-vector product operation with the matrix A. This is not the case for the more accurate ILU(p) factorization, with $p > 0$. If an attempt is made to implement a wavefront ILU preconditioner on a distributed memory computer, a difficulty arises because the natural ordering for the original sparse problem may put an unnecessary limit on the amount of parallelism available. Instead, a two-level ordering is used. First, define a "global" ordering which is a wavefront ordering for the subdomains. This is based on the graph which describes the coupling between the subdomains: Two subdomains are coupled if and only if they contain at least a pair of coupled unknowns, one from each subdomain. Then, within each subdomain, define a local ordering.

To describe the possible parallel implementations of these ILU(0) preconditioners, it is sufficient to consider a local view of the distributed sparse matrix, illustrated in Figure 12.8. The problem is partitioned into p subdomains or subgraphs using some graph partitioning technique. This results in a mapping of the matrix into processors where it is assumed that the i-th equation (row) and the i-th unknown are mapped to the same processor. We distinguish between *interior* points and *interface* points. The interior points are those nodes that are not coupled with nodes belonging to other processors. Interface nodes are those local nodes that are coupled with at least one node which belongs to another processor. Thus, processor number 10 in the figure holds a certain number of rows that are local rows. Consider the rows associated with the interior nodes. The unknowns associated with these nodes are not coupled with variables from other processors. As a result, the rows associated with these nodes can be eliminated independently in the ILU(0) process. The rows associated with the nodes on the interface of the subdomain will require more attention. Recall that an ILU(0) factorization is determined entirely by the order in which the rows are processed. The interior nodes can be eliminated first. Once this is done, the interface rows can be eliminated *in a certain order*. There are two natural choices for this order. The first would be to impose a global order based on the labels of the processors. Thus, in the illustration, the interface rows belonging to Processors 2, 4, and 6 are processed before those in Processor 10. The interface rows in Processor 10 must in turn be processed before those of Processors 13 and 14. The local order, i.e., the order in which we process the interface rows in the same processor (e.g. Processor 10), may not be as important. This global order based on PE-number defines a natural priority graph and parallelism can be exploited easily in a data-driven implementation.

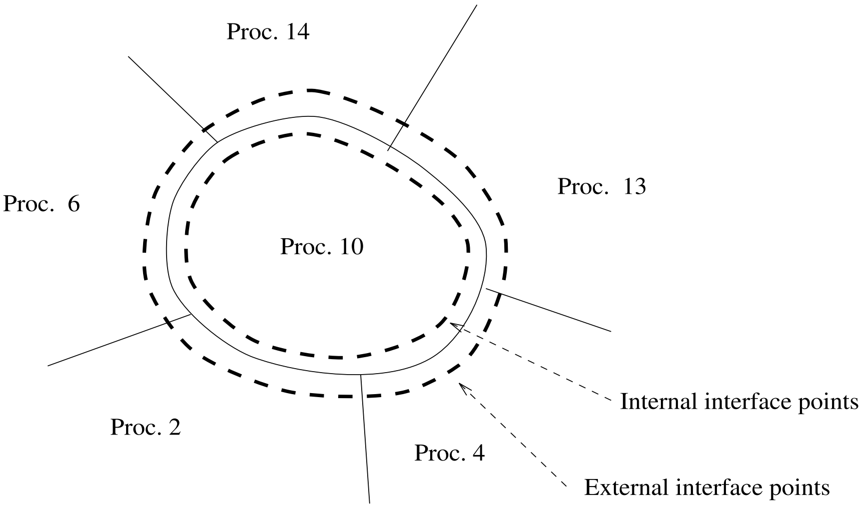


Figure 12.8 *A local view of the distributed ILU(0).*

It is somewhat unnatural to base the ordering just on the processor labeling. Observe that a proper order can also be defined for performing the elimination by *replacing the PE-numbers with any labels, provided that any two neighboring processors have a different label*. The most natural way to do this is by performing a multicoloring of the subdomains, and using the colors in exactly the same way as before to define an order of the tasks. The algorithms will be written in this general form, i.e., with a label associated with each processor. Thus, the simplest valid labels are the PE numbers, which lead to the PE-label-based order. In the following, we define Lab_j as the label of Processor number j.

ALGORITHM 12.6: Distributed ILU(0) factorization

1. *In each processor $P_i,\ i = 1, \ldots, p$ Do:*
2. *Perform the ILU(0) factorization for interior local rows.*
3. *Receive the factored rows from the adjacent processors j with*
4. $Lab_j < Lab_i$.
5. *Perform the ILU(0) factorization for the interface rows with*
6. *pivots received from the external processors in step 3.*
7. *Perform the ILU(0) factorization for the boundary nodes, with*
8. *pivots from the interior rows completed in step 2.*
9. *Send the completed interface rows to adjacent processors j with*

10. $\qquad Lab_j > Lab_i$.
11. EndDo

Step 2 of the above algorithm can be performed in parallel because it does not depend on data from other subdomains. Once this distributed ILU(0) factorization is completed, the preconditioned Krylov subspace algorithm will require a forward and backward sweep at each step. The distributed forward/backward solution based on this factorization can be implemented as follows.

ALGORITHM 12.7: Distributed Forward and Backward Sweep

1. *In each processor* $P_i,\ i = 1, \ldots, p$ *Do:*
2.   *Forward solve:*
3.   *Perform the forward solve for the interior nodes.*
4.   *Receive the updated values from the adjacent processors* j
5.   *with* $Lab_j < Lab_i$.
6.   *Perform the forward solve for the interface nodes.*
7.   *Send the updated values of boundary nodes to the adjacent*
8.   *processors* j *with* $Lab_j > Lab_i$.
9.   *Backward solve:*
10.   *Receive the updated values from the adjacent processors* j
11.   *with* $Lab_j > Lab_i$.
12.   *Perform the backward solve for the boundary nodes.*
13.   *Send the updated values of boundary nodes to the adjacent*
14.   *processors,* j *with* $Lab_j < Lab_i$.
15.   *Perform the backward solve for the interior nodes.*
16. *EndDo*

As in the ILU(0) factorization, the interior nodes do not depend on the nodes from the external processors and can be computed in parallel in lines 3 and 15. In the forward solve, the solution of the interior nodes is followed by an exchange of data and the solution on the interface. The backward solve works in reverse in that the boundary nodes are first computed, then they are sent to adjacent processors. Finally, interior nodes are updated.

12.7 OTHER TECHNIQUES

This section gives a brief account of other parallel preconditioning techniques which are sometimes used. The next chapter also examines another important class of methods, which were briefly mentioned before, namely, the class of Domain Decomposition methods.

12.7.1 APPROXIMATE INVERSES

Another class of preconditioners that require only matrix-by-vector products, is the class of approximate inverse preconditioners. Discussed in Chapter 10, these can be used in many different ways. Besides being simple to implement, both their preprocessing phase and iteration phase allow a large degree of parallelism. Their disadvantage is similar to polynomial preconditioners, namely, the number of steps required for convergence may be large, possibly substantially larger than with the standard techniques. On the positive side, they are fairly robust techniques which can work well where standard methods may fail.

12.7.2 ELEMENT-BY-ELEMENT TECHNIQUES

A somewhat specialized set of techniques is the class of Element-By-Element (EBE) preconditioners which are geared toward finite element problems and are motivated by the desire to avoid assembling finite element matrices. Many finite element codes keep the data related to the linear system in unassembled form. The element matrices associated with each element are stored and never added together. This is convenient when using direct methods since there are techniques, known as frontal methods, that allow Gaussian elimination to be performed by using a few elements at a time.

It was seen in Chapter 2 that the global stiffness matrix A is the sum of matrices $A^{[e]}$ associated with each element, i.e.,

$$A = \sum_{e=1}^{Nel} A^{[e]}.$$

Here, the matrix $A^{[e]}$ is an $n \times n$ matrix defined as

$$A^{[e]} = P_e A_{K_e} P_e^T$$

in which A_{K_e} is the element matrix and P_e is a Boolean connectivity matrix which maps the coordinates of the small A_{K_e} matrix into those of the full matrix A. Chapter 2 showed how matrix-by-vector products can be performed in unassembled form. To perform this product in parallel, note that the only potential obstacle to performing the matrix-by-vector product in parallel, i.e., across all elements, is in the last phase, i.e., when the contributions are summed to the resulting vector y. In order to add the contributions $A^{[e]}x$ in parallel, group elements that do not have nodes in common. Referring to Equation (2.35), the contributions

$$y_e = A_{K_e}(P_e^T x)$$

can all be computed in parallel and do not depend on one another. The operations

$$y := y + P_e y_e$$

can be processed in parallel for any group of elements that do not share any vertices. This grouping can be found by performing a multicoloring of the elements. Any two elements which have a node in common receive a different color. Using this idea, good performance can be achieved on vector computers.

EBE preconditioners are based on similar principles and many different variants have been developed. They are defined by first normalizing each of the element matrices. In the sequel, assume that A is a Symmetric Positive Definite matrix. Typically, a diagonal, or block diagonal, scaling is first applied to A to obtain a scaled matrix $\tilde{A}$,

$$\tilde{A} = D^{-1/2} A D^{-1/2}. \tag{12.24}$$

This results in each matrix $A^{[e]}$ and element matrix A_{K_e} being transformed similarly:

$$\begin{aligned}\tilde{A}^{[e]} &= D^{-1/2} A^{[e]} D^{-1/2} \\ &= D^{-1/2} P_e A_{K_e} D^{-1/2} \\ &= P_e (P_e^T D^{-1/2} P_e) A^{[e]} (P_e D^{-1/2} P_e^T) \\ &\equiv P_e \tilde{A}_{K_e} P_e^T.\end{aligned}$$

The second step in defining an EBE preconditioner is to *regularize* each of these transformed matrices. Indeed, each of the matrices $A^{[e]}$ is of rank p_e at most, where p_e is the size of the element matrix A_{K_e}, i.e., the number of nodes which constitute the e-th element. In the so-called *Winget regularization*, the diagonal of each $A^{[e]}$ is forced to be the identity matrix. In other words, the regularized matrix is defined as

$$\bar{A}^{[e]} = I + \tilde{A}^{[e]} - \mathrm{diag}(\tilde{A}^{[e]}). \tag{12.25}$$

These matrices are positive definite; see Exercise 8.

The third and final step in defining an EBE preconditioner is to choose the factorization itself. In the EBE Cholesky factorization, the Cholesky (or Crout) factorization of each regularized matrix $\bar{A}^{[e]}$ is performed,

$$\bar{A}^{[e]} = L_e D_e L_e^T. \tag{12.26}$$

The preconditioner from it is defined as

$$M = \prod_{e=1}^{nel} L_e \times \prod_{e=1}^{nel} D_e \times \prod_{e=nel}^{1} L_e^T. \tag{12.27}$$

Note that to ensure symmetry, the last product is in reverse order of the first one. The factorization (12.26) consists of a factorization of the small $p_e \times p_e$ matrix $\bar{A}_{K_e}$. Performing the preconditioning operations will therefore consist of a sequence of small $p_e \times p_e$ backward or forward solves. The gather and scatter matrices P_e defined in Chapter 2 must also be applied for each element. These solves are applied to the right-hand side in sequence. In addition, the same multicoloring idea as for the matrix-by-vector product can be exploited to perform these sweeps in parallel.

One of the drawbacks of the EBE Cholesky preconditioner is that an additional set of element matrices must be stored. That is because the factorizations (12.26) must be stored for each element. In EBE/SSOR, this is avoided. Instead of factoring each $\bar{A}^{[e]}$, the usual splitting of each $\bar{A}^{[e]}$ is exploited. Assuming the Winget regularization, we have

$$\bar{A}^{[e]} = I - E_e - E_e^T \tag{12.28}$$

in which $-E_e$ is the strict-lower part of $\bar{A}^{[e]}$. By analogy with the SSOR preconditioner, the EBE-SSOR preconditioner is defined by

$$M = \prod_{e=1}^{nel} (I - \omega E_e) \times \prod_{e=1}^{nel} D_e \times \prod_{e=nel}^{1} (I - \omega E_e^T). \tag{12.29}$$

12.7.3 PARALLEL ROW PROJECTION PRECONDITIONERS

One of the attractions of row-projection methods seen in Chapter 8 is their high degree of parallelism. In Cimmino's method, the scalars δ_i as well as the new residual vector can be computed in parallel. In the Gauss-Seidel-NE (respectively Gauss-Seidel-NR), it is also possible to group the unknowns in such a way that any pair of rows (respectively columns) have disjointed nonzero patterns. Updates of components in the same group can then be performed in parallel. This approach essentially requires finding a multicolor ordering for the matrix $B = AA^T$ (respectively $B = A^T A$).

It is necessary to first identify a partition of the set $\{1, 2, \ldots, N\}$ into subsets $\mathcal{S}_1, \ldots$, $\mathcal{S}_k$ such that the rows (respectively columns) whose indices belong to the same set $\mathcal{S}_i$ are *structurally* orthogonal to each other, i.e., have no nonzero elements in the same column locations. When implementing a block SOR scheme where the blocking is identical with that defined by the partition, all of the unknowns belonging to the same set $\mathcal{S}_j$ can be updated in parallel. To be more specific, the rows are reordered by scanning those in $\mathcal{S}_1$ followed by those in $\mathcal{S}_2$, etc.. Denote by A_i the matrix consisting of the rows belonging to the i-th block. We assume that all rows of the same set are orthogonal to each other and that they have been normalized so that their 2-norm is unity. Then a block Gauss-Seidel sweep, which generalizes Algorithm 8.1, follows.

ALGORITHM 12.8: Forward Block NE-Gauss-Seidel Sweep

1. *Select an initial* x_0.
2. *For* $i = 1, 2, \ldots, k$ *Do:*
3. $\quad d_i = b_i - A_i x$
4. $\quad x := x + A_i^T d_i$
5. *EndDo*

Here, x_i and b_i are subvectors corresponding to the blocking and d_i is a vector of length the size of the block, which replaces the scalar δ_i of Algorithm 8.1. There is parallelism in each of the steps 3 and 4.

The question that arises is how to find good partitions $\mathcal{S}_i$. In simple cases, such as block-tridiagonal matrices, this can easily be done; see Exercise 7. For general sparse matrices, a multicoloring algorithm on the graph of AA^T (respectively $A^T A$) can be employed. However, these matrices are never stored explicitly. Their rows can be generated, used, and then discarded.

EXERCISES

1. Let A be a Symmetric Positive Definite matrix and consider $N = I - D^{-1}A$ where D is a block diagonal of A.

a. Show that D is a Symmetric Positive Definite matrix. Denote by $(.,.)_D$ the associated inner product.

b. Show that N is self-adjoint with respect to to $(.,.)_D$.

c. Show that N^k is self-adjoint with respect to to $(.,.)_D$ for any integer k.

d. Show that the Neumann series expansion preconditioner defined by the right-hand side of (12.3) leads to a preconditioned matrix that is self-adjoint with respect to the D-inner product.

e. Describe an implementation of the preconditioned CG algorithm using this preconditioner.

2. The development of the Chebyshev iteration algorithm seen in Section 12.3.2 can be exploited to derive yet another formulation of the conjugate algorithm from the Lanczos algorithm. Observe that the recurrence relation (12.8) is not restricted to scaled Chebyshev polynomials.

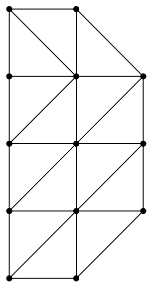
a. The scaled Lanczos polynomials, i.e., the polynomials $p_k(t)/p_k(0)$, in which $p_k(t)$ is the polynomial such that $v_{k+1} = p_k(A)v_1$ in the Lanczos algorithm, satisfy a relation of the form (12.8). What are the coefficients ρ_k and δ in this case?

b. Proceed in the same manner as in Section 12.3.2 to derive a version of the Conjugate Gradient algorithm.

3. Show that ρ_k as defined by (12.7) has a limit ρ. What is this limit? Assume that Algorithm 12.1 is to be executed with the ρ_k's all replaced by this limit ρ. Will the method converge? What is the asymptotic rate of convergence of this modified method?

4. Derive the least-squares polynomials for $\alpha = -\frac{1}{2}, \beta = \frac{1}{2}$ for the interval $[0, 1]$ for $k = 1, 2, 3, 4$. Check that these results agree with those of the table shown at the end of Section 12.3.3.

5. Consider the mesh shown below. Assume that the objective is to solve the Poisson equation with Dirichlet boundary conditions.



a. Consider the resulting matrix obtained (before boundary conditions are applied) from ordering the nodes from bottom up, and left to right (thus, the bottom left vertex is labeled 1 and the top right vertex is labeled 13). What is the bandwidth of the linear system? How many memory locations would be needed to store the matrix in Skyline format? (Assume that the matrix is nonsymmetric so both upper and lower triangular parts must be stored).

b. Is it possible to find a 2-color ordering of the mesh points? If so, show the ordering, or otherwise prove that it is not possible.

c. Find an independent set of size 5. Show the pattern of the matrix associated with this independent set ordering.

d. Find a multicolor ordering of the mesh by using the greedy multicolor algorithm. Can you find a better coloring (i.e., a coloring with fewer colors)? If so, show the coloring [use letters to represent each color].

6. A linear system $Ax = b$ where A is a 5-point matrix, is reordered using red-black ordering as

$$\begin{pmatrix} D_1 & F \\ E & D_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix}.$$

a. Write the block Gauss-Seidel iteration associated with the above partitioned system (where the blocking in block Gauss-Seidel is the same as the above blocking).

b. Express the y iterates, independently of the x iterates, i.e., find an iteration which involves only y-iterates. What type of iteration is the resulting scheme?

7. Consider a tridiagonal matrix $T = \mathrm{tridiag}\ (a_i, b_i, c_i)$. Find a grouping of the rows such that rows in each group are *structurally* orthogonal, i.e., orthogonal regardless of the values of the entry. Find a set of three groups at most. How can this be generalized to block tridiagonal matrices such as those arising from 2-D and 3-D centered difference matrices?

8. Why are the Winget regularized matrices $\bar{A}^{[e]}$ defined by (12.25) positive definite when the matrix $\tilde{A}$ is obtained from A by a *diagonal* scaling from A?

NOTES AND REFERENCES. As vector processing appeared in the middle to late 1970s, a number of efforts were made to change algorithms, or implementations of standard methods, to exploit the new architectures. One of the first ideas in this context was to perform matrix-by-vector products by diagonals [133]. Matrix-by-vector products using this format can yield excellent performance. Hence, came the idea of using polynomial preconditioning. Polynomial preconditioning was exploited independently of supercomputing, as early as 1952 in a paper by Lanczos [141], and later for eigenvalue problems by Stiefel who employed least-squares polynomials [204], and Rutishauser [171] who combined the QD algorithm with Chebyshev acceleration. Dubois et al. [75] suggested using polynomial preconditioning, specifically, the Neumann series expansion, for solving Symmetric Positive Definite linear systems on vector computers. Johnson et al. [129] later extended the idea by exploiting Chebyshev polynomials, and other orthogonal polynomials. It was observed in [129] that least-squares polynomials tend to perform better than those based on the uniform norm, in that they lead to a better overall clustering of the spectrum. Moreover, as was already observed by Rutishauser [171], in the symmetric case there is no need for accurate eigenvalue estimates: It suffices to use the simple bounds that are provided by Gershgorin's theorem. In [175] it was also observed that in some cases the least-squares polynomial approach which requires less information than the Chebyshev approach tends to perform better.

The use of least-squares polynomials over polygons was first advocated by Smolarski and Saylor [200] and later by Saad [176]. The application to the indefinite case was examined in detail in [174]. Still in the context of using polygons instead of ellipses, yet another attractive possibility proposed by Fischer and Reichel [91] avoids the problem of best approximation altogether. The polygon can be conformally transformed into a circle and the theory of Faber polynomials yields a simple way of deriving good polynomials from exploiting specific points on the circle.

Although only approaches based on the formulation (12.5) and (12.11) have been discussed, there are other lesser known possibilities based on minimizing $\|1/\lambda - s(\lambda)\|_\infty$. There has been very little work on polynomial preconditioning or Krylov subspace methods for highly non-normal matrices; see, however, the recent analysis in [207]. Another important point is that polynomial preconditioning can be combined with a subsidiary relaxation-type preconditioning such as SSOR [2, 153]. Finally, polynomial preconditionings can be useful in some special situations such as that of complex linear systems arising from the Helmholtz equation [93].

