Journal of Computational Physics **182,** 418–477 (2002)
doi:10.1006/jcph.2002.7176

Preconditioning Techniques for Large Linear Systems: A Survey

Michele Benzi

Mathematics and Computer Science Department, Emory University, Atlanta, Georgia 30322
E-mail: benzi@mathcs.emory.edu

Received April 17, 2002; revised July 23, 2002

This article surveys preconditioning techniques for the iterative solution of large linear systems, with a focus on algebraic methods suitable for general sparse matrices. Covered topics include progress in incomplete factorization methods, sparse approximate inverses, reorderings, parallelization issues, and block and multilevel extensions. Some of the challenges ahead are also discussed. An extensive bibliography completes the paper. © 2002 Elsevier Science (USA)

Key Words: linear systems; sparse matrices; iterative methods; algebraic preconditioners; incomplete factorizations; sparse approximate inverses; unstructured grids; multilevel methods; parallel computing; orderings; block algorithms.

1. INTRODUCTION

The solution of large sparse linear systems of the form

$$Ax = b, \tag{1}$$

where $A = [a_{ij}]$ is an $n \times n$ matrix and b a given right-hand-side vector, is central to many numerical simulations in science and engineering and is often the most time-consuming part of a computation. While the main source of large matrix problems remains the discretization (and linearization) of partial differential equations (PDEs) of elliptic and parabolic type, large and sparse linear systems also arise in applications not governed by PDEs. These include the design and computer analysis of circuits, power system networks, chemical engineering processes, economics models, and queueing systems.

Direct methods, based on the factorization of the coefficient matrix A into easily invertible matrices, are widely used and are the solver of choice in many industrial codes, especially where reliability is the primary concern. Indeed, direct solvers are very robust, and they tend to require a predictable amount of resources in terms of time and storage [121, 150]. With a state-of-the-art sparse direct solver (see, e.g., [5]) it is possible to efficiently solve in a reasonable amount of time linear systems of fairly large size, particularly when the underlying problem is two dimensional. Direct solvers are also the method of choice in certain areas not governed by PDEs, such as circuits, power system networks, and chemical plant modeling.

Unfortunately, direct methods scale poorly with problem size in terms of operation counts and memory requirements, especially on problems arising from the discretization of PDEs in three space dimensions. Detailed, three-dimensional multiphysics simulations (such as those being carried out as part of the U.S. Department of Energy's ASCI program) lead to linear systems comprising hundreds of millions or even billions of equations in as many unknowns. For such problems, iterative methods are the only option available. Even without considering such extremely large-scale problems, systems with several millions of unknowns are now routinely encountered in many applications, making the use of iterative methods virtually mandatory. While iterative methods require fewer storage and often require fewer operations than direct methods (especially when an approximate solution of relatively low accuracy is sought), they do not have the reliability of direct methods. In some applications, iterative methods often fail and preconditioning is necessary, though not always sufficient, to attain convergence in a reasonable amount of time.

It is worth noting that in some circles, especially in the nuclear power industry and the petroleum industry, iterative methods have always been popular. Indeed, these areas have historically provided the stimulus for much early research on iterative methods, as witnessed in the classic monographs [282, 286, 294]. In contrast, direct methods have been traditionally preferred in the areas of structural analysis and semiconductor device modeling, in most parts of computational fluid dynamics (CFD), and in virtually all applications not governed by PDEs. However, even in these areas iterative methods have made considerable gains in recent years.

To be fair, the traditional classification of solution methods as being either direct or iterative is an oversimplification and is not a satisfactory description of the present state of affairs. First, the boundaries between the two classes of methods have become increasingly blurred, with a number of ideas and techniques from the area of sparse direct solvers being transferred (in the form of preconditioners) to the iterative camp, with the result that iterative methods are becoming more and more reliable. Second, while direct solvers are almost invariably based on some version of Gaussian elimination, the field of iterative methods comprises a bewildering variety of techniques, ranging from truly iterative methods, like the classical Jacobi, Gauss–Seidel, and SOR iterations, to Krylov subspace methods, which theoretically converge in a finite number of steps in exact arithmetic, to multilevel methods. To lump all these techniques under a single heading is somewhat misleading, especially when preconditioners are added to the picture.