Multicoloring has been known for a long time in the numerical analysis literature and was used in particular for understanding the theory of relaxation techniques [232, 213] as well as for deriving efficient alternative formulations of some relaxation algorithms [213, 110]. More recently, it became an essential ingredient in parallelizing iterative algorithms, see for example [4, 2, 82, 155, 154, 164]. It is also commonly used in a slightly different form — coloring elements as opposed to nodes — in finite elements techniques [23, 217]. In [182] and [69], it was observed that k-step SOR preconditioning was very competitive relative to the standard ILU preconditioners. Combined with multicolor ordering, multiple-step SOR can perform quite well on supercomputers. Multicoloring is especially useful in Element-By-Element techniques when forming the residual, i.e., when multiplying an unassembled matrix by a vector [123, 88, 194]. The contributions of the elements of the same color can all be evaluated and applied simultaneously to the resulting vector. In addition to the parallelization aspects, reduced systems can sometimes be much better conditioned than the original system, see [83].

Independent set orderings have been used mainly in the context of parallel direct solution techniques for sparse matrices [66, 144, 145] and multifrontal techniques [77] can be viewed as a particular case. The gist of all these techniques is that it is possible to reorder the system in groups of equations which can be solved simultaneously. A parallel direct solution sparse solver based on performing several successive levels of independent set orderings and reduction was suggested in [144] and in a more general form in [65]. ■

CHAPTER

13

DOMAIN DECOMPOSITION METHODS

As multiprocessing technology is steadily gaining ground, new classes of numerical methods that can take better advantage of parallelism are emerging. Among these techniques, domain decomposition methods are undoubtedly the best known and perhaps the most promising for certain types of problems. These methods combine ideas from Partial Differential Equations, linear algebra, mathematical analysis, and techniques from graph theory. This chapter is devoted to "decomposition" methods, which are based on the general concepts of graph partitionings.

13.1 INTRODUCTION

Domain decomposition methods refer to a collection of techniques which revolve around the principle of divide-and-conquer. Such methods have been primarily developed for solving Partial Differential Equations over regions in two or three dimensions. However, similar principles have been exploited in other contexts of science and engineering. In fact, one of the earliest practical uses for domain decomposition approaches was in structural engineering, a discipline which is not dominated by Partial Differential Equations. Although this chapter considers these techniques from a purely linear algebra view-point, the basic concepts, as well as the terminology, are introduced from a model Partial Differential Equation.

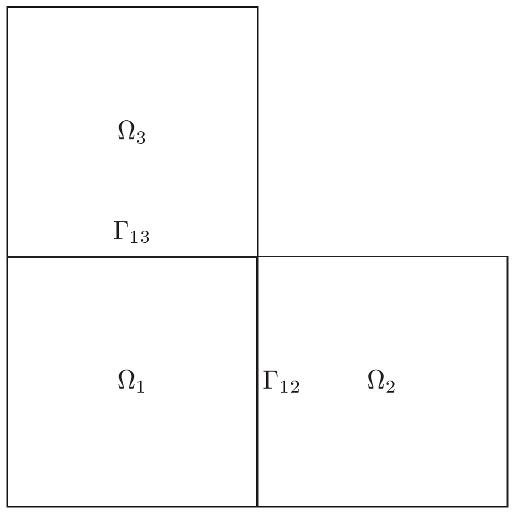


Figure 13.1 *An L-shaped domain subdivided into three subdomains.*

Consider the problem of solving the Laplace Equation on an L-shaped domain Ω partitioned as shown in Figure 13.1. Domain decomposition or substructuring methods attempt to solve the problem on the entire domain

$$\Omega = \bigcup_{i=1}^{s} \Omega_i,$$

from problem solutions on the subdomains Ω_i. There are several reasons why such techniques can be advantageous. In the case of the above picture, one obvious reason is that the subproblems are much simpler because of their rectangular geometry. For example, fast solvers can be used on each subdomain in this case. A second reason is that the physical problem can sometimes be split naturally into a small number of subregions where the modeling equations are different (e.g., Euler's equations on one region and Navier-Stokes in another). Substructuring can also be used to develop "out-of-core" solution techniques. As already mentioned, such techniques were often used in the past to analyze very large mechanical structures. The original structure is partitioned into s pieces, each of which is small enough to fit into memory. Then a form of block-Gaussian elimination is used to solve the global linear system from a sequence of solutions using s subsystems. More recent interest in domain decomposition techniques has been motivated by parallel processing.

13.1.1 NOTATION

In order to review the issues and techniques in use and to introduce some notation, assume that the following problem is to be solved:

$$\Delta u = f \text{ in } \Omega$$

$$u = u_\Gamma \text{ on } \Gamma = \partial\Omega.$$

Domain decomposition methods are all implicitly or explicitly based on different ways of handling the unknown at the interfaces. From the PDE point of view, if the value of the solution is known at the interfaces between the different regions, these values could be used in Dirichlet-type boundary conditions and we will obtain s uncoupled Poisson equations. We can then solve these equations to obtain the value of the solution at the interior points. If the whole domain is discretized by either finite elements or finite difference techniques, then this is easily translated into the resulting linear system.

Now some terminology and notation will be introduced for use throughout this chapter. Assume that the problem associated with domain shown in Figure 13.1 is discretized with centered differences. We can label the nodes by subdomain as shown in Figure 13.3. Note that the interface nodes are labeled last. As a result, the matrix associated with this problem will have the structure shown in Figure 13.4. For a general partitioning into s subdomains, the linear system associated with the problem has the following structure:

$$\begin{pmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & B_s & E_s \\ F_1 & F_2 & \cdots & F_s & C \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_s \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \\ g \end{pmatrix} \tag{13.1}$$

where each x_i represents the subvector of unknowns that are interior to subdomain Ω_i and y represents the vector of all interface unknowns. It is useful to express the above system in the simpler form,

$$A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \quad \text{with} \quad A = \begin{pmatrix} B & E \\ F & C \end{pmatrix}. \tag{13.2}$$

Thus, E represents the subdomain to interface coupling seen from the subdomains, while F represents the interface to subdomain coupling seen from the interface nodes.

13.1.2 TYPES OF PARTITIONINGS

When partitioning a problem, it is common to use graph representations. Since the subproblems obtained from a given partitioning will eventually be mapped into distinct processors, there are some restrictions regarding the type of partitioning needed. For example, in Element-By-Element finite element techniques, it may be desirable to map elements into processors instead of vertices. In this case, the restriction means no element should be split between two subdomains, i.e., all information related to a given element is mapped to the same processor. These partitionings are termed element-based. A somewhat less restrictive class of partitionings are the edge-based partitionings, which do not allow edges to be split between two subdomains. These may be useful for finite volume techniques where computations are expressed in terms of fluxes across edges in two dimensions. Finally, vertex-based partitionings work by dividing the origin vertex set into subsets of vertices and have no restrictions on the edges, i.e., they allow edges or elements to straddle between subdomains. See Figure 13.2, (a), (b), and (c).

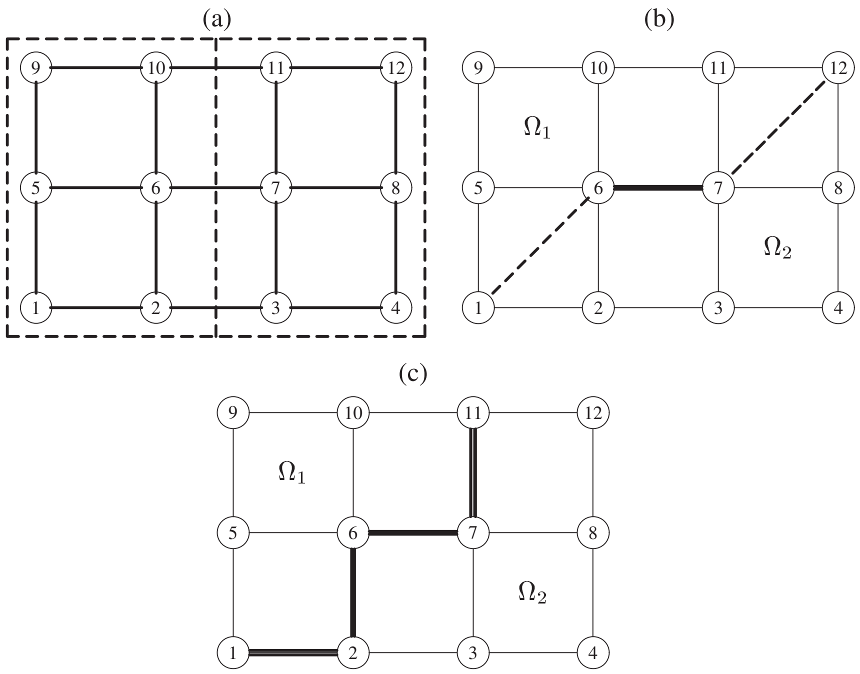


Figure 13.2 *(a) Vertex-based, (b) edge-based, and (c) element-based partitioning of a* 4×3 *mesh into two subregions.*

13.1.3 TYPES OF TECHNIQUES

The interface values can be obtained by employing a form of block-Gaussian elimination which may be too expensive for large problems. In some simple cases, using FFT's, it is possible to explicitly obtain the solution of the problem on the interfaces inexpensively.

Other methods alternate between the subdomains, solving a new problem each time, with boundary conditions updated from the most recent subdomain solutions. These methods are called *Schwarz Alternating Procedures*, after the Swiss mathematician who used the idea to prove the existence for a solution of the Dirichlet problem on irregular regions.

The subdomains may be allowed to *overlap*. This means that the Ω_i's are such that

$$\Omega = \bigcup_{i=1,s} \Omega_i, \quad \Omega_i \cap \Omega_j \neq \phi.$$

For a discretized problem, it is typical to quantify the extent of overlapping by the number of mesh-lines that are common to the two subdomains. In the particular case of Figure 13.3, the overlap is of order one.

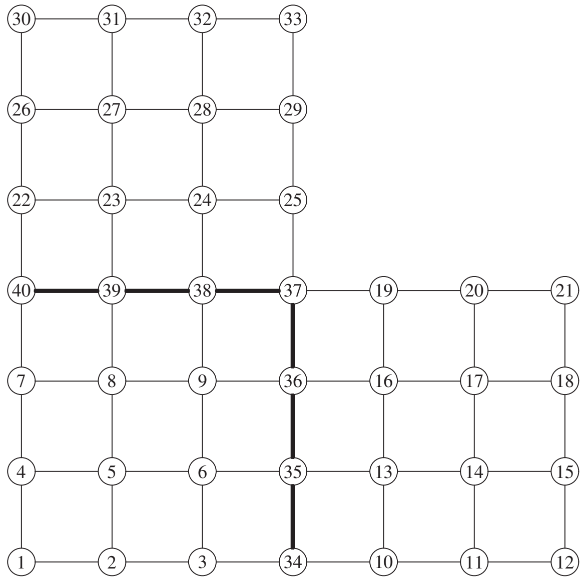


Figure 13.3 *Discretization of problem shown in Figure 13.1.*

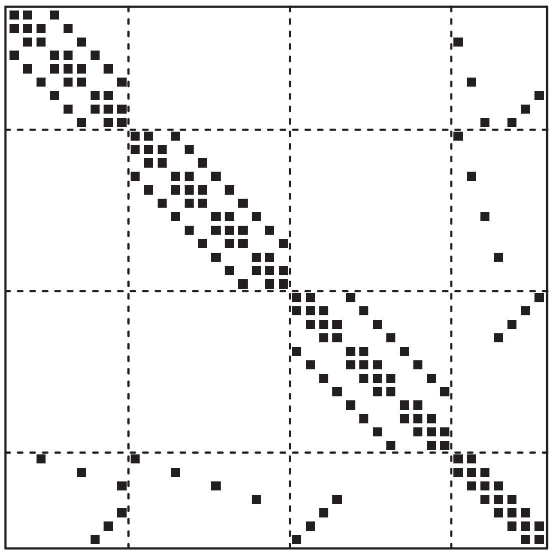


Figure 13.4 *Matrix associated with the finite difference mesh of Figure 13.3.*

The various domain decomposition techniques are distinguished by four features:

1. *Type of Partitioning.* For example, should partitioning occur along edges, or along vertices, or by elements? Is the union of the subdomains equal to the original domain or a superset of it (fictitious domain methods)?

2. *Overlap.* Should sub-domains overlap or not, and by how much?
3. *Processing of interface values.* For example, is the Schur complement approach used? Should there be successive updates to the interface values?
4. *Subdomain solution.* Should the subdomain problems be solved exactly or approximately by an iterative method?

The methods to be discussed in this chapter will be classified in four distinct groups. First, direct methods and the substructuring approach are useful for introducing some definitions and for providing practical insight. Second, among the simplest and oldest techniques are the Schwarz Alternating Procedures. Then, there are methods based on preconditioning the Schur complement system. The last category groups all the methods based on solving the linear system with the matrix A, by using a preconditioning derived from Domain Decomposition concepts.

13.2 DIRECT SOLUTION AND THE SCHUR COMPLEMENT

One of the first divide-and-conquer ideas used in structural analysis exploited the partitioning (13.1) in a direct solution framework. This approach, which is covered in this section, introduces the Schur complement and explains some of its properties.

13.2.1 BLOCK GAUSSIAN ELIMINATION

Consider the linear system written in the form (13.2), in which B is assumed to be nonsingular. From the first equation the unknown x can be expressed as

$$x = B^{-1}(f - Ey). \tag{13.3}$$

Upon substituting this into the second equation, the following *reduced system* is obtained:

$$(C - FB^{-1}E)y = g - FB^{-1}f. \tag{13.4}$$

The matrix

$$S = C - FB^{-1}E \tag{13.5}$$

is called the *Schur complement* matrix associated with the y variable. If this matrix can be formed and the linear system (13.4) can be solved, all the interface variables y will become available. Once these variables are known, the remaining unknowns can be computed, via (13.3). Because of the particular structure of B, observe that any linear system solution with it decouples in s separate systems. The parallelism in this situation arises from this natural decoupling.

A solution method based on this approach involves four steps:

1. Obtain the right-hand side of the reduced system (13.4).
2. Form the Schur complement matrix (13.5).
3. Solve the reduced system (13.4).
4. Back-substitute using (13.3) to obtain the other unknowns.

One linear system solution with the matrix B can be saved by reformulating the algorithm in a more elegant form. Define

$$E' = B^{-1}E \quad \text{and} \quad f' = B^{-1}f.$$

The matrix E' and the vector f' are needed in steps (1) and (2). Then rewrite step (4) as

$$x = B^{-1}f - B^{-1}Ey = f' - E'y,$$

which gives the following algorithm.

ALGORITHM 13.1: Block-Gaussian Elimination

1. *Solve* $BE' = B$, *and* $Bf' = f$ *for* E' *and* f', *respectively*
2. *Compute* $g' = g - Ff'$
3. *Compute* $S = C - FE'$
4. *Solve* $Sy = g'$
5. *Compute* $x = f' - E'y$.

In a practical implementation, all the B_i matrices are factored and then the systems $B_iE_i' = E_i$ and $B_if_i' = f_i$ are solved. In general, many columns in E_i will be zero. These zero columns correspond to interfaces that are not adjacent to subdomain i. Therefore, any efficient code based on the above algorithm should start by identifying the nonzero columns.

13.2.2 PROPERTIES OF THE SCHUR COMPLEMENT

Now the connections between the Schur complement and standard Gaussian elimination will be explored and a few simple properties will be established. Start with the block-LU factorization of A,

$$\begin{pmatrix} B & E \\ F & C \end{pmatrix} = \begin{pmatrix} I & O \\ FB^{-1} & I \end{pmatrix} \begin{pmatrix} B & E \\ O & S \end{pmatrix} \tag{13.6}$$

which is readily verified. The Schur complement can therefore be regarded as the (2,2) block in the U part of the block-LU factorization of A. From the above relation, note that if A is nonsingular, then so is S. Taking the inverse of A with the help of the above equality yields

$$\begin{aligned} \begin{pmatrix} B & E \\ F & C \end{pmatrix}^{-1} &= \begin{pmatrix} B^{-1} & -B^{-1}ES^{-1} \\ O & S^{-1} \end{pmatrix} \begin{pmatrix} I & O \\ -FB^{-1} & I \end{pmatrix} \\ &= \begin{pmatrix} B^{-1} + B^{-1}ES^{-1}FB^{-1} & -B^{-1}ES^{-1} \\ -S^{-1}FB^{-1} & S^{-1} \end{pmatrix}. \end{aligned} \tag{13.7}$$

Observe that S^{-1} is the (2,2) block in the block-inverse of A. In particular, if the original matrix A is Symmetric Positive Definite, then so is A^{-1}. As a result, S is also Symmetric Positive Definite in this case.

Although simple to prove, the above properties are nonetheless important. They are summarized in the following proposition.

PROPOSITION 13.1 *Let A be a nonsingular matrix partitioned as in (13.2) and such that the submatrix B is nonsingular and let R_y be the restriction operator onto the interface variables, i.e, the linear operator defined by*

$$R_y \begin{pmatrix} x \\ y \end{pmatrix} = y.$$

Then the following properties are true.

1. *The Schur complement matrix S is nonsingular.*
2. *If A is SPD, then so is S.*
3. *For any y, $S^{-1}y = R_y A^{-1} \begin{pmatrix} 0 \\ y \end{pmatrix}$.*

The first property indicates that a method that uses the above block Gaussian elimination algorithm is feasible since S is nonsingular. A consequence of the second property is that when A is positive definite, an algorithm such as the Conjugate Gradient algorithm can be used to solve the reduced system (13.4). Finally, the third property establishes a relation which may allow preconditioners for S to be defined based on solution techniques with the matrix A.

13.2.3 SCHUR COMPLEMENT FOR VERTEX-BASED PARTITIONINGS

The partitioning used in Figure 13.3 is edge-based, meaning that a given edge in the graph does not straddle two subdomains. If two vertices are coupled, then they must belong to the same subdomain. From the graph theory point of view, this is perhaps less common than vertex-based partitionings in which a vertex is not shared by two partitions (except when domains overlap). A vertex-based partitioning is illustrated in Figure 13.5.

We will call interface edges all edges that link vertices that do not belong to the same subdomain. In the case of overlapping, this needs clarification. An overlapping edge or vertex belongs to the same subdomain. Interface edges are only those that link a vertex to another vertex which is not in the same subdomain already, whether in the overlapping portion or elsewhere. Interface vertices are those vertices in a given subdomain that are adjacent to an interface edge. For the example of the figure, the interface vertices for subdomain one (bottom, left subsquare) are the vertices labeled 10 to 16. The matrix shown at the bottom of Figure 13.5 differs from the one of Figure 13.4, because here the interface nodes are not relabeled the last in the global labeling as was done in Figure 13.3. Instead, the interface nodes are labeled as the last nodes in each subdomain. The number of interface nodes is about twice that of the edge-based partitioning.

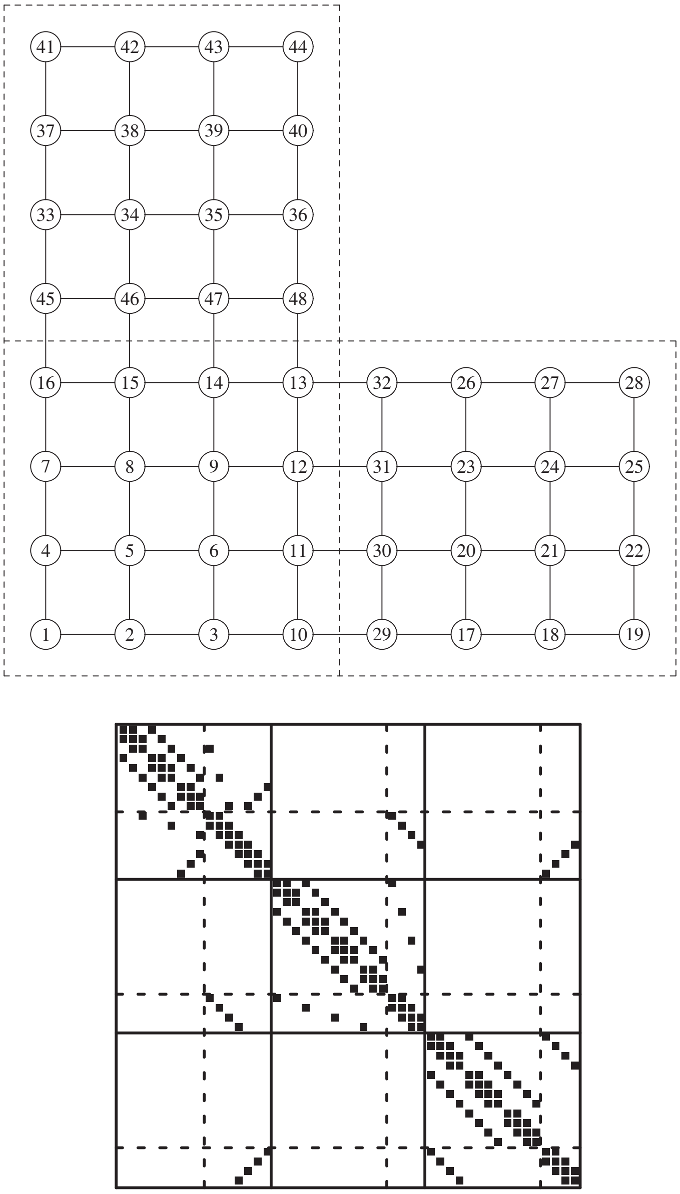


Figure 13.5 *Discretization of problem shown in Figure 13.1 and associated matrix.*

Consider the Schur complement system obtained with this new labeling. It can be written similar to the edge-based case using a reordering in which all interface variables are listed last. The matrix associated with the domain partitioning of the variables will have a natural s-block structure where s is the number of subdomains. For example, when $s = 3$ (as is the case in the above illustration), the matrix has the block structure defined by the solid lines in the figure, i.e.,

$$A = \begin{pmatrix} A_1 & A_{12} & A_{13} \\ A_{21} & A_2 & A_{23} \\ A_{31} & A_{32} & A_3 \end{pmatrix}. \tag{13.8}$$

In each subdomain, the variables are of the form

$$z_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix},$$

where x_i denotes interior nodes while y_i denotes the interface nodes associated with subdomain i. Each matrix A_i will be called the local matrix. The structure of A_i is as follows:

$$A_i = \begin{pmatrix} B_i & E_i \\ F_i & C_i \end{pmatrix} \tag{13.9}$$

in which, as before, B_i represents the matrix associated with the internal nodes of subdomain i and E_i and F_i represent the couplings to/from external nodes. The matrix C_i is the local part of the interface matrix C defined before, and represents the coupling between local interface points. A careful look at the matrix in Figure 13.5 reveals an additional structure for the blocks A_{ij} $j \neq i$. Each of these blocks contains a zero sub-block in the part that acts on the variable x_j. This is expected since x_i and x_j are not coupled. Therefore,

$$A_{ij} = \begin{pmatrix} 0 \\ E_{ij} \end{pmatrix}. \tag{13.10}$$

In addition, most of the E_{ij} matrices are zero since only those indices j of the subdomains that have couplings with subdomain i will yield a nonzero E_{ij}.

Now write the part of the linear system that is local to subdomain i, as

$$\begin{array}{lllllll} B_i x_i & + & E_i y_i & & & = & f_i \\ F_i x_i & + & C_i y_i & + & \sum_{j \in N_i} E_{ij} y_j & = & g_i \end{array}. \tag{13.11}$$

The term $E_{ij} y_j$ is the contribution to the equation from the neighboring subdomain number j, and N_i is the set of subdomains that are adjacent to subdomain i. Assuming that B_i is nonsingular, the variable x_i can be eliminated from this system by extracting from the first equation $x_i = B_i^{-1}(f_i - E_i y_i)$ which yields, upon substitution in the second equation,

$$S_i y_i + \sum_{j \in N_i} E_{ij} y_j = g_i - F_i B_i^{-1} f_i, \quad i = 1, \ldots, s \tag{13.12}$$

in which S_i is the "local" Schur complement

$$S_i = C_i - F_i B_i^{-1} E_i. \tag{13.13}$$

When written for all subdomains i, the equations (13.12) yield a system of equations which involves only the interface points y_j, $j = 1, 2, \ldots, s$ and which has a natural block structure associated with these vector variables

$$S = \begin{pmatrix} S_1 & E_{12} & E_{13} & \cdots & E_{1s} \\ E_{21} & S_2 & E_{23} & \cdots & E_{2s} \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ E_{s1} & E_{s2} & E_{s3} & \cdots & S_s \end{pmatrix}. \tag{13.14}$$

The diagonal blocks in this system, namely, the matrices S_i, are dense in general, but the offdiagonal blocks E_{ij} are sparse and most of them are zero. Specifically, $E_{ij} \neq 0$ only if subdomains i and j have at least one equation that couples them.

A structure of the global Schur complement S has been unraveled which has the following important implication: *For vertex-based partitionings, the Schur complement matrix can be assembled from local Schur complement matrices (the S_i's) and interface-to-interface information (the E_{ij}'s).* The term "assembled" was used on purpose because a similar idea will be exploited for finite element partitionings.

13.2.4 SCHUR COMPLEMENT FOR FINITE-ELEMENT PARTITIONINGS

In finite-element partitionings, the original discrete set Ω is subdivided into s subsets Ω_i, each consisting of a distinct set of elements. Given a finite element discretization of the domain Ω, a finite dimensional space V_h of functions over Ω is defined, e.g., functions that are piecewise linear and continuous on Ω, and that vanish on the boundary Γ of Ω. Consider now the Dirichlet problem on Ω and recall that its weak formulation on the finite element discretization can be stated as follows (see Section 2.3):

$$\text{Find} \quad u \in V_h \quad \text{such that} \quad a(u,v) = (f,v), \quad \forall\ v \in V_h,$$

where the bilinear form $a(.,.)$ is defined by

$$a(u,v) = \int_\Omega \nabla u.\nabla v \ dx = \int_\Omega \left(\frac{\partial u}{\partial x_1}\frac{\partial v}{\partial x_1} + \frac{\partial u}{\partial x_2}\frac{\partial u}{\partial x_2} \right) dx.$$

It is interesting to observe that since the set of the elements of the different Ω_i's are disjoint, $a(.,.)$ can be decomposed as

$$a(u,v) = \sum_{i=1}^{s} a_i(u,v),$$

where

$$a_i(u,v) = \int_{\Omega_i} \nabla u.\nabla v \ dx.$$

In fact, this is a generalization of the technique used to assemble the stiffness matrix from element matrices, which corresponds to the extreme case where each Ω_i consists of exactly one element.

If the unknowns are ordered again by subdomains and the interface nodes are placed last as was done in Section 13.1, immediately the system shows the same structure,

$$\begin{pmatrix} B_1 & & & & E_1 \\ & B_2 & & & E_2 \\ & & \ddots & & \vdots \\ & & & B_s & E_s \\ F_1 & F_2 & \cdots & F_s & C \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_s \\ y \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_s \\ g \end{pmatrix} \tag{13.15}$$

where each B_i represents the coupling between interior nodes and E_i and F_i represent the coupling between the interface nodes and the nodes interior to Ω_i. Note that each of these matrices has been assembled from element matrices and can therefore be obtained from contributions over all subdomain Ω_j that contain any node of Ω_i.

In particular, assume that the assembly is considered only with respect to Ω_i. Then the assembled matrix will have the structure

$$A_i = \begin{pmatrix} B_i & E_i \\ F_i & C_i \end{pmatrix},$$

where C_i contains only contributions from local elements, i.e., elements that are in Ω_i. Clearly, C is the sum of the C_i's,

$$C = \sum_{i=1}^{s} C_i.$$

The Schur complement associated with the interface variables is such that

$$\begin{aligned} S &= C - FB^{-1}E \\ &= C - \sum_{i=1}^{s} F_i B_i^{-1} E_i \\ &= \sum_{i=1}^{s} C_i - \sum_{i=1}^{s} F_i B_i^{-1} E_i \\ &= \sum_{i=1}^{s} \left[C_i - F_i B_i^{-1} E_i \right]. \end{aligned}$$

Therefore, if S_i denotes the *local* Schur complement

$$S_i = C_i - F_i B_i^{-1} E_i,$$

then the above proves that,

$$S = \sum_{i=1}^{s} S_i, \tag{13.16}$$

showing again that the Schur complement can be obtained easily from smaller Schur complement matrices.

Another important observation is that the stiffness matrix A_k, defined above by restricting the assembly to Ω_k, solves a Neumann-Dirichlet problem on Ω_k. Indeed, consider the problem

$$\begin{pmatrix} B_k & E_k \\ F_k & C_k \end{pmatrix} \begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} b_k \\ g_k \end{pmatrix}. \tag{13.17}$$

The elements of the submatrix C_k are the terms $a_k(\phi_i, \phi_j)$ where ϕ_i, ϕ_j are the basis functions associated with nodes belonging to the interface Γ_k. As was stated above, the matrix C is the sum of these submatrices. Consider the problem of solving the Poisson equation on Ω_k with boundary conditions defined as follows: On Γ_{k0}, the part of the boundary which belongs to Γ_k, use the original boundary conditions; on the interfaces Γ_{kj} with other subdomains, use a Neumann boundary condition. According to Equation (2.36) seen in Section 2.3, the j-th equation will be of the form,

$$\int_{\Omega_k} \nabla u . \nabla \phi_j \ \ dx = \int_{\Omega_k} f \phi_j dx + \int_{\Gamma_k} \phi_j \frac{\partial u}{\partial \vec{n}} \, ds. \tag{13.18}$$

This gives rise to a system of the form (13.17) in which the g_k part of the right-hand side incorporates the Neumann data related to the second integral on the right-hand side of (13.18).

It is interesting to note that if a problem were to be solved with all-Dirichlet conditions, i.e., if the Neumann conditions at the interfaces were replaced by Dirichlet conditions, the resulting matrix problem would be of the form,

$$\begin{pmatrix} B_k & E_k \\ 0 & I \end{pmatrix} \begin{pmatrix} x_k \\ y_k \end{pmatrix} = \begin{pmatrix} b_k \\ g_k \end{pmatrix} \tag{13.19}$$

where g_k represents precisely the Dirichlet data. Indeed, according to what was seen in Section 2.3, Dirichlet conditions are handled simply by replacing equations associated with boundary points by identity equations.

13.3 SCHWARZ ALTERNATING PROCEDURES

The original alternating procedure described by Schwarz in 1870 consisted of three parts: alternating between two overlapping domains, solving the Dirichlet problem on one domain at each iteration, and taking boundary conditions based on the most recent solution obtained from the other domain. This procedure is called the Multiplicative Schwarz procedure. In matrix terms, this is very reminiscent of the block Gauss-Seidel iteration with overlap defined with the help of projectors, as seen in Chapter 5. The analogue of the block-Jacobi procedure is known as the Additive Schwarz procedure.

13.3.1 MULTIPLICATIVE SCHWARZ PROCEDURE

In the following, assume that each pair of neighboring subdomains has a nonvoid overlapping region. The boundary of subdomain Ω_i that is included in subdomain j is denoted by $\Gamma_{i,j}$.

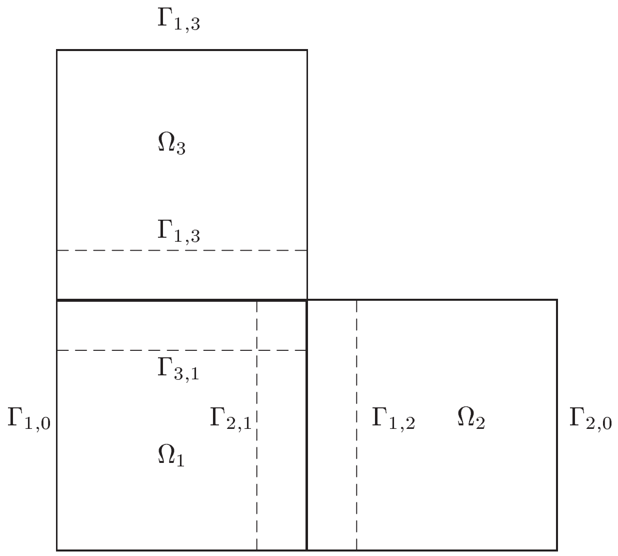


Figure 13.6 *An L-shaped domain subdivided into three overlapping subdomains.*

This is illustrated in Figure 13.6 for the L-shaped domain example. Each subdomain extends beyond its initial boundary into neighboring subdomains. Call Γ_i the boundary of Ω_i consisting of its original boundary (which is denoted by $\Gamma_{i,0}$) and the $\Gamma_{i,j}$'s, and denote by u_{ji} the restriction of the solution u to the boundary Γ_{ji}. Then the Schwarz Alternating Procedure can be described as follows.

ALGORITHM 13.2: SAP

1. *Choose an initial guess* u *to the solution*
2. *Until convergence Do:*
3. *For* $i = 1, \cdots, s$ *Do:*
4. *Solve* $\Delta u = f$ *in* Ω_i *with* $u = u_{ij}$ *in* Γ_{ij}
5. *Update* u *values on* $\Gamma_{ji}, \ \forall j$
6. *EndDo*
7. *EndDo*

The algorithm sweeps through the s subdomains and solves the original equation in each of them by using boundary conditions that are updated from the most recent values of u. Since each of the subproblems is likely to be solved by some iterative method, we can take advantage of a good initial guess. It is natural to take as initial guess for a given subproblem the most recent approximation. Going back to the expression (13.11) of the local problems, observe that each of the solutions in line 4 of the algorithm will be translated into an update of the form

$$u_i := u_i + \delta_i,$$

where the correction δ_i solves the system

$$A_i \delta_i = r_i.$$

Here, r_i is the local part of the most recent global residual vector $b - Ax$, and the above system represents the system associated with the problem in line 4 of the algorithm when a nonzero initial guess is used in some iterative procedure. The matrix A_i has the block structure (13.9). Writing

$$u_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}, \quad \delta_i = \begin{pmatrix} \delta_{x,i} \\ \delta_{y,i} \end{pmatrix}, \quad r_i = \begin{pmatrix} r_{x,i} \\ r_{y,i} \end{pmatrix},$$

the correction to the current solution step in the algorithm leads to

$$\begin{pmatrix} x_i \\ y_i \end{pmatrix} := \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} B_i & E_i \\ F_i & C_i \end{pmatrix}^{-1} \begin{pmatrix} r_{x,i} \\ r_{y,i} \end{pmatrix}. \tag{13.20}$$

After this step is taken, normally a residual vector r would have to be computed again to get the components associated with domain $i + 1$ and to proceed with a similar step for the next subdomain. However, only those residual components that have been affected by the change of the solution need to be updated. Specifically, employing the same notation used in equation (13.11), we can simply update the residual $r_{y,j}$ for each subdomain j for which $i \in N_j$ as

$$r_{y,j} := r_{y,j} - E_{ji}\delta_{y,i}.$$

This amounts implicitly to performing Step 5 of the above algorithm. Note that since the matrix pattern is assumed to be symmetric, then the set of all indices j such that $i \in N_j$, i.e., $N_i^* = \{j \mid i \in N_i\}$, is identical to N_i. Now the loop starting in line 3 of Algorithm 13.2 and called *domain sweep* can be restated as follows.

ALGORITHM 13.3: Multiplicative Schwarz Sweep – Matrix Form

1. *For* $i = 1, \cdots, s$ *Do:*
2. *Solve* $A_i \delta_i = r_i$
3. *Compute* $x_i := x_i + \delta_{x,i}$, $y_i := y_i + \delta_{y,i}$, *and set* $r_i := 0$
4. *For each* $j \in N_i$ *Compute* $r_{y,j} := r_{y,j} - E_{ji}\delta_{y,i}$
5. *EndDo*

Considering only the y iterates, the above iteration would resemble a form of Gauss-Seidel procedure on the Schur complement matrix (13.14). In fact, it is mathematically equivalent, provided a consistent initial guess is taken. This is stated in the next result established by Chan and Goovaerts [48]:

THEOREM 13.1 *Let the guess* $\begin{pmatrix} x_i^{(0)} \\ y_i^{(0)} \end{pmatrix}$ *for the Schwarz procedure in each subdomain be chosen such that*

$$x_i^{(0)} = B_i^{-1}[f_i - E_i y_i^{(0)}]. \tag{13.21}$$

Then the y *iterates produced by the Algorithm 13.3 are identical to those of a Gauss-Seidel sweep applied to the Schur complement system (13.12).*

Proof. We start by showing that with the choice (13.21), the y components of the initial residuals produced by the algorithm are identical to those of the Schur complement system (13.12). Refer to Section 13.2.3 and the relation (13.10) which defines the E_{ij}'s from the block structure (13.8) of the global matrix. Observe that $A_{ij}u_j = E_{ij}y_j$ and note from (13.11) that for the global system the y components of the initial residual vectors are

$$\begin{aligned} r_{y,i}^{(0)} &= g_i - F_i x_i^{(0)} - C_i y_i^{(0)} - \sum_{j \in N_i} E_{ij} y_j^{(0)} \\ &= g_i - F_i B^{-1}[f_i - E_i y_i^{(0)}] - C_i y_i^{(0)} - \sum_{j \in N_i} E_{ij} y_j^{(0)} \\ &= g_i - F_i B^{-1} f_i - S_i y_i^{(0)} - \sum_{j \in N_i} E_{ij} y_j^{(0)}. \end{aligned}$$

This is precisely the expression of the residual vector associated with the Schur complement system (13.12) with the initial guess $y_i^{(0)}$.

Now observe that the initial guess has been selected so that $r_{x,i}^{(0)} = 0$ for all i. Because only the y components of the residual vector are modified, according to line 4 of Algorithm 13.3, this property remains valid throughout the iterative process. By the updating equation (13.20) and the relation (13.7), we have

$$y_i := y_i + S_i^{-1} r_{y,i},$$

which is precisely a Gauss-Seidel step associated with the system (13.14). Note that the update of the residual vector in the algorithm results in the same update for the y components as in the Gauss-Seidel iteration for (13.14). ■

It is interesting to interpret Algorithm 13.2, or rather its discrete version, in terms of projectors. For this we follow the model of the overlapping block-Jacobi technique seen in the previous chapter. Let $\mathcal{S}_i$ be an index set

$$\mathcal{S}_i = \{j_1, j_2, \ldots, j_{n_i}\},$$

where the indices j_k are those associated with the n_i mesh points of the interior of the discrete subdomain Ω_i. Note that as before, the $\mathcal{S}_i$'s form a collection of index sets such that

$$\bigcup_{i=1,\ldots,s} \mathcal{S}_i = \{1, \ldots, n\},$$

and the $\mathcal{S}_i$'s are not necessarily disjoint. Let R_i be a *restriction operator* from Ω to Ω_i. By definition, $R_i x$ belongs to Ω_i and keeps only those components of an arbitrary vector x that are in Ω_i. It is represented by an $n_i \times n$ matrix of zeros and ones. The matrices R_i associated with the partitioning of Figure 13.4 are represented in the three diagrams of Figure 13.7, where each square represents a nonzero element (equal to one) and every other element is a zero. These matrices depend on the ordering chosen for the local problem. Here, boundary nodes are labeled last, for simplicity. Observe that each row of each R_i has exactly one nonzero element (equal to one). Boundary points such as the nodes 36 and 37 are represented several times in the matrices R_1, R_2, and R_3 because of the overlapping of the boundary points. Thus, node 36 is represented in matrices R_1 and R_2, while 37 is represented in all three matrices.