The focus of this survey is on preconditioning techniques for improving the performance and reliability of Krylov subspace methods. It is widely recognized that preconditioning is the most critical ingredient in the development of efficient solvers for challenging problems in scientific computation, and that the importance of preconditioning is destined to increase even further. The following excerpt from the textbook [272, p. 319] by Trefethen and Bau aptly underscores this point:

> In ending this book with the subject of preconditioners, we find ourselves at the philosophical center of the scientific computing of the future.... Nothing will be more central to computational science in the next century than the art of transforming a problem that appears intractable into another whose solution can be approximated rapidly. For Krylov subspace matrix iterations, this is preconditioning.

Indeed, much effort has been put in the development of effective preconditioners, and preconditioning has been a more active research area than either direct solution methods or Krylov subspace methods for the past few years. Because an optimal general-purpose preconditioner is unlikely to exist, this situation is probably not going to change in the foreseeable future.

As is well known, the term *preconditioning* refers to transforming the system (1) into another system with more favorable properties for iterative solution. A *preconditioner* is a matrix that effects such a transformation. Generally speaking, preconditioning attempts to improve the spectral properties of the coefficient matrix. For symmetric positive definite (SPD) problems, the rate of convergence of the conjugate gradient method depends on the distribution of the eigenvalues of A. Hopefully, the transformed (preconditioned) matrix will have a smaller spectral condition number, and/or eigenvalues clustered around 1. For nonsymmetric (nonnormal) problems the situation is more complicated, and the eigenvalues may not describe the convergence of nonsymmetric matrix iterations like GMRES (see [161]). Nevertheless, a clustered spectrum (away from 0) often results in rapid convergence, particularly when the preconditioned matrix is close to normal.

If M is a nonsingular matrix that approximates A (in some sense), then the linear system

$$M^{-1}Ax = M^{-1}b \tag{2}$$

has the same solution as (1) but may be easier to solve. Here M is the preconditioner. In cases where M^{-1} is explicitly known (as with polynomial preconditioners or sparse approximate inverses), the preconditioner is M^{-1} rather than M.

System (2) is preconditioned from the left, but one can also precondition from the right:

$$AM^{-1}y = b, \quad x = M^{-1}y. \tag{3}$$

When Krylov subspace methods are used, it is not necessary to form the preconditioned matrices $M^{-1}A$ or AM^{-1} explicitly (this would be too expensive, and we would lose sparsity). Instead, matrix–vector products with A and solutions of linear systems of the form $Mz = r$ are performed (or matrix–vector products with M^{-1} if this is explicitly known).

In addition, *split preconditioning* is also possible, i.e.,

$$M_1^{-1}AM_2^{-1}y = M_1^{-1}b, \quad x = M_2^{-1}y, \tag{4}$$

where the preconditioner is now $M = M_1M_2$. Which type of preconditioning to use depends on the choice of the iterative method, problem characteristics, and so forth. For example, with residual minimizing methods, like GMRES, right preconditioning is often used. In exact arithmetic, the residuals for the right-preconditioned system are identical to the true residuals $r_k = b - Ax_k$.

Notice that the matrices $M^{-1}A$, AM^{-1}, and $M_1^{-1}AM_2^{-1}$ are all similar and therefore have the same eigenvalues. If A and M are SPD, the convergence of the CG method will be the same (except possibly for round-off effects) in all cases. On the other hand, in the nonnormal case, solvers like GMRES can behave very differently depending on whether a given preconditioner is applied on the left or on the right (see the discussion in [251, p. 255]; see also [202, p. 66] for a striking example).

In general, a good preconditioner M should meet the following requirements:

- The preconditioned system should be easy to solve.
- The preconditioner should be cheap to construct and apply.

The first property means that the preconditioned iteration should converge rapidly, while the second ensures that each iteration is not too expensive. Notice that these two requirements are in competition with each other. It is necessary to strike a balance between the two needs. With a good preconditioner, the computing time for the preconditioned iteration should be significantly less than that for the unpreconditioned one.