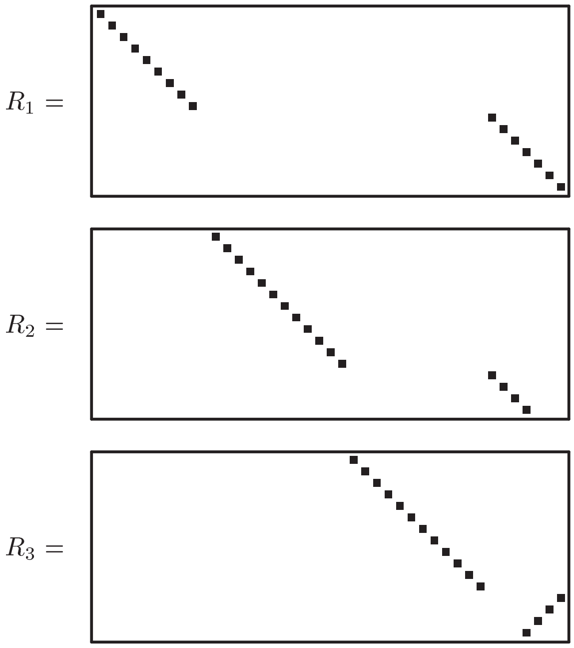


Figure 13.7 *Patterns of the three matrices R_i associated with the partitioning of Figure 13.4.*

From the linear algebra point of view, the restriction operator R_i is an $n_i \times n$ matrix formed by the transposes of columns e_j of the $n \times n$ identity matrix, where j belongs to the index set $\mathcal{S}_i$. The transpose R_i^T of this matrix is a *prolongation operator* which takes a variable from Ω_i and *extends* it to the equivalent variable in Ω. The matrix

$$A_i = R_i A R_i^T$$

of dimension $n_i \times n_i$ defines a restriction of A to Ω_i. Now a problem associated with A_i can be solved which would update the unknowns in the domain Ω_i. With this notation, the multiplicative Schwarz procedure can be described as follows:

1. *For* $i = 1, \ldots, s$ *Do*
2. $\quad x := x + R_i^T A_i^{-1} R_i (b - Ax)$
3. *EndDo*

We change notation and rewrite step 2 as

$$x_{new} = x + R_i^T A_i^{-1} R_i (b - Ax). \tag{13.22}$$

If the errors $d = x_* - x$ are considered where x_* is the exact solution, then notice that $b - Ax = A(x_* - x)$ and, at each iteration the following equation relates the new error d_{new} and the previous error d,

$$d_{new} = d - R_i^T A_i^{-1} R_i A d.$$

Starting from a given x_0 whose error vector is $d_0 = x_* - x$, each sub-iteration produces an error vector which satisfies the relation

$$d_i = d_{i-1} - R_i^T A_i^{-1} R_i A d_{i-1},$$

for $i = 1, \ldots, s$. As a result,

$$d_i = (I - P_i) d_{i-1}$$

in which

$$P_i = R_i^T A_i^{-1} R_i A. \tag{13.23}$$

Observe that the operator $P_i \equiv R_i^T A_i^{-1} R_i A$ is a projector since

$$(R_i^T A_i^{-1} R_i A)^2 = R_i^T A_i^{-1} (R_i A R_i^T) A_i^{-1} R_i A = R_i^T A_i^{-1} R_i A.$$

Thus, one sweep produces an error which satisfies the relation

$$d_s = (I - P_s)(I - P_{s-1}) \ldots (I - P_1) d_0. \tag{13.24}$$

In the following, we use the notation

$$Q_s \equiv (I - P_s)(I - P_{s-1}) \ldots (I - P_1). \tag{13.25}$$

13.3.2 MULTIPLICATIVE SCHWARZ PRECONDITIONING

Because of the equivalence of the multiplicative Schwarz procedure and a block Gauss-Seidel iteration, it is possible to recast one Multiplicative Schwarz sweep in the form of a global fixed-point iteration of the form $x_{new} = Gx + f$. Recall that this is a fixed-point iteration for solving the *preconditioned* system $M^{-1}Ax = M^{-1}b$ where the preconditioning matrix M and the matrix G are related by $G = I - M^{-1}A$. To interpret the operation associated with M^{-1}, it is helpful to identify the result of the error vector produced by this iteration with that of (13.24), which is $x_{new} - x_* = Q_s(x - x_*)$. This comparison yields,

$$x_{new} = Q_s x + (I - Q_s) x_*,$$

and therefore,

$$G = Q_s \quad f = (I - Q_s) x_*.$$

Hence, the preconditioned matrix is $M^{-1}A = I - Q_s$. This result is restated as follows.

PROPOSITION 13.2 *The multiplicative Schwarz procedure is equivalent to a fixed-point iteration for the "preconditioned" problem*

$$M^{-1}Ax = M^{-1}b,$$

in which

$$M^{-1}A = I - Q_s \tag{13.26}$$

$$M^{-1}b = (I - Q_s) x_* = (I - Q_s) A^{-1} b. \tag{13.27}$$

The transformed right-hand side in the proposition is not known explicitly since it is expressed in terms of the exact solution. However, a procedure can be found to compute it. In other words, it is possible to operate with M^{-1} without invoking A^{-1}. Note that $M^{-1} = (I - Q_s)A^{-1}$. As the next lemma indicates, M^{-1}, as well as $M^{-1}A$, can be computed recursively.

LEMMA 13.1 *Define the matrices*

$$Z_i = I - Q_i \tag{13.28}$$
$$M_i = Z_i A^{-1} \tag{13.29}$$
$$T_i = P_i A^{-1} = R_i^T A_i^{-1} R_i \tag{13.30}$$

for $i = 1, \ldots, s$. *Then* $M^{-1} = M_s$, $M^{-1}A = Z_s$, *and the matrices* Z_i *and* M_i *satisfy the recurrence relations*

$$\begin{aligned} Z_1 &= P_1, \\ Z_i &= Z_{i-1} + P_i(I - Z_{i-1}), \quad i = 2, \ldots, s \end{aligned} \tag{13.31}$$

and

$$\begin{aligned} M_1 &= T_1, \\ M_i &= M_{i-1} + T_i(I - AM_{i-1}), \quad i = 2, \ldots, s. \end{aligned} \tag{13.32}$$

Proof. It is clear by the definitions (13.28) and (13.29) that $M_s = M^{-1}$ and that $M_1 = T_1$, $Z_1 = P_1$. For the cases $i > 1$, by definition of Q_i and Q_{i-1}

$$Z_i = I - (I - P_i)(I - Z_{i-1}) = P_i + Z_{i-1} - P_i Z_{i-1}, \tag{13.33}$$

which gives the relation (13.31). Multiplying (13.33) to the right by A^{-1} yields,

$$M_i = T_i + M_{i-1} - P_i M_{i-1}.$$

Rewriting the term P_i as $T_i A$ above yields the desired formula (13.32). ■

Note that (13.31) yields immediately the important relation

$$Z_i = \sum_{j=1}^{i} P_j Q_{j-1}. \tag{13.34}$$

If the relation (13.32) is multiplied to the right by a vector v and if the vector $M_i A^{-1} v$ is denoted by z_i, then the following recurrence results.

$$z_i = z_{i-1} + T_i(v - Az_{i-1}).$$

Since $z_s = (I - Q_s)A^{-1}v = M^{-1}v$, the end result is that $M^{-1}v$ can be computed for an arbitrary vector v, by the following procedure.

ALGORITHM 13.4: Multiplicative Schwarz Preconditioner

1. *Input:* v; *Output:* $z = M^{-1}v$.
2. $z := T_1 v$
3. *For* $i = 2, \ldots, s$ *Do:*

4. $\quad z := z + T_i(v - Az)$
5. *EndDo*

By a similar argument, a procedure can be found to compute vectors of the form $z = M^{-1}Av$. In this case, the following algorithm results:

ALGORITHM 13.5: Multiplicative Schwarz Preconditioned Operator

1. *Input:* v, *Output:* $z = M^{-1}Av$.
2. $z := P_1 v$
3. *For* $i = 2, \ldots, s$ *Do*
4. $\quad z := z + P_i(v - z)$
5. *EndDo*

In summary, the Multiplicative Schwarz procedure is equivalent to solving the "preconditioned system"

$$(I - Q_s)x = g \tag{13.35}$$

where the operation $z = (I - Q_s)v$ can be computed from Algorithm 13.5 and $g = M^{-1}b$ can be computed from Algorithm 13.4. Now the above procedures can be used within an accelerator such as GMRES. First, to obtain the right-hand side g of the preconditioned system (13.35), Algorithm 13.4 must be applied to the original right-hand side b. Then GMRES can be applied to (13.35) in which the preconditioned operations $I - Q_s$ are performed by Algorithm 13.5.

Another important aspect of the Multiplicative Schwarz procedure is that multicoloring can be exploited in the same way as it is done traditionally for block SOR. Finally, note that symmetry is lost in the preconditioned system but it can be recovered by following the sweep 1, 2, ..., s by a sweep in the other direction, namely, $s-1, s-2, \ldots, 1$. This yields a form of the block SSOR algorithm.

13.3.3 ADDITIVE SCHWARZ PROCEDURE

The additive Schwarz procedure is similar to a block-Jacobi iteration and consists of updating all the new (block) components from the same residual. Thus, it differs from the multiplicative procedure only because the components in each subdomain are not updated until a whole cycle of updates through all domains are completed. The basic Additive Schwarz iteration would therefore be as follows:

1. *For* $i = 1, \ldots, s$ *Do*
2. $\quad$ *Compute* $\delta_i = R_i^T A_i^{-1} R_i(b - Ax)$
3. *EndDo*
4. $x_{new} = x + \sum_{i=1}^{s} \delta_i$

The new approximation (obtained after a cycle of the s substeps in the above algorithm are applied) is

$$x_{new} = x + \sum_{i=1}^{s} R_i^T A_i^{-1} R_i (b - Ax).$$

Each instance of the loop redefines different components of the new approximation and there is no data dependency between the subproblems involved in the loop.

The preconditioning matrix is rather simple to obtain for the additive Schwarz procedure. Using the matrix notation defined in the previous section, notice that the new iterate satisfies the relation

$$x_{new} = x + \sum_{i=1}^{s} T_i (b - Ax) = \left(I - \sum_{i=1}^{s} P_i \right) x + \sum_{i=1}^{s} T_i b.$$

Thus, using the same analogy as in the previous section, this iteration corresponds to a fixed-point iteration $x_{new} = Gx + f$ with

$$G = I - \sum_{i=1}^{s} P_i, \quad f = \sum_{i=1}^{s} T_i b.$$

With the relation $G = I - M^{-1}A$, between G and the preconditioning matrix M, the result is that

$$M^{-1}A = \sum_{i=1}^{s} P_i,$$

and

$$M^{-1} = \sum_{i=1}^{s} P_i A^{-1} = \sum_{i=1}^{s} T_i.$$

Now the procedure for applying the preconditioned operator M^{-1} becomes clear.

ALGORITHM 13.6: Additive Schwarz Preconditioner

1. *Input:* v; *Output:* $z = M^{-1}v$.
2. *For* $i = 1, \ldots, s$ *Do:*
3. *Compute* $z_i := T_i v$
4. *EndDo*
5. *Compute* $z := z_1 + z_2 \ldots + z_s$.

Note that the do loop can be performed in parallel. Step 5 sums up the vectors z_i in each domain to obtain a global vector z. In the nonoverlapping case, this step is parallel and consists of just forming these different components since the addition is trivial. In the presence of overlap, the situation is similar except that the overlapping components are added up from the different results obtained in each subdomain.

The procedure for computing $M^{-1}Av$ is identical to the one above except that T_i in line 3 is replaced by P_i.

13.3.4 CONVERGENCE

Throughout this section, it is assumed that A is Symmetric Positive Definite. The projectors P_i defined by (13.23) play an important role in the convergence theory of both additive and multiplicative Schwarz. A crucial observation here is that these projectors are orthogonal with respect to the A-inner product. Indeed, it is sufficient to show that P_i is self-adjoint with respect to the A-inner product,

$$(P_i x, y)_A = (AR_i^T A_i^{-1} R_i A x, y) = (Ax, R_i^T A_i^{-1} R_i A y) = (x, P_i y)_A.$$

Consider the operator,

$$A_J = \sum_{i=1}^{s} P_i. \tag{13.36}$$

Since each P_j is self-adjoint with respect to the A-inner product, i.e., A-self-adjoint, their sum A_J is also A-self-adjoint. Therefore, it will have real eigenvalues. An immediate consequence of the fact that the P_i's are projectors is stated in the following theorem.

THEOREM 13.2 *The largest eigenvalue of A_J is such that*

$$\lambda_{max}(A_J) \le s,$$

where s is the number of subdomains.

Proof. For any matrix norm, $\lambda_{max}(A_J) \le \|A_J\|$. In particular, if the A-norm is used, we have

$$\lambda_{max}(A_J) \le \sum_{i=1}^{s} \|P_i\|_A.$$

Each of the A-norms of P_i is equal to one since P_i is an A-orthogonal projector. This proves the desired result. ∎

This result can be improved substantially by observing that the projectors can be grouped in sets that have disjoint ranges. Graph coloring techniques seen in Chapter 3 can be used to obtain such colorings of the subdomains. Assume that c sets of indices $\Theta_i, i = 1, \ldots, c$ are such that all the subdomains Ω_j for $j \in \Theta_i$ have no intersection with one another. Then,

$$P_{\Theta_i} = \sum_{j \in \Theta_i} P_j \tag{13.37}$$

is again an orthogonal projector.

This shows that the result of the previous theorem can be improved trivially into the following.

THEOREM 13.3 *Suppose that the subdomains can be colored in such a way that two subdomains with the same color have no common nodes. Then, the largest eigenvalue of A_J is such that*

$$\lambda_{max}(A_J) \le c,$$

where c is the number of colors.

In order to estimate the lowest eigenvalue of the preconditioned matrix, an assumption must be made regarding the decomposition of an arbitrary vector x into components of Ω_i.

Assumption 1. *There exists a constant K_0 such that the inequality*

$$\sum_{i=1}^{s}(Au_i, u_i) \ \leq \ K_0(Au, u),$$

is satisfied by the representation of $u \ \in \ \Omega$ as the sum

$$u = \sum_{i=1}^{s} u_i, \quad u_i \ \in \ \Omega_i.$$

The following theorem has been proved by several authors in slightly different forms and contexts.

THEOREM 13.4 *If Assumption 1 holds, then*

$$\lambda_{min}(A_J) \geq \frac{1}{K_0}.$$

Proof. Unless otherwise stated, all summations in this proof are from 1 to s. Start with an arbitrary u decomposed as $u = \sum u_i$ and write

$$(u, u)_A = \sum (u_i, u)_A = \sum (P_i u_i, u)_A = \sum (u_i, P_i u)_A.$$

The last equality is due to the fact that P_i is an A-orthogonal projector onto Ω_i and it is therefore self-adjoint. Now, using Cauchy-Schwarz inequality, we get

$$(u, u)_A = \sum (u_i, P_i u)_A \leq \left(\sum (u_i, u_i)_A\right)^{1/2} \left(\sum (P_i u, P_i u)_A\right)^{1/2}.$$

By Assumption 1, this leads to

$$\|u\|_A^2 \leq K_0^{1/2} \|u\|_A \left(\sum (P_i u, P_i u)_A\right)^{1/2},$$

which, after squaring, yields

$$\|u\|_A^2 \leq K_0 \sum (P_i u, P_i u)_A.$$

Finally, observe that since each P_i is an A-orthogonal projector, we have

$$\sum (P_i u, P_i u)_A = \sum (P_i u, u)_A = \left(\sum P_i u, u\right)_A.$$

Therefore, for any u, the inequality

$$(A_J u, u)_A \geq \frac{1}{K_0}(u, u)_A$$

holds, which yields the desired upper bound by the min-max theorem. ∎

Note that the proof uses the following form of the Cauchy-Schwarz inequality:

$$\sum_{i=1}^{p}(x_i, y_i) \;\le\; \left(\sum_{i=1}^{p}(x_i, x_i)\right)^{1/2} \left(\sum_{i=1}^{p}(y_i, y_i)\right)^{1/2}.$$

See Exercise 1 for a proof of this variation.

We now turn to the analysis of the Multiplicative Schwarz procedure. We start by recalling that the error after each outer iteration (sweep) is given by

$$d = Q_s d_0.$$

We wish to find an upper bound for $\|Q_s\|_A$. First note that (13.31) in Lemma 13.1 results in

$$Q_i = Q_{i-1} - P_i Q_{i-1},$$

from which we get, using the A-orthogonality of P_i,

$$\|Q_i v\|_A^2 = \|Q_{i-1} v\|_A^2 - \|P_i Q_{i-1} v\|_A^2.$$

The above equality is valid for $i = 1$, provided $Q_0 \equiv I$. Summing these equalities from $i = 1$ to s gives the result,

$$\|Q_s v\|_A^2 = \|v\|_A^2 - \sum_{i=1}^{s} \|P_i Q_{i-1} v\|_A^2. \tag{13.38}$$

This indicates that the A-norm of the error will not increase at each substep of the sweep.

Now a second assumption must be made to prove the next lemma.

Assumption 2. *For any subset S of $\{1, 2, \ldots, s\}^2$ and $u_i, v_j \in \Omega$, the following inequality holds:*

$$\sum_{(i,j)\,\in\, S} (P_i v_i, P_j v_j)_A \le K_1 \left(\sum_{i=1}^{s} \|P_i u_i\|_A^2\right)^{1/2} \left(\sum_{j=1}^{s} \|P_j v_j\|_A^2\right)^{1/2}. \tag{13.39}$$

LEMMA 13.2 *If Assumptions 1 and 2 are satisfied, then the following is true,*

$$\sum_{i=1}^{s} \|P_i v\|_A^2 \le (1 + K_1)^2 \sum_{i=1}^{s} \|P_i Q_{i-1} v\|_A^2. \tag{13.40}$$

Proof. Begin with the relation which follows from the fact that P_i is an A-orthogonal projector,

$$(P_i v, P_i v)_A = (P_i v, P_i Q_{i-1} v)_A + (P_i v, (I - Q_{i-1}) v)_A,$$

which yields, with the help of (13.34),

$$\sum_{i=1}^{s} \|P_i v\|_A^2 = \sum_{i=1}^{s} (P_i v, P_i Q_{i-1} v)_A + \sum_{i=1}^{s} \sum_{j=1}^{i-1} (P_i v, P_j Q_{j-1} v)_A. \tag{13.41}$$

For the first term of the right-hand side, use the Cauchy-Schwarz inequality to obtain

$$\sum_{i=1}^{s}(P_i v, P_i Q_{i-1}v)_A \ \le \ \left(\sum_{i=1}^{s}\|P_i v\|_A^2\right)^{1/2}\left(\sum_{i=1}^{s}\|P_i Q_{i-1}v\|_A^2\right)^{1/2}.$$

For the second term of the right-hand side of (13.41), use the assumption (13.39) to get

$$\sum_{i=1}^{s}\sum_{j=1}^{i-1}(P_i v, P_j Q_{j-1}v)_A \le K_1\left(\sum_{i=1}^{s}\|P_i v\|_A^2)\right)^{1/2}\left(\sum_{j=1}^{s}\|P_j Q_{j-1}v\|_A^2)\right)^{1/2}.$$

Adding these two inequalities, squaring the result, and using (13.41) leads to the inequality (13.40). ■

From (13.38), it can be deduced that if Assumption 2 holds, then,

$$\|Q_s v\|_A^2 \le \|v\|_A^2 - \frac{1}{(1+K_1)^2}\sum_{i=1}^{s}\|P_i v\|_A^2. \tag{13.42}$$

Assumption 1 can now be exploited to derive a lower bound on $\sum_{i=1}^{s}\|P_i v\|_A^2$. This will yield the following theorem.

THEOREM 13.5 *Assume that Assumptions 1 and 2 hold. Then,*

$$\|Q_s\|_A \le \left[1 - \frac{1}{K_0(1+K_1)^2}\right]^{1/2}. \tag{13.43}$$

Proof. Using the notation of Section 13.3.3, the relation $\|P_i v\|_A^2 = (P_i v, v)_A$ yields

$$\sum_{i=1}^{s}\|P_i v\|_A^2 = \left(\sum_{i=1}^{s}P_i v,\ v\right)_A = (A_J v, v)_A.$$

According to Theorem 13.4, $\lambda_{min}(A_J) \ge \frac{1}{K_0}$, which implies $(A_J v, v)_A \ge (v,v)_A/K_0$. Thus,

$$\sum_{i=1}^{s}\|P_i v\|_A^2 \ge \frac{(v,v)_A}{K_0},$$

which upon substitution into (13.42) gives the inequality

$$\frac{\|Q_s v\|_A^2}{\|v\|_A^2} \le 1 - \frac{1}{K_0(1+K_1)^2}.$$

The result follows by taking the maximum over all vectors v. ■

This result provides information on the speed of convergence of the multiplicative Schwarz procedure by making two key assumptions. These assumptions are not verifiable from linear algebra arguments alone. In other words, given a linear system, it is unlikely that one can establish that these assumptions are satisfied. However, they are satisfied for equations originating from finite element discretization of elliptic Partial Differential Equations. For details, refer to Drya and Widlund [72, 73, 74] and Xu [230].

13.4 SCHUR COMPLEMENT APPROACHES

Schur complement methods are based on solving the reduced system (13.4) by some preconditioned Krylov subspace method. Procedures of this type involve three steps.

1. *Get the right-hand side* $g' = g - FB^{-1}f$.
2. *Solve the reduced system* $Sy = g'$ *via an iterative method.*
3. *Back-substitute, i.e., compute* x *via (13.3).*

The different methods relate to the way in which step 2 is performed. First observe that the matrix S *need not be formed explicitly* in order to solve the reduced system by an iterative method. For example, if a Krylov subspace method without preconditioning is used, then the only operations that are required with the matrix S are matrix-by-vector operations $w = Sv$. Such operations can be performed as follows.