What constitutes an acceptable cost of constructing the preconditioner, or setup time, will typically depend on whether the preconditioner can be reused or not. In the common situation where a sequence of linear systems with the same coefficient matrix (or a slowly varying one) and different right-hand sides has to be solved, it may pay to spend some time computing a powerful preconditioner, since its cost can be amortized over repeated solves. This is often the case when solving evolution problems by implicit methods, or mildly nonlinear problems by some variant of Newton's method.

Generally speaking, there are two approaches to constructing preconditioners. One popular approach in applications involving PDEs is to design specialized algorithms that are optimal (or nearly so) for a narrow class of problems. This application-specific approach can be very successful, but it may require complete knowledge of the problem at hand, including the original (continuous) equations, the domain of integration, the boundary conditions, details of the discretization, and so forth. By making use of available information about the analysis, geometry, and physics of the problem, very effective preconditioners can be developed. These include preconditioners based on "nearby" PDEs which are easier to solve than the given one, preconditioners based on lower order discretizations, and so on. As pointed out in [160], the method of diffusion synthetic acceleration (DSA) [6, 200, 214], which is widely used in the transport community, can be regarded as a physics-based preconditioner for the transport equation. Also, multigrid preconditioners are often of this kind (see [222] for a recent example).

The problem-specific approach may not always be feasible or even desirable. For a variety of reasons, the solver developer may not have complete knowledge of the problem to be solved, or the information might be too difficult to obtain or use. Furthermore, problem-specific approaches are generally very sensitive to the details of the problem, and even modest changes in the problem can compromise the effectiveness of the solver. For these reasons, there is a need for preconditioning techniques that are universally applicable. This justifies the enduring interest in general-purpose, purely algebraic methods (and related software) that use only information contained in the coefficient matrix A. These techniques, while not optimal for any particular problem, achieve reasonable efficiency on a wide range of problems. Algebraic methods are often easier to develop and to use and are particularly well suited for irregular problems such as may arise from discretizations involving unstructured meshes or from mesh-free applications. Also, codes using general-purpose solvers can be more easily adapted and retargeted as the underlying application changes. Furthemore, algebraic methods can often be fine-tuned to exploit specific features of a problem. The resulting "gray-box" solvers can be competitive with optimal, less versatile problem-specific techniques. Finally, algebraic methods are often used as basic components (building blocks) in more sophisticated, problem-specific solvers (see, e.g., the approach in [287]).

The purpose of this article is to provide computational scientists with an update of certain recent algorithmic developments in the field of preconditioning for large sparse systems of linear equations. Attention is restricted almost exclusively to general-purpose, algebraic techniques that can be applied to general sparse matrices. The main themes are incomplete factorization techniques and sparse approximate inverses. The important, emerging class of algebraic multilevel methods will be discussed only very briefly; this is due in part to lack of space, and also because there exists an up-to-date discussion of algebraic multigrid (AMG) methods in Stüben's chapter in [273]. In the present paper, special attention is given to practical issues, such as robustness and efficiency of implementation on modern computer architectures; theoretical aspects are for the most part not discussed.

2. PRECONDITIONED ITERATIVE METHODS: SOME HISTORY

This section describes, in broad outline, the main developments in the area of iterative methods and preconditioning techniques from a historical perspective. For more detailed information, the interested reader should consult Golub and O'Leary [155] (for developments up to 1976), the papers by Hestenes and by Young in [224] (for early work on conjugate gradients and SOR, respectively), and the recent overview by Saad and van der Vorst [255].

2.1. *Iterative Methods*

The first instances of iterative methods for solving systems of linear equations appeared in works of Gauss, Jacobi, Seidel, and Nekrasov during the 19th century (see [173]). Important developments took place in the first half of the 20th century, but the systematic study of iterative methods for large linear systems began only after the development of digital electronic computers, toward the end of World War II. In this sense, anything that took place in the period that goes from Gauss to the late 1940s belongs to the *prehistory* of our subject.

It is possible to look at the history of iterative methods as being divided into two main periods. The first period lasts roughly from 1950 to the early 1970s and is dominated by stationary iterative methods; the second period begins in the mid-1970s and is dominated by Krylov subspace methods. The development of multigrid methods also belongs to this second period.

The beginning of the first period is symbolized by David Young's famous Harvard thesis [293]. At this time, stationary iterative methods, such as successive overrelaxation (SOR) and its variants, were perfected and widely applied to the solution of large linear systems arising from the discretization of PDEs of elliptic type. Recall that a stationary iteration is any iterative scheme of the form

$$x^{k+1} = Tx^k + c, \quad k = 0, 1, \ldots, \tag{5}$$

where T is the (fixed) iteration matrix, c is a fixed vector, and x^0 an initial guess. Early on, Chebyschev acceleration of symmetrizable iterations (like SSOR) was considered [261]. The alternating direction implicit (ADI) method [117, 237], a serious competitor to the SOR method (especially in the petroleum industry), also belongs to this period. A crucial event is the publication of Varga's famous book [282] which, among other things, provided an elegant analysis of stationary iterative methods based on the Perron–Frobenius theory of matrices with nonnegative elements. This pioneering phase can be considered concluded with the publication of David Young's book [294], which contains the definitive treatment of SOR, ADI, and other classical iteration methods.

In spite of their mathematical elegance, stationary iterations suffer from serious limitations, such as lack of sufficient generality and dependence on convergence parameters that might be difficult to estimate without *a priori* information, for example on the spectrum of the coefficient matrix. For many problems of practical interest, stationary iterations diverge or converge very slowly.

Much work has been done to overcome these limitations. Adaptive parameter estimation procedures, together with acceleration techniques based on several emerging Krylov subspace methods, are covered in the monograph by Hageman and Young [168]. In a sense, this book marks the transition from the old to the new era in the history of iterative methods.

In the past few years, a number of books entirely devoted to iterative methods for linear systems have appeared [12, 26, 63, 71, 142, 160, 167, 217, 251, 289]. This is the culmination of several years of intensive development of iterative methods, particularly Krylov subspace methods. The very existence of these monographs is a testimony to the vitality, and also the maturity, of this field.

The history of Krylov subspace methods can be briefly summed up as follows (see [155, 255] for more detailed historical information). In 1952, Lanczos [199] and Hestenes and Stiefel [171] discovered (independently and almost simultaneously) the method of conjugate gradients (CG) for solving linear systems with a symmetric and positive definite matrix A. This method was initially regarded as an exact (direct) solution method, guaranteed to converge in no more than n steps, where n is the dimension of the problem. More precisely, in exact arithmetic the method is guaranteed to compute the solution to $Ax = b$ in exactly m steps, where m is the degree of the minimal polynomial of A, i.e., the number of distinct eigenvalues of A (since A is assumed to be symmetric). Based on numerical experiments it was observed that in the presence of rounding errors, a few more than n iterations were often necessary to obtain an accurate solution, particularly on ill-conditioned systems. On the other hand, it was observed in [171] that on well-conditioned systems, satisfactory approximations could be obtained in far fewer than n iterations, making the conjugate gradient method an attractive alternative to Gaussian elimination. In spite of this, for a number of reasons (see [155]) the conjugate gradient method saw relatively little use for almost 20 years, during which Gaussian elimination (for dense matrices) and SOR and Chebyschev iteration (for sparse matrices) were usually the preferred solution methods.

This situation finally changed around 1970 with the publication of a paper of Reid [241]. In this paper it was shown that for large sparse matrices that are reasonably well conditioned, the conjugate gradient method is a very powerful iterative scheme that yields good approximate solutions in far fewer than n steps. (The same observation had been made already in [133], but for some reason it had gone largely unnoticed.) Reid's paper set the stage for the rehabilitation of the conjugate gradient method and stimulated research in various directions. On one hand, extensions to linear systems with indefinite and/or nonsymmetric matrices were sought; on the other, techniques to improve the conditioning of linear systems were developed in order to improve the rate of convergence of the method [101].