1. *Compute* $v' = Ev$,
2. *Solve* $Bz = v'$
3. *Compute* $w = Cv - Fz$.

The above procedure involves only matrix-by-vector multiplications and one linear system solution with B. Recall that a linear system involving B translates into s-independent linear systems. Also note that the linear systems with B must be solved exactly, either by a direct solution technique or by an iterative technique with a high level of accuracy.

While matrix-by-vector multiplications with S cause little difficulty, it is much harder to precondition the matrix S, since this full matrix is often not available explicitly. There have been a number of methods, derived mostly using arguments from Partial Differential Equations to precondition the Schur complement. Here, we consider only those preconditioners that are derived from a linear algebra viewpoint.

13.4.1 INDUCED PRECONDITIONERS

One of the easiest ways to derive an approximation to S is to exploit Proposition 13.1 and the intimate relation between the Schur complement and Gaussian elimination. This proposition tells us that a preconditioning operator M to S can be defined from the (approximate) solution obtained with A. To precondition a given vector v, i.e., to compute $w = M^{-1}v$, where M is the desired preconditioner to S, first solve the system

$$A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ v \end{pmatrix}, \tag{13.44}$$

then take $w = y$. Use any approximate solution technique to solve the above system. Let M_A be any preconditioner for A. Using the notation defined earlier, let R_y represent the restriction operator on the interface variables, as defined in Proposition 13.1. Then the preconditioning operation for S which is induced from M_A is defined by

$$M_S^{-1}v = R_y M_A^{-1} \begin{pmatrix} 0 \\ v \end{pmatrix} = R_y M_A^{-1} R_y^T v.$$

Observe that when M_A is an exact preconditioner, i.e., when $M_A = A$, then according to Proposition 13.1, M_S is also an exact preconditioner, i.e., $M_S = S$. This induced preconditioner can be expressed as

$$M_S = \left(R_y M_A^{-1} R_y^T\right)^{-1}. \tag{13.45}$$

It may be argued that this uses a preconditioner related to the original problem to be solved in the first place. However, even though the preconditioning on S may be defined from a preconditioning of A, the linear system is being solved for the interface variables. That is typically much smaller than the original linear system. For example, GMRES can be used with a much larger dimension of the Krylov subspace since the Arnoldi vectors to keep in memory are much smaller. Also note that from a Partial Differential Equations viewpoint, systems of the form (13.44) correspond to the Laplace equation, the solutions of which are "Harmonic" functions. There are fast techniques which provide the solution of such equations inexpensively.

In the case where M_A is an ILU factorization of A, M_S can be expressed in an explicit form in terms of the entries of the factors of M_A. This defines a preconditioner to S that is induced canonically from an incomplete LU factorization of A. Assume that the preconditioner M_A is in a factored form $M_A = L_A U_A$, where

$$L_A = \begin{pmatrix} L_B & 0 \\ FU_B^{-1} & L_S \end{pmatrix} \quad U_A = \begin{pmatrix} U_B & L_B^{-1}E \\ 0 & U_S \end{pmatrix}.$$

Then, the inverse of M_A will have the following structure:

$$\begin{aligned} M_A^{-1} &= U_A^{-1} L_A^{-1} \\ &= \begin{pmatrix} \star & \star \\ 0 & U_S^{-1} \end{pmatrix} \begin{pmatrix} \star & 0 \\ \star & L_S^{-1} \end{pmatrix} \\ &= \begin{pmatrix} \star & \star \\ \star & U_S^{-1} L_S^{-1} \end{pmatrix} \end{aligned}$$

where a star denotes a matrix whose actual expression is unimportant. Recall that by definition,

$$R_y = (\,0 \quad I\,),$$

where this partitioning conforms to the above ones. This means that

$$R_y M_A^{-1} R_y^T = U_S^{-1} L_S^{-1}$$

and, therefore, according to (13.45), $M_S = L_S U_S$. This result is stated in the following proposition.

PROPOSITION 13.3 *Let $M_A = L_A U_A$ be an ILU preconditioner for A. Then the preconditioner M_S for S induced by M_A, as defined by (13.45), is given by*

$$M_S = L_S U_S, \quad \textit{with} \quad L_S = R_y L_A R_y^T, \quad U_S = R_y U_A R_y^T.$$

In words, the proposition states that the L and U factors for M_S are the $(2,2)$ blocks of the L and U factors of the ILU factorization of A. An important consequence of the above idea is that the parallel Gaussian elimination can be exploited for deriving an ILU preconditioner for S by using a general purpose ILU factorization. In fact, the L and U factors of M_A have the following structure:

$$A = L_A U_A - R \quad \text{with,}$$

$$L_A = \begin{pmatrix} L_1 & & & & \\ & L_2 & & & \\ & & \ddots & & \\ & & & L_s & \\ F_1 U_1^{-1} & F_2 U_2^{-1} & \cdots & F_s U_s^{-1} & L \end{pmatrix}$$

$$U_A = \begin{pmatrix} U_1 & & & & L_1^{-1} E_1 \\ & U_2 & & & L_2^{-1} E_2 \\ & & \ddots & & \vdots \\ & & & U_s & L_s^{-1} E_s \\ & & & & U \end{pmatrix}.$$

Each L_i, U_i pair is an incomplete LU factorization of the local B_i matrix. These ILU factorizations can be computed independently. Similarly, the matrices $L_i^{-1} E_i$ and $F_i U_i^{-1}$ can also be computed independently once the LU factors are obtained. Then each of the matrices

$$\tilde{S}_i = C_i - F_i U_i^{-1} L_i^{-1} E_i,$$

which are the approximate local Schur complements, is obtained. Note that since an incomplete LU factorization is being performed, some drop strategy is applied to the elements in $\tilde{S}_i$. Let T_i be the matrix obtained after this is done,

$$T_i = \tilde{S}_i - R_i.$$

Then a final stage would be to compute the ILU factorization of the matrix (13.14) where each S_i is replaced by T_i.

13.4.2 PROBING

To derive preconditioners for the Schur complement, another general purpose technique exploits ideas used in approximating sparse Jacobians when solving nonlinear equations. In general, S is a dense matrix. However, it can be observed, and there are physical justifications for model problems, that its entries decay away from the main diagonal. Assume that S is nearly tridiagonal, i.e., neglect all diagonals apart from the main diagonal and the two codiagonals, and write the corresponding tridiagonal approximation to S as

$$T = \begin{pmatrix} a_1 & b_2 & & & \\ c_2 & a_2 & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & c_{m-1} & a_{m-1} & b_m \\ & & & c_m & a_m \end{pmatrix}.$$

Then, it is easy to recover T by applying it to three well-chosen vectors. Consider the three vectors

$$w_1 = (1,0,0,1,0,0,1,0,0,\ldots,)^T,$$
$$w_2 = (0,1,0,0,1,0,0,1,0,\ldots,)^T,$$
$$w_3 = (0,0,1,0,0,1,0,0,1,\ldots,)^T.$$

Then we have

$$Tw_1 = (a_1, c_2, b_4, a_4, c_5, \ldots, b_{3i+1}, a_{3i+1}, c_{3i+2}, \ldots)^T,$$
$$Tw_2 = (b_2, a_2, c_3, b_5, a_5, c_6, \ldots, b_{3i+2}, a_{3i+2}, c_{3i+3}, \ldots)^T,$$
$$Tw_3 = (b_3, a_3, c_4, b_6, a_6, c_7, \ldots, b_{3i}, a_{3i}, c_{3i+1}, \ldots)^T.$$

This shows that all the coefficients of the matrix T are indeed all represented in the above three vectors. The first vector contains the nonzero elements of the columns 1, 4, 7, ..., $3i + 1$, ..., in succession written as a long vector. Similarly, Tw_2 contains the columns $2, 5, 8, \ldots$, and Tw_3 contains the columns $3, 6, 9, \ldots$. We can easily compute $Sw_i, i = 1, 3$ and obtain a resulting approximation T which can be used as a preconditioner to S. The idea can be extended to compute any banded approximation to S. For details and analysis see [49].

13.4.3 PRECONDITIONING VERTEX-BASED SCHUR COMPLEMENTS

We now discuss some issues related to the preconditioning of a linear system with the matrix coefficient of (13.14) associated with a vertex-based partitioning. As was mentioned before, this structure is helpful in the direct solution context because it allows the Schur complement to be formed by local pieces. Since incomplete LU factorizations will utilize the same structure, this can be exploited as well.

Note that multicolor SOR or SSOR can also be exploited and that graph coloring can be used to color the interface values y_i in such a way that no two adjacent interface variables will have the same color. In fact, this can be achieved by coloring the domains. In the course of a multicolor block-SOR iteration, a linear system must be solved with the diagonal blocks S_i. For this purpose, it is helpful to interpret the Schur complement. Call P the canonical injection matrix from the local interface points to the local nodes. If n_i points are local and if m_i is the number of the local interface points, then P is an $n_i \times m_i$ matrix whose columns are the last m_i columns of the $n_i \times n_i$ identity matrix. Then it is easy to see that

$$S_i = (P^T A_{loc,i}^{-1} P)^{-1}. \tag{13.46}$$

If $A_{loc,i} = LU$ is the LU factorization of $A_{loc,i}$ then it can be verified that

$$S_i^{-1} = P^T U^{-1} L^{-1} P = P^T U^{-1} P P^T L^{-1} P, \tag{13.47}$$

which indicates that in order to operate with $P^T L^{-1} P$, the last $m_i \times m_i$ principal submatrix of L must be used. The same is true for $P^T U^{-1} P$ which requires only a back-solve with the last $m_i \times m_i$ principal submatrix of U. Therefore, only the LU factorization of $A_{loc,i}$ is needed to solve a system with the matrix S_i. Interestingly, approximate solution methods associated with incomplete factorizations of $A_{loc,i}$ can be exploited.

13.5 FULL MATRIX METHODS

We call any technique that iterates on the original system (13.2) a *full matrix method*. In the same way that preconditioners were derived from the LU factorization of A for the Schur complement, preconditioners for A can be derived from approximating interface values.

Before starting with preconditioning techniques, we establish a few simple relations between iterations involving A and S.

PROPOSITION 13.4 *Let*

$$L_A = \begin{pmatrix} I & O \\ FB^{-1} & I \end{pmatrix}, \quad U_A = \begin{pmatrix} B & E \\ O & I \end{pmatrix} \tag{13.48}$$

and assume that a Krylov subspace method is applied to the original system (13.1) with left preconditioning L_A and right preconditioning U_A, and with an initial guess of the form

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} B^{-1}(f - Ey_0) \\ y_0 \end{pmatrix}. \tag{13.49}$$

Then this preconditioned Krylov iteration will produce iterates of the form

$$\begin{pmatrix} x_m \\ y_m \end{pmatrix} = \begin{pmatrix} B^{-1}(f - Ey_m) \\ y_m \end{pmatrix} \tag{13.50}$$

in which the sequence y_m is the result of the same Krylov subspace method applied without preconditioning to the reduced linear system $Sy = g'$ with $g' = g - FB^{-1}f$ starting with the vector y_0.

Proof. The proof is a consequence of the factorization

$$\begin{pmatrix} B & E \\ F & C \end{pmatrix} = \begin{pmatrix} I & O \\ FB^{-1} & I \end{pmatrix} \begin{pmatrix} I & O \\ O & S \end{pmatrix} \begin{pmatrix} B & E \\ O & I \end{pmatrix}. \tag{13.51}$$

Applying an iterative method (e.g., GMRES) on the original system, preconditioned from the left by L_A and from the right by U_A, is equivalent to applying this iterative method to

$$L_A^{-1} A U_A^{-1} = \begin{pmatrix} I & O \\ O & S \end{pmatrix} \equiv A'. \tag{13.52}$$

The initial residual for the preconditioned system is

$$\begin{aligned} L_A^{-1} \begin{pmatrix} f \\ g \end{pmatrix} &- (L_A^{-1} A U_A^{-1}) U_A \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} \\ &= \begin{pmatrix} I & O \\ -FB^{-1} & I \end{pmatrix} \left(\begin{pmatrix} f \\ g \end{pmatrix} - \begin{pmatrix} f \\ FB^{-1}(f - Ey_0) + Cy_0 \end{pmatrix} \right) \end{aligned}$$

$$= \begin{pmatrix} 0 \\ g' - Sy_0 \end{pmatrix} \equiv \begin{pmatrix} 0 \\ r_0 \end{pmatrix}.$$

As a result, the Krylov vectors obtained from the preconditioned linear system associated with the matrix A' have the form

$$\begin{pmatrix} 0 \\ r_0 \end{pmatrix}, \begin{pmatrix} 0 \\ Sr_0 \end{pmatrix} \cdots, \begin{pmatrix} 0 \\ S^{m-1} r_0 \end{pmatrix} \tag{13.53}$$

and the associated approximate solution will be of the form

$$\begin{aligned} \begin{pmatrix} x_m \\ y_m \end{pmatrix} &= \begin{pmatrix} x_0 \\ y_0 \end{pmatrix} + \begin{pmatrix} B^{-1} & -B^{-1}E \\ O & I \end{pmatrix} \begin{pmatrix} 0 \\ \sum_{i=0}^{m-1} \alpha_i S^i r_0 \end{pmatrix} \\ &= \begin{pmatrix} B^{-1}(f - Ey_0) - B^{-1}E(y_m - y_0) \\ y_m \end{pmatrix} \\ &= \begin{pmatrix} B^{-1}(f - Ey_m) \\ y_m \end{pmatrix}. \end{aligned}$$

Finally, the scalars α_i that express the approximate solution in the Krylov basis are obtained implicitly via inner products of vectors among the vector sequence (13.53). These inner products are identical to those of the sequence $r_0, Sr_0, \cdots, S^{m-1}r_0$. Therefore, these coefficients will achieve the same result as the same Krylov method applied to the reduced system $Sy = g'$, if the initial guess gives the residual guess r_0. ■

A version of this proposition should allow S to be preconditioned. The following result is an immediate extension that achieves this goal.

PROPOSITION 13.5 *Let $S = L_S U_S - R$ be an approximate factorization of S and define*

$$L_A = \begin{pmatrix} I & O \\ FB^{-1} & L_S \end{pmatrix}, \quad U_A = \begin{pmatrix} B & E \\ O & U_S \end{pmatrix}. \tag{13.54}$$

Assume that a Krylov subspace method is applied to the original system (13.1) with left preconditioning L_A and right preconditioning U_A, and with an initial guess of the form

$$\begin{pmatrix} x_0 \\ y_0 \end{pmatrix} = \begin{pmatrix} B^{-1}(f - Ey_0) \\ y_0 \end{pmatrix}. \tag{13.55}$$

Then this preconditioned Krylov iteration will produce iterates of the form

$$\begin{pmatrix} x_m \\ y_m \end{pmatrix} = \begin{pmatrix} B^{-1}(f - Ey_m) \\ y_m \end{pmatrix}. \tag{13.56}$$

Moreover, the sequence y_m is the result of the same Krylov subspace method applied to the reduced linear system $Sy = g - FB^{-1}f$, left preconditioned with L_S, right preconditioned with U_S, and starting with the vector y_0.

Proof. The proof starts with the equality

$$\begin{pmatrix} B & E \\ F & C \end{pmatrix} = \begin{pmatrix} I & O \\ FB^{-1} & L_S \end{pmatrix} \begin{pmatrix} I & O \\ O & L_S^{-1} S U_S^{-1} \end{pmatrix} \begin{pmatrix} B & E \\ O & U_S \end{pmatrix}. \tag{13.57}$$

The rest of the proof is similar to that of the previous result and is omitted. ■

Also there are two other versions in which S is allowed to be preconditioned from the left or from the right. Thus, if M_S is a certain preconditioner for S, use the following factorizations

$$\begin{pmatrix} B & E \\ F & C \end{pmatrix} = \begin{pmatrix} I & O \\ FB^{-1} & M_S \end{pmatrix} \begin{pmatrix} I & O \\ O & M_S^{-1}S \end{pmatrix} \begin{pmatrix} B & E \\ O & I \end{pmatrix} \tag{13.58}$$

$$= \begin{pmatrix} I & O \\ FB^{-1} & I \end{pmatrix} \begin{pmatrix} I & O \\ O & SM_S^{-1} \end{pmatrix} \begin{pmatrix} B & E \\ O & M_S \end{pmatrix}, \tag{13.59}$$

to derive the appropriate left or right preconditioners. Observe that when the preconditioner M_S to S is exact, i.e., when $M = S$, then the block preconditioner L_A, U_A to A induced from M_S is also exact.

Although the previous results indicate that a Preconditioned Schur Complement iteration is mathematically equivalent to a certain preconditioned full matrix method, there are some practical benefits in iterating with the nonreduced system. The main benefit involves the requirement in the Schur Complement techniques to compute Sx exactly at each Krylov subspace iteration. Indeed, the matrix S represents the coefficient matrix of the linear system, and inaccuracies in the matrix-by-vector operation may result in loss of convergence. In the full matrix techniques, the operation Sx is never needed explicitly. In addition, this opens up the possibility of preconditioning the original matrix with approximate solves with the matrix B in the preconditioning operation L_A and U_A.

GRAPH PARTITIONING

13.6

The very first task that a programmer faces when solving a problem on a parallel computer, be it a dense or a sparse linear system, is to decide how to map the data into the processors. For shared memory and SIMD computers, directives are often provided to help the user input a desired mapping, among a small set of choices. Distributed memory computers are more general since they allow mapping the data in an arbitrary fashion. However, this added flexibility puts the burden on the user to find good mappings. In particular, when implementing Domain Decomposition ideas on a parallel computer, efficient techniques must be available for partitioning an arbitrary graph. This section gives an overview of the issues and covers a few techniques.

13.6.1 BASIC DEFINITIONS

Consider a general sparse linear system whose adjacency graph is $G = (V, E)$. There are two issues related to the distribution of mapping a general sparse linear system on a number of processors. First, a good partitioning must be found for the original problem. This translates into partitioning the graph G into subgraphs and can be viewed independently from the underlying architecture or topology. The second issue, which is architecture dependent, is to find a good mapping of the subdomains or subgraphs to the processors, after the partitioning has been found. Clearly, the partitioning algorithm can take advantage of a measure of quality of a given partitioning by determining different weight functions for the vertices, for vertex-based partitionings. Also, a good mapping could be found to minimize communication costs, given some knowledge on the architecture.

Graph partitioning algorithms address only the first issue. Their goal is to subdivide the graph into smaller subgraphs in order to achieve a good load balancing of the work among the processors and ensure that the ratio of communication over computation is small for the given task. We begin with a general definition.

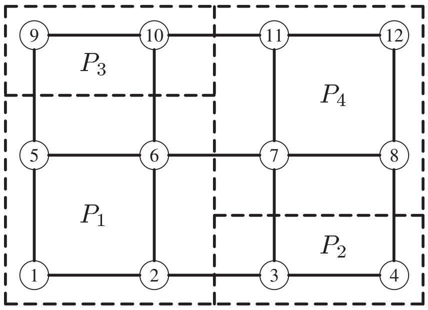


Figure 13.8 *Mapping of a simple* 4×3 *mesh to 4 processors.*

DEFINITION 13.1 *We call a map of V, any set $V_1, V_2, \ldots, V_s$, of subsets of the vertex set V, whose union is equal to V:*

$$V_i \subseteq V, \qquad \bigcup_{i=1,s} V_i = V.$$

When all the V_i subsets are disjoint, the map is called a proper partition; otherwise we refer to it as an overlapping partition.

The most general way to describe a node-to-processor mapping is by setting up a list for each processor, containing all the nodes that are mapped to that processor. Three distinct classes of algorithms have been developed for partitioning graphs. An overview of each of these three approaches is given next.

13.6.2 GEOMETRIC APPROACH

The geometric approach works on the physical mesh and requires the coordinates of the mesh points to find adequate partitionings. In the simplest case, for a 2-dimensional rectangular grid, stripes in the horizontal and vertical direction can be defined to get square subregions which have roughly the same number of points. Other techniques utilize notions of moment of inertia to divide the region recursively into two roughly equal-sized subregions.