The extension of conjugate gradients to symmetric indefinite systems led to the development, by Paige and Saunders, of the MINRES and SYMMLQ methods (see [236]). Important contributions were also made by Fletcher (see [143]). In the late 1970s and early 1980s several researchers turned their attention to the nonsymmetric case. This work, which continued throughout the 1980s, culminated with the development of the GMRES algorithm by Saad and Schultz [253], the QMR algorithm by Freund and Nachtigal [148], and the Bi-CGSTAB method by van der Vorst [280]. For an excellent discussion of developments up to about 1990, see [147]. In the 1990s, research on Krylov subspace methods proceeded at a more modest pace, mostly focusing on the analysis and refinement of existing techniques. An important development was an increased understanding of the finite-precision behavior of several basic Krylov subspace methods (see, e.g., [118, 162]). Also worth mentioning is the development of flexible variants of Krylov methods, which allow for the use of variable preconditioners [157, 230, 247, 269].

Another crucial event in the area of iterative methods is the development of multigrid methods by Brandt [61], Hackbusch [166], and others (see also the early papers by Fedorenko [138, 139]). Up-to-date treatments can be found in [59, 68, 273]. These methods can be interpreted as (very sophisticated) stationary iterative schemes, which can be used alone or in connection with CG or other Krylov subspace acceleration. The original (geometric) multigrid algorithms were problem-specific methods, primarily targeted at solving scalar, second-order elliptic PDEs on structured grids in simple geometries. With time, the scope of multigrid and, more generally, multilevel methods has grown to include larger and larger classes of problems, discretizations, geometries, and so forth. Notable examples of this trend are Dendy's black box multigrid [110] and especially Ruge and Stüben's algebraic multigrid (AMG) method [244] (see also the early papers [62, 159, 266] and the recent survey [267]). While not completely general purpose, AMG is widely applicable and it is currently the focus of intensive development (see, e.g., [97]). AMG is a promising technique for the solution of very large linear systems arising from the discretization of elliptic PDEs on unstructured grids, where it has been shown to be *algorithmically scalable* in many cases. This means that AMG achieves mesh-independent convergence rates and $\mathcal{O}(n)$ complexity on a wide range of problems. In spite of this, AMG is still not as widely used as other, nonscalable algorithms. This is due in part to robustness issues, and in part to the fact that AMG is more difficult to implement and to parallelize than other solvers. Furthermore, AMG sometimes incurs high setup and storage costs which might be difficult to amortize except for very large problems and computing resources.

An interesting consequence of the current interest in AMG is the development, in recent years, of a number of multilevel (AMG-like) variants of incomplete factorization and sparse approximate inverse preconditioners. These new algorithms attempt to achieve a compromise between the generality of purely algebraic techniques and the optimality of special-purpose methods like multigrid. This trend will be briefly discussed in Section 6.

Another major class of iterative schemes that should be mentioned here is that of domain decomposition methods, which became popular in the 1980s, in part as a result of the emergence of parallel computing (see [240, 264] for recent surveys). Currently, domain decomposition methods (and their multilevel variants) are used almost exclusively as preconditioners. Mathematically, they can be regarded as an extension of simple block preconditioners such as block Jacobi and block Gauss–Seidel. More than a specific class of algorithms, domain decomposition can be regarded as a paradigm to decompose a problem into smaller ones for the purpose of parallel processing.

2.2. *Preconditioners*

With the realization that preconditioning is essential for the successful use of iterative methods, research on preconditioners has moved to center stage in recent years. Of course, preconditioning as a way of transforming a difficult problem into one that is easier to solve has a long history. The term *preconditioning* appears to have been used for the first time in 1948 by Turing [274], in a paper on the effect of round-off errors on direct solution methods. The first use of the term in connection with iterative methods is found in a paper by Evans on Chebyschev acceleration of SSOR [134], in 1968.