Next is a very brief description of a technique based on work by Miller, Teng, Thurston, and Vavasis [150]. This technique finds good separators for a mesh using projections into a higher space. Given a mesh in $\mathbb{R}^d$, the method starts by projecting the mesh points into a unit sphere centered at the origin in $\mathbb{R}^{d+1}$. Stereographic projection is used: A line is drawn from a given point p in the plane to the North Pole $(0, \ldots, 0, 1)$ and the stereographic projection of p is the point where this line intersects the sphere. In the next step, a *centerpoint* of the projected points is found. A centerpoint c of a discrete set S is defined as a point where every hyperplane passing through c will divide S approximately evenly. Once the centerpoint is found, the points of the sphere are rotated so that the centerpoint is aligned with the North Pole, i.e., so that coordinates of c are transformed into $(0, \ldots, 0, r)$. The points are further transformed by dilating them so that the centerpoint becomes the origin. Through all these transformations, the point c remains a centerpoint. Therefore, if any hyperplane is taken that passes through the centerpoint which is now the origin, it should cut the sphere into two roughly equal-sized subsets. Any hyperplane passing through the origin will intersect the sphere along a large circle C. Transforming this circle back into the original space will give a desired separator. Notice that there is an infinity of circles to choose from. One of the main ingredients in the above algorithm is a heuristic for finding centerpoints in $\mathbb{R}^d$ space (actually, $\mathbb{R}^{d+1}$ in the algorithm). The heuristic that is used repeatedly replaces randomly chosen sets of $d + 2$ points by their centerpoint, which are easy to find in this case.

There are a number of interesting results that analyze the quality of geometric graph partitionings based on separators. With some minimal assumptions on the meshes, it is possible to show that there exist "good" separators. In addition, the algorithm discussed above constructs such separators. We start with two definitions.

DEFINITION 13.2 *A k-ply neighborhood system in $\mathbb{R}^d$ is a set of n closed disks D_i, $i = 1, \ldots, n$ in $\mathbb{R}^d$ such that no point in $\mathbb{R}^d$ is (strictly) interior to more than k disks.*

DEFINITION 13.3 *Let $\alpha \geq 1$ and let $D_1, \ldots, D_n$ be a k-ply neighborhood system in $\mathbb{R}^d$. The (α, k)-overlap graph for the neighborhood system is the graph with vertex set $V = \{1, 2, \ldots, n\}$ and edge set, the subset of $V \times V$ defined by*

$$\{(i, j) \; : \; (D_i \cap (\alpha.D_j) \neq \phi) \text{ and } (D_j \cap (\alpha.D_i) \neq \phi)\}.$$

A mesh in $\mathbb{R}^d$ is associated with an overlap graph by assigning the coordinate of the center c_i of disk i to each node i of the graph. Overlap graphs model computational meshes in d dimensions. Indeed, every mesh with bounded *aspect ratio* elements (ratio of largest to smallest edge length of each element) is contained in an overlap graph. In addition, any planar graph is an overlap graph. The main result regarding separators of overlap graphs is the following theorem [150].

THEOREM 13.6 *Let G be an n-vertex (α, k) overlap graph in d dimensions. Then the vertices of G can be partitioned into three sets A, B, and C such that:*

1. *No edge joins A and B.*
2. *A and B each have at most $n(d + 1)/(d + 2)$ vertices.*
3. *C has only $O(\alpha \, k^{1/d} n^{(d-1)/d})$ vertices.*

Thus, for $d = 2$, the theorem states that it is possible to partition the graph into two subgraphs A and B, with a separator C, such that the number of nodes for each of A and B does not exceed $\frac{3}{4}n$ vertices in the worst case and such that the separator has a number of nodes of the order $O(\alpha\ k^{1/2}n^{1/2})$.

13.6.3 SPECTRAL TECHNIQUES

Spectral bisection refers to a technique which exploits some known properties of the eigenvectors of the *Laplacean of a graph*. Given an adjacency graph $G = (V, E)$, we associate to it a Laplacian matrix L which is a sparse matrix having the same adjacency graph G and defined as follows:

$$l_{ij} = \begin{cases} -1 & \text{if}(v_i, v_j) \ \in\ E \text{ and } i \neq j \\ \deg(\mathrm{i}) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

There are some interesting fundamental properties of such matrices. Assuming the graph is undirected, the matrix is symmetric. It can easily be seen that it is also *negative semi definite* (see Exercise 9). Zero is an eigenvalue and it is the smallest one. An eigenvector associated with this eigenvalue is any constant vector, and this eigenvector bears little interest. However, the second smallest eigenvector, called the *Fiedler vector*, has the useful property that the signs of its components divide the domain into roughly two equal subdomains. To be more accurate, the Recursive Spectral Bisection (RSB) algorithm consists of sorting the components of the eigenvector and assigning the first half of the sorted vertices to the first subdomain and the second half to the second subdomain. The two subdomains are then partitioned in two recursively, until a desirable number of domains is reached.

ALGORITHM 13.7: RSB (Recursive Spectral Bisection)

1. *Compute the Fiedler vector f of the graph G.*
2. *Sort the components of f, e.g., increasingly.*
3. *Assign first $\lfloor n/2 \rfloor$ nodes to V_1, and the rest to V_2 .*
4. *Apply RSB recursively to V_1, V_2, until the desired number of partitions*
5. *is reached.*

The main theoretical property that is exploited here is that the differences between the components of the Fiedler vector represent some sort of distance between the corresponding nodes. Thus, if these components are sorted they would be grouping effectively the associated node by preserving nearness. In addition, another interesting fact is that the algorithm will also tend to minimize the number n_c of *cut-edges*, i.e., the number of edges (v_i, v_j) such that $v_i \ \in V_1$ and $v_j \ \in V_2$. Let p be a *partition vector* whose components are $+1$ or -1 in equal number, so that $e^T p = 0$ where $e = (1, 1, \ldots, 1)^T$. Assume that V_1 and V_2 are of equal size and that the components of p are set to $+1$ for those in V_1 and -1 for those in V_2. Then notice that

$$(Lp, p) = 4n_c, \quad (p, e) = 0.$$

Ideally, the objective function (Lp, p) should be minimized subject to the constraint that

$(p, e) = 0$. Note that here p is a vector of signs. If, instead, the objective function $(Lx, x)/(x, x)$ were minimized with respect to the constraint $(x, e) = 0$ for x real, the solution would be the Fiedler vector, since e is the eigenvector associated with the eigenvalue zero. The Fiedler vector is an eigenvector associated with the second smallest eigenvalue of L. This eigenvector can be computed by the Lanczos algorithm or any other method efficient for large sparse matrices. Recursive Specrtal Bisection gives excellent partitionings. On the other hand, it is rather expensive because of the requirement to compute eigenvectors.

13.6.4 GRAPH THEORY TECHNIQUES

There exist a number of other techniques which, like spectral techniques, are also based on the adjacency graph only. The simplest idea is one that is borrowed from the technique of *nested dissection* in the context of direct sparse solution methods. Refer to Chapter 3 where level set orderings are described. An initial node is given which constitutes the level zero. Then, the method recursively traverses the k-th level ($k \geq 1$), which consists of the neighbors of all the elements that constitute level $k - 1$. A simple idea for partitioning the graph in two traverses enough levels to visit about half of all the nodes. The visited nodes will be assigned to one subdomain and the others will constitute the second subdomain. The process can then be repeated recursively on each of the subdomains. A key ingredient for this technique to be successful is to determine a good initial node from which to start the traversal. Often, a heuristic is used for this purpose. Recall that $d(x, y)$ is the distance between vertices x and y in the graph, i.e., the length of the shortest path between x and y. If the diameter of a graph is defined as

$$\delta(G) = \max\{d(x, y) \mid x \in V,\ y \in V\}$$

then, ideally, one of two nodes in a pair (x, y) that achieves the diameter can be used as a starting node. These *peripheral nodes*, are expensive to determine. Instead, a *pseudo-peripheral* node, as defined through the following procedure, is often employed.

ALGORITHM 13.8: Pseudo-Peripheral Node

1. *Select an initial node x. Set $\delta = 0$.*
2. *Do a level set traversal from x*
3. *Select a node y in the last level set, with minimum degree*
4. *If $d(x, y) > \delta$ then*
5.   *Set $x := y$ and $\delta := d(x, y)$*
6.   *GoTo 2*
7.   *Else Stop: x is a pseudo-peripheral node.*
8. *EndIf*

The distance $d(x, y)$ in line 5 is the number of levels in the level set traversal needed in Step 2. The algorithm traverses the graph from a node of the last level in the previous traversal, until the number of levels stabilizes. It is easy to see that the algorithm does indeed stop after a finite number of steps, typically small.

The above algorithm plays a key role in sparse matrix computations. It is very helpful in the context of graph partitioning as well. A first heuristic approach based on level set traversals is the recursive dissection procedure mentioned above and described next.

ALGORITHM 13.9: Recursive Graph Bisection

1. *Set* $G_* := G, S := \{G\}, n_{dom} := 1$
2. *While* $n_{dom} < s$ *Do:*
3. *Select in* S *the subgraph* G_* *with largest size.*
4. *Find a pseudo-peripheral node* p *in* G_* *and*
5. *Do a level set traversal from* p*. Let* $lev :=$ *number of levels.*
6. *Let* G_1 *the subgraph of* G_* *consisting of the first* $lev/2$
7. *levels, and* G_2 *the subgraph containing the rest of* G_*.
8. *Remove* G_* *from* S *and add* G_1 *and* G_2 *to it*
9. $n_{dom} := n_{dom} + 1$
10. *EndWhile*

The cost of this algorithm is rather small. Each traversal of a graph $G = (V, E)$ costs around $|E|$, where $|E|$ is the number of edges (assuming that $|V| = O(|E|)$). Since there are s traversals of graphs whose size decreases by 2 at each step, it is clear that the cost is $O(|E|)$, the order of edges in the original graph.

As can be expected, the results of such an algorithm are not always good. Typically, two qualities that are measured are the sizes of the domains as well as the number of cut-edges. Ideally, the domains should be equal. In addition, since the values at the interface points should be exchanged with those of neighboring processors, their total number, as determined by the number of cut-edges, should be as small as possible. The first measure can be easily controlled in a recursive Graph Bisection Algorithm — for example, by using variants in which the number of nodes is forced to be exactly half that of the original subdomain. The second measure is more difficult to control. Thus, the top part of Figure 13.9 shows the result of the RGB algorithm on a sample finite-element mesh. This is a vertex-based partitioning. The dashed lines are the cut-edges that link two different domains.

An approach that is competitive with the one described above is that of *double striping*. This method uses two parameters p_1, p_2 such that $p_1 p_2 = s$. The original graph is first partitioned into p_1 large partitions, using one-way partitioning, then each of these partitions is subdivided into p_2 partitions similarly. One-way partitioning into p subgraphs consists of performing a level set traversal from a pseudo-peripheral node and assigning each set of roughly n/p consecutive nodes in the traversal to a different subgraph. The result of this approach with $p_1 = p_2 = 4$ is shown in Figure 13.9 on the same graph as before. As can be observed, the subregions obtained by both methods have elongated and twisted shapes. This has the effect of giving a larger number of cut-edges.

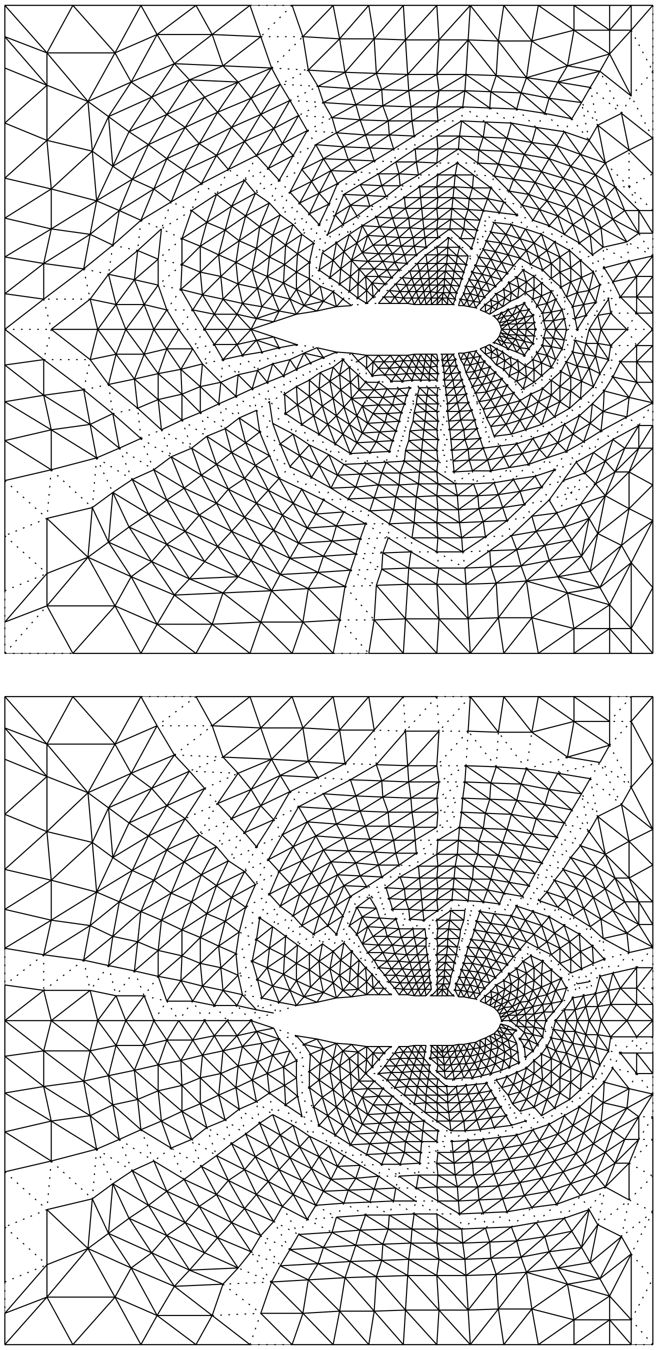


Figure 13.9 *The RGB algorithm (top) and the double-striping algorithm (bottom) for partitioning a graph into 16 subgraphs.*

There are a number of heuristic ways to remedy this. One strategy is based on the fact that a level set traversal from k nodes can be defined instead of only one node. These k nodes are called the *centers* or *sites*. Each subdomain will expand from one of these k centers and the expansion will stop when it is no longer possible to acquire another point that is not already assigned. The boundaries of each domain that are formed this way will tend to be more "circular." To smooth the boundaries of an initial partition, find some center point of each domain and perform a level set expansion from the set of points. The process can be repeated a few times.

ALGORITHM 13.10: Multinode Level-Set Expansion Algorithm

1. *Find a partition* $S = \{G_1, G_2, \ldots, G_s\}$.
2. *For* $iter = 1, \ldots, nouter$ *Do:*
3. *For* $k = 1, \ldots, s$ *Do:*
4. *Find a center* c_k *of* G_k. *Set* $label(c_k) = k$.
5. *EndDo*
6. *Do a level set traversal from* $\{c_1, c_2, \ldots, c_s\}$. *Label each child*
7. *in the traversal with the same label as its parent.*
8. *For* $k = 1, \ldots, s$ *set* G_k *:= subgraph of all nodes having label* k
9. *EndDo*

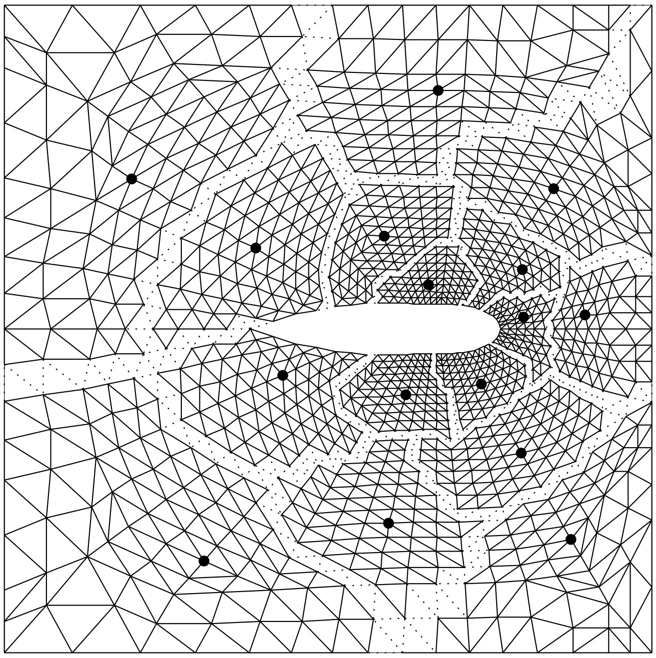


Figure 13.10 *Multinode expansion starting with the partition obtained in Figure 13.9.*

For this method, a total number of cut-edges equal to 548 and a rather small standard deviation of 0.5 are obtained for the example seen earlier.

Still to be decided is how to select the center nodes mentioned in line 4 of the algorithm. Once more, the pseudo-peripheral algorithm will be helpful. Find a pseudo-peripheral node, then do a traversal from it until about one-half of the nodes have been traversed. Then, traverse the latest level set (typically a line or a very narrow graph), and take the middle point as the center.

A typical number of outer steps, *nouter*, to be used in line 2, is less than five. This heuristic works well in spite of its simplicity. For example, if this is applied to the graph obtained from the RGB algorithm, with $nouter = 3$, the partition shown in Figure 13.10 is obtained. With this technique, the resulting total number of cut-edges is equal to 441 and the standard deviation is 7.04. As is somewhat expected, the number of cut-edges has decreased dramatically, while the standard deviation of the various sizes has increased.

EXERCISES

1. In the proof of Theorem 13.4, the following form of the Cauchy-Schwarz inequality was used:

$$\sum_{i=1}^{p}(x_i, y_i) \le \left(\sum_{i=1}^{p}(x_i, x_i)\right)^{1/2}\left(\sum_{i=1}^{p}(y_i, y_i)\right)^{1/2}.$$

(a) Prove that this result is a consequence of the standard Cauchy-Schwarz inequality. (b) Extend the result to the A-inner product. (c) Assume that the x_i's and y_i's are the columns of two $n \times p$ matrix X and Y. Rewrite the result in terms of these matrices.

2. Using Lemma 13.1, write explicitly the vector $M^{-1}b$ for the Multiplicative Schwarz procedure, in terms of the matrix A and the R_i's, when $s = 2$, and then when $s = 3$.

3. (a) Show that in the multiplicative Schwarz procedure, the residual vectors $r_i = b - Ax_i$ obtained at each step satisfy the recurrence,

$$r_i = r_{i-1} - AR_i^T A_i^{-1} R_i r_{i-1}$$

for $i = 1, \ldots, s$. (b) Consider the operator $Q_i \equiv AR_i^T A_i^{-1} R_i$. Show that Q_i is a projector. (c) Is Q_i an orthogonal projector with respect to the A-inner product? With respect to which inner product is it orthogonal?

4. The analysis of the Additive Schwarz procedure assumes that A_i^{-1} is "exact," i.e., that linear systems $A_i x = b$ are solved exactly, each time A_i^{-1} is applied. Assume that A_i^{-1} is replaced by some approximation Θ_i^{-1}. (a) Is P_i still a projector? (b) Show that if Θ_i is Symmetric Positive Definite, then so is P_i. (c) Now make the assumption that $\lambda_{max}(P_i) \le \omega_*$. What becomes of the result of Theorem 13.2?

5. In Element-By-Element (EBE) methods, the extreme cases of the Additive or the Multiplicative Schwarz procedures are considered in which the subdomain partition corresponds to taking Ω_i to be an element. The advantage here is that the matrices do not have to be assembled. Instead, they are kept in unassembled form (see Chapter 2). Assume that Poisson's equation is being solved.

(a) What are the matrices A_i? (b) Are they SPD? (c) Write down the EBE preconditioning corresponding to the multiplicative Schwarz procedure, its multicolor version, and the additive Schwarz procedure.

6. Theorem 13.1 was stated only for the multiplicative version of the Schwarz procedure. There is a similar result for the additive Schwarz procedure. State this result and prove it.

7. Show that the matrix defined by (13.37) is indeed a projector. Is it possible to formulate Schwarz procedures in terms of projection processes as seen in Chapter 5?

8. It was stated at the end of the proof of Theorem 13.4 that if

$$(A_J u, u)_A \geq \frac{1}{C}(u, u)_A$$

for any nonzero u, then $\lambda_{min}(A_J) \geq \frac{1}{C}$. (a) Prove this result without invoking the min-max theory. (b) Prove a version of the min-max theorem with the A-inner product, i.e., prove that the min-max theorem is valid for any inner product for which A is self-adjoint.