However, the *concept* of preconditioning as a way of improving the convergence of iterative methods is much older. As far back as 1845 Jacobi [180] used plane rotations to achieve diagonal dominance in systems of normal equations arising from least-squares problems, thus ensuring the convergence of the simple iterative scheme that became known as Jacobi's method. Preconditioning as a means of reducing the condition number in order to improve convergence of an iterative process seems to have been first considered by Cesari in 1937 (see [77] and the account in [50]). Cesari's idea was to use a low-degree polynomial $p(A)$ in A as a preconditioner for a Richardson-type iteration applied to the preconditioned system $p(A)Ax = p(A)b$. For A symmetric and positive definite, he showed how to determine the coefficients of the polynomial so that the condition number of the preconditioned matrix $p(A)A$ was as small as possible. Polynomial preconditioners for Krylov subspace methods came into vogue in the late 1970s with the advent of vector computers [7, 119, 181] but they are currently out of favor because of their limited effectiveness and robustness, especially for nonsymmetric problems (see [10, 246] for a discussion).

Early ideas for preconditioning the conjugate gradient method can be found in [133], and occasionally in several papers published during the 1960s [155]. A detailed study of SSOR preconditioning as a means of accelerating the convergence of the CG method for some model problems was carried out in 1972 by Axelsson [9].

A major breakthrough took place around the mid-1970s, with the introduction by Meijerink and van der Vorst of the incomplete Cholesky-conjugate gradient (ICCG) algorithm [215]. Incomplete factorization methods were introduced for the first time by Buleev in the then-Soviet Union in the late 1950s, and independently by Varga (see [72, 73, 179, 281]; see also [231]). However, Meijerink and van der Vorst deserve credit for recognizing the potential of incomplete factorizations as preconditioners for the conjugate gradient method. Another paper that did much to popularize this approach is that by Kershaw [190]. Since then, a number of improvements and extensions have been made, including level-of-fills and drop tolerance-based incomplete factorizations, generalizations to block matrices, modified and stabilized variants, and even (most recently) efficient parallel implementations. Incomplete factorization preconditioners are reviewed in the next section, with focus on recent developments (see [81] for a survey reflecting the state of the art in the mid-1990s).

Notwithstanding their popularity, incomplete factorization methods have their limitations. These include potential instabilities, difficulty of parallelization, and lack of algorithmic scalability. The first two of these limitations have motivated considerable interest, during the past decade, in an alternative class of algebraic preconditioning techniques called *sparse approximate inverses*. These methods were first introduced in the early 1970s [28, 31], but it is only in the past few years that efficient algorithms and implementations have become available. Some of these techniques will be reviewed in Section 5. The lack of scalability, on the other hand, has led to the development of a number of variants based on the multilevel paradigm (AMG-like) (see Section 6). This brings us to the cutting edge of current research on preconditioners.

3. INCOMPLETE FACTORIZATION METHODS

When a sparse matrix is factored by Gaussian elimination, fill-in usually takes place. This means that the triangular factors L and U of the coefficient matrix A are considerably less sparse than A. Even though sparsity-preserving pivoting techniques can be used to reduce fill-in, sparse direct methods are not considered viable for solving very large linear systems such as those arising from the discretization of three-dimensional boundary value problems, due to time and space constraints.

However, by discarding part of the fill-in in the course of the factorization process, simple but powerful preconditioners can be obtained in the form $M = \bar{L}\bar{U}$, where $\bar{L}$ and $\bar{U}$ are the incomplete (approximate) LU factors.

After a brief review of the basic techniques, this section and the next describe recent developments concerning robustness, orderings, and block and parallel variants of incomplete factorization methods. Implementation details are not given, and the interested reader is referred to the original references.

3.1. *Overview of Algorithms*

Incomplete factorization algorithms differ in the rules that govern the dropping of fill-in in the incomplete factors. Fill-in can be discarded based on several different criteria, such as position, value, or a combination of the two.