9. Consider the Laplacean of a graph as defined in Section 13.6. Show that

$$(Lx, x) = \sum_{(i,j)\,\in\,E} (x_i - x_j)^2.$$

10. Consider a rectangular finite difference mesh, with mesh size $\Delta x = h$ in the x-direction and $\Delta y = h$ closest to the y-direction.

a. To each mesh point $p = (x_i, y_j)$, associate the closed disk D_{ij} of radius h centered at p_i. What is the smallest k such that the family $\{D_{ij}\}$ is a k-ply system?

b. Answer the same question for the case where the radius is reduced to $h/2$. What is the overlap graph (and associated mesh) for any α such that

$$\frac{1}{2} < \alpha < \frac{\sqrt{2}}{2} \ ?$$

What about when $\alpha = 2$?

11. Determine the cost of a level set expansion algorithm starting from p distinct centers.

12. Write a FORTRAN subroutine (or C function) which implements the Recursive Graph Partitioning algorithm.

13. Write recursive versions of the Recursive Graph Partitioning algorithm and Recursive Spectral Bisection algorithm. [Hint: Recall that a recursive program unit is a subprogram or function, say *foo*, which calls itself, so *foo* is allowed to make a subroutine call to *foo* within its body. Recursivity is not allowed in FORTRAN but is possible in C or C++.] (a) Give a pseudo-code for the RGB algorithm which processes the subgraphs in any order. (b) Give a pseudo-code for the RGB algorithm case when the larger subgraph is to be processed before the smaller one in any dissection. Is this second version equivalent to Algorithm 13.9?

NOTES AND REFERENCES. To start with, the original paper by Schwarz is the reference [193], but an earlier note appeared in 1870. In recent years, research on Domain Decomposition techniques has been very active and productive. This rebirth of an old technique has been in large part motivated by parallel processing. However, the first practical use of Domain Decomposition ideas has been in applications to very large structures; see [166, 29], and elasticity problems; see, e.g., [169, 205, 198, 51, 28] for references.

Two recent monographs that describe the use of Domain Decomposition approaches in structural mechanics are [143] and [87]. Recent survey papers include those by Keyes and Gropp [135] and another by Chan and Matthew [50]. The recent volume [136] discusses the various uses of "domain-based" parallelism in computational sciences and engineering.

The bulk of recent work on Domain Decomposition methods has been geared toward a Partial Differential Equations viewpoint. Often, there appears to be a dichotomy between this viewpoint and that of "applied Domain Decomposition," in that the good methods from a theoretical point of view are hard to implement in practice. The Schwarz multiplicative procedure, with multicoloring, represents a compromise between good intrinsic properties and ease of implementation. For example, Venkatakrishnan concludes in [215] that although the use of global coarse meshes may accelerate convergence of local, domain-based, ILU preconditioners, it does not necessarily reduce the overall time to solve a practical aerodynamics problem.

Much is known about the convergence of the Schwarz procedure; refer to the work by Widlund and co-authors [30, 72, 73, 74, 46]. The convergence results of Section 13.3.4 have been adapted from Xu [230] as well as Hackbusch [116]. The result on the equivalence between Schwarz and Schur complement iterations stated in Theorem 13.1 seems to have been originally proved by Chan and Goovaerts [48]. The results on the equivalence between the full matrix techniques and the Schur matrix techniques seen in Section 13.5 have been adapted from results by S. E. Eisenstat, reported in [135]. These connections are rather interesting and useful in practice since they provide some flexibility on ways to implement a method. A number of preconditioners have also been derived using these connections in the PDE framework [32, 31, 33, 34, 35].

Research on graph partitioning is currently very active. So far, variations of the Recursive Spectral Bisection algorithm [165] seem to give the best results in terms of overall quality of the subgraphs. However, the algorithm is rather expensive, and less costly multilevel variations have been developed [119]. Alternatives of the same class as those presented in Section 13.6.4 may be quite attractive for a number of reasons, including cost, ease of implementation, and flexibility; see [107]. There is a parallel between the techniques based on level set expansions and the ideas behind Voronoi diagrams known in computational geometry. The description of the geometric partitioning techniques in Section 13.6.2 is based on the recent papers [105] and [150]. Earlier approaches have been developed in [55, 56, 57]. ■

REFERENCES

1. J. Abaffy and E. Spedicato. *ABS Projection Methods.* Halstead Press, 1989.

2. L. M. Adams. *Iterative algorithms for large sparse linear systems on parallel computers.* Ph.D. thesis, Applied Mathematics, University of Virginia, Charlottsville, VA, 1982. Also NASA Contractor Report 166027.

3. L. M. Adams and H. Jordan. Is SOR color-blind? *SIAM Journal on Scientific and Statistical Computing*, 6:490–506, 1985.

4. L. M. Adams and J. Ortega. A multi-color SOR Method for Parallel Computers. In *Proceedings of the 1982 International Conference on Pararallel Processing*, pages 53–56, 1982.

5. J. I. Aliaga, D. L. Boley, R. W. Freund, and V. Hernández. A Lanczos-type algorithm for multiple starting vectors. Technical Report Numerical Analysis Manuscript No 95-11, AT&T Bell Laboratories, Murray Hill, NJ, 1995.

6. F. L. Alvarado. Manipulation and visualization of sparse matrices. *ORSA Journal on Computing*, 2:186–206, 1990.

7. E. C. Anderson. Parallel implementation of preconditioned conjugate gradient methods for solving sparse systems of linear equations. Technical Report 805, CSRD, University of Illinois, Urbana, IL, 1988. MS Thesis.

8. E. C. Anderson and Y. Saad. Solving sparse triangular systems on parallel computers. *International Journal of High Speed Computing*, 1:73–96, 1989.

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

10. C. C. Ashcraft and R. G. Grimes. On vectorizing incomplete factorization and SSOR preconditioners. *SIAM Journal on Scientific and Statistical Computing*, 9:122–151, 1988.

11. O. Axelsson. A generalized SSOR method. *BIT*, 12:443–467, 1972.

12. O. Axelsson. Conjugate gradient type-methods for unsymmetric and inconsistent systems of linear equations. Technical Report 74-10, CERN, Geneva, 1974.

13. O. Axelsson. Conjugate gradient type-methods for unsymmetric and inconsistent systems of linear equations. *Linear Algebra and its Applications*, 29:1–16, 1980.

14. O. Axelsson. A generalized conjugate gradient, least squares method. *Numerische Mathematik*, 51:209–227, 1987.

15. O. Axelsson. *Iterative Solution Methods*. Cambridge University Press, New York, 1994.

16. O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Value Problems*. Academic Press, Orlando, FL, 1984.

17. O. Axelsson, S. Brinkkemper, and V. P. Ill'n. On some versions of incomplete block-matrix factorization iterative methods. *Linear Algebra and its Applications*, 58:3–15, 1984.

18. O. Axelsson and P. S. Vassilevski. A block generalized conjugate gradient solver with inner iterations and variable step preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 12, 1991.

19. S. Balay, W. D. Gropp, L. Curfman McInnes, and B. F. Smith. PETSc 2.0 users manual. Technical Report ANL-95/11 - Revision 2.0.24, Argonne National Laboratory, 1999.

20. R. E. Bank and T. F. Chan. An analysis of the composite step biconjugate gradient method. *Numerische Mathematik*, 66:259–319, 1993.

21. T. Barth and T. Manteuffel. Variable metric conjugate gradient methods. In *Advances in Numerical Methods for Large Sparse Sets of Linear Equations, Number 10, Matrix Analysis and Parallel Computing, PCG 94*, pages 165–188. Keio University, Yokohama, Japan, 1994.

22. D. Baxter, J. Saltz, M. H. Schultz, S. C. Eisenstat, and K. Crowley. An experimental study of methods for parallel preconditioned Krylov methods. In *Proceedings of the 1988 Hypercube Multiprocessors Conference*, pages 1698–1711. Pasadena, CA, Jan. 1988.

23. M. Benantar and J. E. Flaherty. A six color procedure for the parallel solution of Elliptic systems using the finite quadtree structure. In J. Dongarra, P. Messina, D. C. Sorenson, and R. G. Voigt, editors, *Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing*, pages 230–236, 1990.

24. H. Berryman, J. Saltz, W. Gropp, and R. Mirchandaney. Krylov methods preconditioned with incompletely factored matrices on the CM-2. *Journal of Parallel and Distributed Computing*, 8:186–190, 1990.

25. D. P. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation*. Prentice Hall, Englewood Cliffs, NJ, 1989.

26. G. Birkhoff, R. Varga, S R., and D. Young. Alternating direction implicit methods. In *Advances in Computers*, pages 189–273. Academic Press, New York, 1962.

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

28. P. E. Bjørstad and Anders Hvidsten. Iterative methods for substructured elasticity problems in structural analysis. In Roland Glowinski, Gene H. Golub, Gérard A. Meurant, and Jacques Périaux, editors, *Domain Decomposition Methods for Partial Differential Equations*. SIAM, Philadelphia, PA, 1988.

29. P. E. Bjørstad and O. B. Widlund. Solving elliptic problems on regions partitioned into substructures. In Garrett Birkhoff and Arthur Schoenstadt, editors, *Elliptic Problem Solvers II*, pages 245–256. Academic Press, New York, NY, 1984.

30. P. E. Bjørstad and O. B. Widlund. Iterative methods for the solution of elliptic problems on regions partitioned into substructures. *SIAM Journal on Numerical Analysis*, 23(6):1093–1120, 1986.

31. J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring, I. *Mathematics of Computations*, 47(175):103–134, 1986.

32. J. H. Bramble, J. E. Pasciak, and A. H. Schatz. An iterative method for elliptic problems on regions partitioned into substructures. *Mathematics of Computations*, 46(173):361–369, 1986.

33. J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring, II. *Mathematics of Computations*, 49:1–16, 1987.

34. J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring, III. *Mathematics of Computations*, 51:415–430, 1988.

35. J. H. Bramble, J. E. Pasciak, and A. H. Schatz. The construction of preconditioners for elliptic problems by substructuring, IV. *Mathematics of Computations*, 53:1–24, 1989.

36. R. Bramley and A. Sameh. Row projection methods for large nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13:168–193, 1992.

37. R. Bramley and A. Sameh. A robust parallel solver for block tridiagonal systems. In *Proceedings of the International Conference on Supercomputing*, pages 39–54. ACM, July 1988.

38. C. Brezinski. *Padé Type Approximation and General Orthogonal Polynomials*. Birkhäuser-Verlag, Basel-Boston-Stuttgart, 1980.

39. C. Brezinski and M. Redivo Zaglia. *Extrapolation Methods: Theory and Practice*. North-Holland, Amsterdam, 1991.

40. C. Brezinski and M. Redivo-Zaglia. Hybrid procedures for solving systems of linear equations. *Numerische Mathematik*, 67:1–19, 1994.

41. C. Brezinski, M. Redivo-Zaglia, and H. Sadok. Avoiding breakdown and near-breakdown in Lanczos-type algorithms. *Numerical Algorithms*, 1:261–284, 1991.

42. C. Brezinski, M. Redivo-Zaglia, and H. Sadok. A breakdown-free Lanczos-type algorithm for solving linear systems. *Numerische Mathematik*, 63:29–38, 1992.

43. P. N. Brown. A theoretical comparison of the Arnoldi and GMRES algorithms. *SIAM Journal on Scientific and Statistical Computing*, 12:58–78, 1991.

44. P. N. Brown and A. C. Hindmarsh. Matrix-free methods for stiff systems of ODEs. *SIAM Journal on Numerical Analysis*, 23:610–638, 1986.

45. N. I. Buleev. A numerical method for the solution of two-dimensional and three-dimensional equations of diffusion. *Math. Sb*, 51:227–238, 1960. (in Russian).

46. X. C. Cai and O. Widlund. Multiplicative Schwarz algorithms for some nonsymmetric and indefinite problems. *SIAM Journal on Numerical Analysis*, 30(4), August 1993.

47. T. F. Chan, E. Gallopoulos, V. Simoncini, T. Szeto, and C.H. Tong. A quasi-minimal residual variant of the Bi-CGSTAB algorithm for nonsymmetric systems. *SIAM Journal on Scientific Computing*, 15(2):338–347, 1994.

48. T. F. Chan and D. Goovaerts. On the relationship between overlapping and nonoverlapping domain decomposition methods. *SIAM Journal on Matrix Analysis and Applications*, 13:663–670, 1992.

49. T. F. Chan and T. P. Mathew. The interface probing technique in domain decomposition. *SIAM Journal on Matrix Analysis and Applications*, 13(1):212–238, 1992.

50. T. F. Chan and T. P. Mathew. Domain decomposition algorithms. *Acta Numerica*, pages 61–143, 1994.

51. H. C. Chen and A. Sameh. A matrix decomposition method for orthotropic elasticity problems. *SIAM Journal on Matrix Analysis and Applications*, 10(1):39–64, 1989.

52. C. C. Cheney. *Introduction to Approximation Theory*. McGraw Hill, NY, 1966.

53. E. Chow and Y. Saad. ILUS: an incomplete LU factorization for matrices in sparse skyline format. *International Journal for Numerical Methods in Fluids*, 25:739–748, 1997.

54. E. Chow and Y. Saad. Approximate inverse preconditioners via sparse-sparse iterations. *SIAM Journal on Scientific Computing*, 19:995–1023, 1998.

55. N. Chrisochoides, Geoffrey Fox, and Joe Thompson. MENUS-PGG mapping environment for numerical unstructured and structured parallel grid generation. In *Proceedings of the Seventh International Conference on Domain Decomposition Methods in Scientific and Engineering Computing*, 1993.

56. N. Chrisochoides, C. E. Houstis, E. N. Houstis, P. N. Papachiou, S. K. Kortesis, and J. Rice. DOMAIN DECOMPOSER: a software tool for mapping PDE computations to parallel architectures. In R. Glowinski et. al., editor, *Domain Decomposition Methods for Partial Differential Equations*, pages 341–357. SIAM publications, 1991.

57. N. Chrisochoides, E. Houstis, and J. Rice. Mapping algorithms and software environment for data parallel PDE iterative solvers. *Journal of Parallel and Distributed Computing*, 21:75–95, 1994.

58. G. Cimmino. Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari. *Ric. Sci. Progr. tecn. econom. naz.*, 9:326–333, 1938.

59. A. Clayton. Further results on polynomials having least maximum modulus over an ellipse in the complex plane. Technical Report AEEW-7348, UKAEA, Harewell-UK, 1963.

60. P. Concus and G. H. Golub. A generalized conjugate gradient method for nonsymmetric systems of linear equations. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Sciences and Engineering*, pages 56–65. Springer Verlag, New York, 1976.

61. P. Concus, G. H. Golub, and G. Meurant. Block preconditioning for the conjugate gradient method. *SIAM Journal on Scientific and Statistical Computing*, 6:220–252, 1985.

62. J. D. F. Cosgrove, J. C. Diaz, and A. Griewank. Approximate inverse preconditioning for sparse linear systems. *Interernaitonal Journal of Computational Mathematics*, 44:91–110, 1992.

63. J. Cullum and Anne Greenbaum. Residual relationships within three pairs of iterative algorithms for solving Ax=b. Technical Report IBM Research Report RC 18672, IBM T. J. Watson Research Center, Yorktown Heights, New York, January 1993. To appear, SIAM Journal of Matrix Analysis and Applications, 1996.

64. B. N. Datta. *Numerical Linear Algebra and Applications.* Brooks/Cole Publishing, Pacific Grove, CA, 1995.

65. P. J. Davis. *Interpolation and Approximation.* Blaisdell, Waltham, MA, 1963.

66. T. A. Davis. *A parallel algorithm for sparse unsymmetric LU factorizations.* Ph.D. thesis, University of Illinois at Urbana Champaign, Urbana, IL., 1989.

67. E. F. D'Azevedo, F. A. Forsyth, and W. P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM Journal on Matrix Analysis and Applications*, 13:944–961, 1992.

68. E. F. D'Azevedo, F. A. Forsyth, and W. P. Tang. Towards a cost effective ILU preconditioner with high level fill. *BIT*, 31:442–463, 1992.

69. M. A. DeLong and J. M. Ortega. SOR as a preconditioner. *Applied Numerical Mathematics*, 18:431–440, 1995.

70. P. Deufflhard, R. W. Freund, and A. Walter. Fast secant methods for the iterative solution of large nonsymmetric linear systems. *IMPACT of Computing in Science and Engineering*, 2:244–276, 1990.

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

72. M. Dryja and O. B. Widlund. Some domain decomposition algorithms for elliptic problems. In Linda Hayes and David Kincaid, editors, *Iterative Methods for Large Linear Systems*, pages 273–291. Academic Press, New York, NY, 1989.

73. M. Dryja and O. B. Widlund. Towards a unified theory of domain decomposition algorithms for elliptic problems. In Tony Chan, Roland Glowinski, Jacques Périaux, and O. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, held in Houston, TX, March 20-22, 1989.* SIAM, Philadelphia, PA, 1990.

74. M. Dryja and O. B. Widlund. Additive Schwarz methods for elliptic finite element problems in three dimensions. In T. F. Chan, David E. Keyes, Gérard A. Meurant, Jeffrey S. Scroggs, and Robert G. Voigt, editors, *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations.* SIAM, Philadelphia, PA, 1992.

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

76. I. S. Duff. A survey of sparse matrix research. In *Proceedings of the IEEE, 65*, pages 500–535. Prentice Hall, New York, 1977.

77. I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford, 1986.

78. I. S. Duff, R. G. Grimes, and J. G. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15:1–14, 1989.

79. T. Eirola and O. Nevanlinna. Accelerating with rank-one updates. *Linear Algebra and its Applications*, 121:511–520, 1989.

80. S. Eisenstat. Efficient implementation of a class of conjugate gradient methods. *SIAM Journal on Scientific and Statistical Computing*, 2:1–4, 1988.

81. H. C. Elman. A stability analysis of incomplete LU factorizations. *Mathematics of Computations*, 47:191–217, 1986.

82. H. C. Elman and E. Agron. Ordering techniques for the preconditioned conjugate gradient method on parallel computers. *Computer Physics Communications*, 53:253–269, 1989.

83. H. C. Elman and G. H. Golub. Iterative methods for cyclically reduced non-self-adjoint linear systems. *Mathematics of Computations*, 54:671–700, 1990.

84. M. Engleman. FIDAP manuals. Technical Report Vol. 1, 2, and 3, Fluid Dynamics International, Evanston, IL, 1986.

85. V. Faber and T. Manteuffel. Necessary and sufficient conditions for the existence of a conjugate gradient method. *SIAM Journal on Numerical Analysis*, 21:352–361, 1984.

86. Ky Fan. Note on M-matrices. *Quarterly Journal of Mathematics, Oxford series (2)*, 11:43–49, 1960.

87. C. Farhat and J. X. Roux. Implicit parallel processing in structural mechanics. *Computational Mechanics Advances*, 2(1):1–124, 1994.

88. R. M. Ferencz. *Element-by-element preconditioning techniques for large scale vectorized finite element analysis in nonlinear solid and structural mechanics*. Ph.D. thesis, Department of Applied Mathematics, Stanford, CA, 1989.

89. B. Fischer and R. W. Freund. On the constrained Chebyshev approximation problem on ellipses. *Journal of Approximation Theory*, 62:297–315, 1990.

90. B. Fischer and R. W. Freund. Chebyshev polynomials are not always optimal. *Journal of Approximation Theory*, 65:261–272, 1991.

91. B. Fischer and L. Reichel. A stable Richardson iteration method for complex linear systems. *Numerische Mathematik*, 54:225–241, 1988.

92. R. Fletcher. Conjugate gradient methods for indefinite systems. In G. A. Watson, editor, *Proceedings of the Dundee Biennal Conference on Numerical Analysis 1974*, pages 73–89. Springer Verlag, New York, 1975.

93. R. W. Freund. Conjugate gradient-type methods for linear systems with complex symmetric coefficient matrices. *SIAM Journal on Scientific and Statistical Computing*, 13:425–448, 1992.

94. R. W. Freund. Quasi-kernel polynomials and convergence results for quasi-minimal residual iterations. In Dietriech Braess and Larry L. Schumaker, editors, *Numerical Methods of Approximation Theory, Vol 9*, International series of numerical mathematics, pages 1–19. Birkhaüser Verlag, Basel, 1992.

95. R. W. Freund. A Transpose-Free Quasi-Minimal Residual algorithm for non-Hermitian linear systems. *SIAM Journal on Scientific Computing*, 14(2):470–482, 1993.

96. R. W. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of of the Look-Ahead Lanczos algorithm. *SIAM Journal on Scientific and Statistical Computing*, 14(2):470–482, 1993.

97. R. W. Freund and N. M. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60:315–339, 1991.

98. K. Gallivan, A. Sameh, and Z. Zlatev. A parallel hybrid sparse linear system solver. *Computing Systems in Engineering*, 1(2-4):183–195, June 1990.

99. K. A. Gallivan, M. T. Heath, E. Ng, J. M. Ortega, R. J. Plemmons, C. H. Romine, A. H. Sameh, and R. G. Voigt. *Parallel Algorithms for Matrix Computations*. SIAM, Philadelphia, 1990.

100. F. R. Gantmacher. *The Theory of Matrices*. Chelsea, New York, 1959.

101. N. Gastinel. *Analyse Numérique Linéaire*. Hermann, Paris, 1966.

102. W. Gautschi. On generating orthogonal polynomials. *SIAM Journal on Scientific and Statistical Computing*, 3:289–317, 1982.

103. A. George. Computer implementation of the finite element method. Technical Report STAN-CS-208, Stanford University, Department of Computer Science, 1971.

104. J. A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

105. J. R. Gilbert, G. L. Miller, and S.-H. Teng. Geometric mesh partitioning: Implementation and experiments. In *IPPS 94, Submitted to SIAM SISC*, 1995.

106. S. K. Godunov and G. P. Propkopov. A method of minimal iteration for evaluating the eigenvalues of an elliptic operator. *Zh. Vichsl. Mat. Mat. Fiz.*, 10:1180–1190, 1970.

107. T. Goehring and Y. Saad. Heuristic algorithms for automatic graph partitioning. Technical Report umsi-94-29, University of Minnesota Supercomputer Institute, Minneapolis, MN, February 1994.

108. G. H. Golub and 3rd edn C. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, 1996.

109. G. H. Golub and M. L. Overton. The convergence of inexact Chebyshev and Richardson iterative methods for solving linear systems. *Numerische Mathematik*, 53:571–593, 1988.

110. G. H. Golub and R. S. Varga. Chebyshev semi iterative methods successive overrelaxation iterative methods and second order Richardson iterative methods. *Numerische Mathematik*, 3:147–168, 1961.

111. A. Greenbaum, C. Li, and H. Z. Chao. Parallelizing preconditioned conjugate gradient algorithms. *Computer Physics Communications*, 53:295–309, 1989.

112. M. Grote and H. D. Simon. Parallel preconditioning and approximate inverses on the connection machine. In R. F. Sincovec, D. E. Keyes, L. R. Petzold, and D. A. Reed, editors, *Parallel Processing for Scientific Computing – vol. 2*, pages 519–523. SIAM, 1992.

113. M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms. Part I. *SIAM Journal on Matrix Analysis and Applications*, 13:594–639, 1992.

114. M. H. Gutknecht. A completed theory of the unsymmetric Lanczos process and related algorithms. Part II. *SIAM Journal on Matrix Analysis and Applications*, 15:15–58, 1994.

115. W. Hackbusch. *Multi-Grid Methods and Applications.* Springer Verlag, New York, 1985.

116. W. Hackbusch. *Iterative Solution of Large Linear Systems of Equations.* Springer Verlag, New York, 1994.

117. P. R. Halmos. *Finite-Dimensional Vector Spaces.* Springer Verlag, New York, 1958.

118. S. Hammond and R. Schreiber. Efficient iccg on a shared memory multiprocessor. Technical Report 89. 24, RIACS, NASA Ames research center, Moffett Field CA., 1989.

119. B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. Technical Report SAND92-1460, UC-405, Sandia National Laboratories, Albuquerque, NM, 1992.

120. M. R. Hestenes and E. L. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, Section B, 49:409–436, 1952.

121. C. Hirsch. *Numerical Computation of Internal and External Flows.* John Wiley and Sons, New York, 1988.

122. A. S. Householder. *Theory of Matrices in Numerical Analysis.* Blaisdell Pub. Co., Johnson, CO, 1964.

123. T. J. R. Hughes, R. M. Ferencz, and J. O. Hallquist. Large-scale vectorized implicit calculations in solid mechanics on a Cray X-MP/48 utilizing EBE preconditioned conjugate gradients. *Computer Methods in Applied Mechanics and Engineering*, 61:215–248, 1987.

124. K. Hwang. *Advanced Computer Architecture with Parallel Programming.* Mc Graw Hill, New York, 1993.

125. K. Jbilou. Projection minimization methods for nonsymmetric linear systems. To appear, Linear Algebra and its Applications.

126. K. Jbilou and H. Sadok. Analysis of some vector extrapolation methods for solving systems of linear equations. *Numerische Mathematik*, pages 73–89, 1995.

127. K. C. Jea and D. M. Young. Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods. *Linear Algebra and its Applications*, 34:159–194, 1980.

128. C. Johnson. *Numerical Solutions of Partial Differential Equations by the Finite Element Method.* Cambridge University Press, Cambridge, UK, 1987.

129. O. G. Johnson, C. A. Micchelli, and G. Paul. Polynomial preconditionings for conjugate gradient calculations. *SIAM Journal on Numerical Analysis*, 20:362–376, 1983.

130. S. Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bulletin international de l'Académie polonaise des Sciences et Lettres*, pages 355–357, 1937.

131. C. Kamath and A. Sameh. A projection method for solving nonsymmetric linear systems on multiprocessors. *Parallel Computing*, 9:291–312, 1988.

132. R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–104. Plenum Press, New York, 1972.

133. T. I. Karush, N. K. Madsen, and G. H. Rodrigue. Matrix multiplication by diagonals on vector/parallel processors. Technical Report UCUD, Lawrence Livermore National Lab., Livermore, CA, 1975.

134. D. S. Kershaw. The incomplete Choleski conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26:43–65, 1978.

135. D. E. Keyes and W. D. Gropp. A comparison of domain decomposition techniques for elliptic partial differential equations and their parallel implementation. *SIAM Journal on Scientific and Statistical Computing*, 8(2):s166–s202, 1987.

136. D. E. Keyes, Y. Saad, and D. G. Truhlar. *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*. SIAM, Philadelphia, PA, 1995. (Conference proceedings).

137. L. Yu. Kolotilina and A. Yu. Yeremin. On a family of two-level preconditionings of the incomplete block factorization type. *Soviet Journal of Numerical Analysis and Mathematical Modeling*, 1:293–320, 1986.

138. M. A. Krasnoselskii et al. *Approximate Solutions of Operator Equations*. Wolters-Nordhoff, Groningen, 1972.

139. V. Kumar, A. Grama, A. Gupta, and G. Kapyris. *Parallel Computing*. Benjamin Cummings, Redwood City, CA, 1994.

140. C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *Journal of Research of the National Bureau of Standards*, 45:255–282, 1950.

141. C. Lanczos. Chebyshev polynomials in the solution of large-scale linear systems. In *Proceedings of the ACM*, pages 124–133, 1952.

142. C. Lanczos. Solution of systems of linear equations by minimized iterations. *Journal of Research of the National Bureau of Standards*, 49:33–53, 1952.

143. P. LeTallec. Domain decomposition methods in computational mechanics. *Computational Mechanics Advances*, 1(2):121–220, 1994.

144. R. Leuze. Independent set orderings for parallel matrix factorizations by Gaussian elimination. *Parallel Computing*, 10:177–191, 1989.

145. J. G. Lewis, B. W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorizations. *SIAM Journal on Scientific and Statistical Computing*, 6:1146–1173, 1989.

146. J. G. Lewis and H. D. Simon. The impact of hardware scatter-gather on sparse Gaussian elimination. *SIAM Journal on Scientific and Statistical Computing*, 9:304–311, 1988.

147. S. Ma. *Parallel block preconditioned Krylov subspace methods for Partial Differential Equations*. Ph.D. thesis, Department of Computer Science, Minneapolis, MN, 1993.

148. T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computations*, 34:473–497, 1980.

149. 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. *Mathematics of Computations*, 31(137):148–162, 1977.

150. G. L. Miller, S. H. Teng, W. Thurston, and S. A. Vavasis. Automatic mesh partitioning. In A. George, J. Gilbert, and J. Liu, editors, *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, 1993. IMA Volumes in Mathematics and Its Applications.

151. N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method. *ACM Transactions on Mathematical Software*, 6:206–219, 1980.

152. N. M. Nachtigal. *A look-ahead variant of the Lanczos Algorithm and its application to the Quasi-Minimal Residual method for non-Hermitian linear systems*. Ph.D. thesis, Applied Mathematics, Cambridge, 1991.

153. J. Ortega. Efficient implementation of certain iterative methods. *SIAM Journal on Scientific and Statistical Computing*, 9:882–891, 1988.

154. J. Ortega. Orderings for conjugate gradient preconditionings. *SIAM Journal on Scientific and Statistical Computing*, 12:565–582, 1991.

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

156. J. M. Ortega and R. G. Voigt. Solution of partial differential equations on vector and parallel computers. *SIAM Review*, 27:149–240, 1985.

157. O. Osterby and Z. Zlatev. *Direct Methods for Sparse Matrices*. Springer Verlag, New York, 1983.

158. C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *Journal of the Institute of Mathematics and its Applications*, 10:373–381, 1972.

159. C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12:617–624, 1975.

160. B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice Hall, Englewood Cliffs, 1980.

161. B. N. Parlett, D. R. Taylor, and Z. S. Liu. A look-ahead Lanczos algorithm for nonsymmetric matrices. *Mathematics of Computation*, 44:105–124, 1985.

162. D. Peaceman and H. Rachford. The numerical solution of elliptic and parabolic differential equations. *Journal of SIAM*, 3:28–41, 1955.

163. S. Pissanetzky. *Sparse Matrix Technology*. Academic Press, New York, 1984.

164. E. L Poole and J. M. Ortega. Multicolor ICCG methods for vector computers. *SIAM Journal on Numerical Analysis*, 24:1394–1418, 1987.

165. A. Pothen, H. D. Simon, and K. P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11:430–452, 1990.

166. J. S. Przemieniecki. Matrix structural analysis of substructures. *Am. Inst. Aero. Astro. J.*, 1:138–147, 1963.

167. J. K. Reid. On the method of conjugate gradients for the solution of large sparse systems of linear equations. In J. K. Reid, editor, *Large Sparse Sets of Linear Equations*, pages 231–254. Academic Press, 1971.

168. T. J. Rivlin. *The Chebyshev Polynomials: from Approximation Theory to Algebra and Number Theory*. J. Wiley and Sons, New York, 1990.

169. F. X. Roux. Acceleration of the outer conjugate gradient by reorthogonalization for a domain decomposition method for structural analysis problems. In Tony Chan and Roland Glowinski, editors, *Proceedings of the Third International Symposium on Domain Decomposition Methods, Houston March 20-22 1989*. SIAM, Philadelphia, PA, 1990.

170. A. Ruhe. Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices. *Mathematics of Computations*, 33:680–687, 1979.

171. H. Rutishauser. Theory of gradient methods. In *Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems*, pages 24–49. Institute of Applied Mathematics, Zurich, Birkhäuser Verlag, Basel-Stuttgart, 1959.

172. Y. Saad. Krylov subspace methods for solving large unsymmetric linear systems. *Mathematics of Computation*, 37:105–126, 1981.

173. Y. Saad. The Lanczos biorthogonalization algorithm and other oblique projection methods for solving large unsymmetric systems. *SIAM Journal on Numerical Analysis*, 19:470–484, 1982.

174. Y. Saad. Iterative solution of indefinite symmetric systems by methods using orthogonal polynomials over two disjoint intervals. *SIAM Journal on Numerical Analysis*, 20:784–811, 1983.

175. Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM Journal on Scientific and Statistical Computing*, 6:865–881, 1985.

176. Y. Saad. Least squares polynomials in the complex plane and their use for solving sparse nonsymmetric linear systems. *SIAM Journal on Numerical Analysis*, 24:155–169, 1987.

177. Y. Saad. On the Lanczos method for solving symmetric linear systems with several right-hand sides. *Mathematics of Computations*, 48:651–662, 1987.

178. Y. Saad. Krylov subspace methods on supercomputers. *SIAM Journal on Scientific and Statistical Computing*, 10:1200–1232, 1989.

179. Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.

180. Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Halstead Press, New York, 1992.

181. Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific and Statistical Computing*, 14:461–469, 1993.

182. Y. Saad. Highly parallel preconditioners for general sparse matrices. In G. Golub, M. Luskin, and A. Greenbaum, editors, *Recent Advances in Iterative Methods, IMA Volumes in Mathematics and Its Applications*, volume 60, pages 165–199. Springer Verlag, New York, 1994.

183. Y. Saad. ILUT: a dual threshold incomplete ILU factorization. *Numerical Linear Algebra with Applications*, 1:387–402, 1994.

184. Y. Saad. Analysis of augmented Krylov subspace techniques. *SIAM Journal on Matrix Analysis and Applications*, 18:435–449, 1997.

185. Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

186. Y. Saad and M. H. Schultz. Parallel implementations of preconditioned conjugate gradient methods. In W. E. Fitzgibbon, editor, *Mathematical and Computational Methods in Seismic Exploration and Reservoir Modeling*. SIAM, Philadelphia, PA, 1986.

187. Y. Saad and K. Wu. DQGMRES: a direct quasi-minimal residual algorithm based on incomplete orthogonalization. *Numerical Linear Algebra with Applications*, 3:329–343, 1996.

188. H. Sadok. *Méthodes de projection pour les systèmes linéaires et non linéaires.* Ph.D. thesis, University of Lille 1, Lille, France, 1981.

189. J. Saltz, R. Mirchandaney, and K. Crowley. Run-time paralellization and scheduling of loops. *IEEE Transactions on Computers*, 40:603–612, 1991.

190. J. H. Saltz. Automated problem scheduling and reduction of synchronization delay effects. Technical Report 87-22, ICASE, Hampton, VA, 1987.

191. W. Schonauer. The efficient solution of large linear systems, resulting from the fdm for 3-d PDE's, on vector computers. In Proceedings of the 1-st International colloquium on vector and parallel computing in Scientific applications - Paris March 1983, 1983.

192. W. Schönauer. *Scientific Computing on Vector Computers.* North-Holland, New York, 1987.

193. H. A. Schwarz. *Gesammelte Mathematische Abhandlungen*, volume 2, pages 133–143. Springer Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc., 1890. First published in Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich, volume 15, 1870, pp. 272–286.

194. F. Shakib. *Finite element analysis of the compressible Euler and Navier Stokes Equations.* Ph.D. thesis, Department of Aeronautics, Stanford, CA, 1989.

195. A. Sidi. Extrapolation vs. projection methods for linear systems of equations. *Journal of Computational and Applied Mathematics*, 22:71–88, 1988.

196. V. Simoncini and E. Gallopoulos. An iterative method for nonsymmetric systems with multiple right-hand sides. *SIAM Journal on Scientific Computing*, 16(4):917–933, July 1995.

197. V. Simoncini and E. Gallopoulos. Convergence properties of block GMRES and matrix polynomials. *Linear Algebra and its Applications*, To appear.

198. B. F. Smith. An optimal domain decomposition preconditioner for the finite element solution of linear elasticity problems. *SIAM Journal on Scientific and Statistical Computing*, 13:364–378, 1992.

199. D. A. Smith, W. F. Ford, and A. Sidi. Extrapolation methods for vector sequences. *SIAM review*, 29:199–233, 1987.

200. D. C. Smolarski and P. E. Saylor. An optimum iterative method for solving any linear system with a square matrix. *BIT*, 28:163–178, 1988.

201. P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10(1):36–52, 1989.

202. G. W. Stewart. *Introduction to Matrix Computations.* Academic Press, New York, 1973.

203. W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, Princeton, NJ, 1994.

204. E. L. Stiefel. Kernel polynomials in linear algebra and their applications. *U. S. National Bureau of Standards, Applied Mathematics Series*, 49:1–24, 1958.

205. Patrick Le Tallec, Yann-Hervé De Roeck, and Marina Vidrascu. Domain-decomposition methods for large linearly elliptic three dimensional problems. *Journal of Computational and Applied Mathematics*, 34, 1991. Elsevier Science Publishers, Amsterdam.

206. D. Taylor. *Analysis of the look-ahead Lanczos algorithm.* Ph.D. thesis, Department of Computer Science, Berkeley, CA, 1983.

207. L. N. Trefethen. Approximation theory and numerical linear algebra. Technical Report Numerical Analysis Report 88-7, Massachussetts Institute of Technology, Cambridge, MA, 1988.

208. H. A. van der Vorst. The performance of FORTRAN implementations for preconditioned conjugate gradient methods on vector computers. *Parallel Computing*, 3:49–58, 1986.

209. H. A. van der Vorst. Large tridiagonal and block tridiagonal linear systems on vector and parallel computers. *Parallel Computing*, 5:303–311, 1987.

210. 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 Journal on Scientific and Statistical Computing*, 12:631–644, 1992.

211. H. A. van der Vorst and C. Vuik. GMRESR: a family of nested GMRES methods. *Numerical Linear Algebra with Applications*, 1:369–386, 1994.

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

213. R. S. Varga. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1962.

214. V. Venkatakrishnan. Preconditioned Conjugate Gradient methods for the compressible Navier Stokes equations. *AIAA Journal*, 29:1092–1100, 1991.

215. V. Venkatakrishnan. Parallel implicit methods for aerodynamic applications on unstructured grids. In D. E. Keyes, Y. Saad, and D. G. Truhlar, editors, *Domain-Based Parallelism and Problem Decomposition Methods in Computational Science and Engineering*, pages 57–74. SIAM, Philadelphia, PA, 1995.

216. V. Venkatakrishnan and D. J. Mavriplis. Implicit solvers for unstructured meshes. *Journal of Computational Physics*, 105:83–91, 1993.

217. V. Venkatakrishnan, H. D. Simon, and T. J. Barth. A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids. *The Journal of Supercomputing*, 6:117–137, 1992.

218. P. K. W. Vinsome. ORTHOMIN, an iterative method for solving sparse sets of simultaneous linear equations. In *Proceedings of the Fourth Symposium on Resevoir Simulation*, pages 149–159. Society of Petroleum Engineers of AIME, 1976.

219. V. V. Voevodin. The problem of a non-selfadjoint generalization of the conjugate gradient method has been closed. *USSR Computational Mathematics and Mathematical Physics*, 23:143–144, 1983.

220. E. L. Wachspress. *Iterative Solution of Elliptic Systems and Applications to the Neutron Equations of Reactor Physics*. Prentice Hall, Englewood Cliffs, NJ, 1966.

221. H. F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific Computing*, 9:152–163, 1988.

222. X. Wang, K. Gallivan, and R. Bramley. CIMGS: An incomplete orthogonal factorization preconditioner. Technical Report 394, Indiana University at Bloomington, Bloomington, IN, 1993.

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

224. R. Weiss. A theoretical overview of Krylov subspace methods. In W. Schönauer and R. Weiss, editors, *Special Issue on Iterative Methods for Linear Systems*, pages 33–56. Applied Numerical Methods, 1995.

225. O. Widlund. A Lanczos method for a class of non-symmetric systems of linear equations. *SIAM Journal on Numerical Analysis*, 15:801–812, 1978.

226. L. B. Wigton. Application of MACSYMA and sparse matrix technology to multi-element airfoil calculations. In *Proceedings of the AIAA-87 conference, Honolulu, Hawai, June 9-11, 1987*, pages 444–457. AIAA, New York, 1987. Paper number AIAA-87-1142-CP.

227. J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Clarendon Press, Oxford, 1965.

228. O. Wing and J. W. Huang. A computation model of parallel solution of linear equations. *IEEE Transactions on Computers*, C-29:632–638, 1980.

229. C. H. Wu. A multicolor SOR method for the finite-element method. *Journal of Computational and Applied Mathematics*, 30:283–294, 1990.

230. J. Xu. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34:581–613, December 1992.

231. Q. Ye. A breakdown-free variation of the Lanczos algorithm. *Mathematics of Computations*, 62:179–207, 1994.

232. D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, New York, 1971.

233. D. P. Young, R. G. Melvin, F. T. Johnson, J. E. Bussoletti, L. B. Wigton, and S. S. Samant. Application of sparse matrix solvers as effective preconditioners. *SIAM Journal on Scientific and Statistical Computing*, 10:1186–1199, 1989.

234. L. Zhou and H. F. Walker. Residual smoothing techniques for iterative methods. *SIAM Journal on Scientific Computing*, 15:297–312, 1994.

235. Z. Zlatev. Use of iterative refinement in the solution of sparse linear systems. *SIAM Journal on Numerical Analysis*, 19:381–399, 1982.

INDEX