Letting $\mathbf{n} = \{1, 2, \ldots, n\}$, one can fix a subset $\mathcal{S} \subseteq \mathbf{n} \times \mathbf{n}$ of positions in the matrix, usually including the main diagonal and all (i, j) such that $a_{ij} \neq 0$, and allow fill-in in the LU factors only in positions which are in $\mathcal{S}$. Formally, an incomplete factorization step can be described as

$$a_{ij} \leftarrow \begin{cases} a_{ij} - a_{ik}a_{kk}^{-1}a_{kj}, & \text{if } (i, j) \in \mathcal{S}, \\ a_{ij}, & \text{otherwise,} \end{cases} \tag{6}$$

for each k and for $i, j > k$. Notice that the incomplete factorization may fail due to division by zero (this is usually referred to as a *breakdown*), even if A admits an LU factorization without pivoting. This and related issues are discussed in the next subsection; here, we assume that the incomplete factorization can be carried out without breakdowns.

If $\mathcal{S}$ coincides with the set of positions which are nonzero in A, we obtain the *no-fill* ILU *factorization*, or ILU(0). For SPD matrices the same concept applies to the Cholesky factorization $A = LL^T$, resulting in the no-fill IC factorization, or IC(0). Remarkably, for certain matrices with special structure (e.g., five-point matrices), CG with no-fill incomplete Cholesky preconditioning can be implemented at nearly the same cost per iteration as CG with no preconditioning [130].

The no-fill ILU and IC preconditioners are very simple to implement, their computation is inexpensive, and they are quite effective for significant problems, such as low-order discretizations of scalar elliptic PDEs leading to M-matrices[1] and diagonally dominant matrices. Indeed, these are the types of problems for which these preconditioners were originally proposed (see [215]). However, for more difficult and realistic problems the no-fill factorizations result in too crude an approximation of A, and more sophisticated

[1] Recall that a nonsingular M-matrix $A = [a_{ij}]$ is a matrix for which $a_{ij} \neq 0$ for $i \neq j$ and A^{-1} has all nonnegative entries.

preconditioners, which allow some fill-in in the incomplete factors, are needed. For instance, this is the case for highly nonsymmetric and indefinite matrices, such as those arising in many CFD applications.

A hierarchy of ILU preconditioners may be obtained based on the "levels of fill-in" concept, as formalized by Gustafsson [164] for finite difference discretizations and by Watts [288] for more general problems. A level of fill is attributed to each matrix entry that occurs in the incomplete factorization process. Fill-ins are dropped based on the value of the level of fill. The formal definition (as given in [251]) is as follows. The initial level of fill of a matrix entry a_{ij} is defined as

$$lev_{ij} = \begin{cases} 0, & \text{if } a_{ij} \neq 0 \quad \text{or} \quad i = j, \\ \infty, & \text{otherwise.} \end{cases}$$

Each time this element is modified by the ILU process, its level of fill must be updated according to

$$lev_{ij} = \min\{lev_{ij}, lev_{ik} + lev_{kj} + 1\}.$$

Let ℓ be a nonnegative integer. With ILU(ℓ), all fill-ins whose level is greater than ℓ are dropped. Note that for $\ell = 0$, we recover the no-fill ILU(0) preconditioner. The motivation behind this approach is that for diagonally dominant matrices, the higher the level of fill of an element, the smaller the element tends to be in absolute value.

In many cases, ILU(1) is already a considerable improvement over ILU(0), and the cost of forming and using the preconditioner is still acceptable. It rarely pays to consider high values of ℓ, except perhaps for very difficult problems, due to the rapidly increasing cost of computing and using the preconditioner for increasing ℓ.

The level-of-fill approach may not be robust enough for certain classes of problems. For matrices which are far from being diagonally dominant, ILU(ℓ) may require storing many fill-ins that are small in absolute value and, therefore, contribute little to the quality of the preconditioner, while making it expensive to compute and use. In many cases, an efficient preconditioner can be obtained from an incomplete factorization where new fill-ins are accepted or discarded on the basis of their size. In this way, only fill-ins that contribute significantly to the quality of the preconditioner are stored and used.

A *drop tolerance* is a positive number τ which is used in a dropping criterion. An absolute dropping strategy can be used, whereby new fill-ins are accepted only if greater than τ in absolute value. This criterion may work poorly if the matrix is badly scaled, in which case it is better to use a relative drop tolerance. For example, when eliminating row i, a new fill-in is accepted only if it is greater in absolute value than $\tau \|a_i\|_2$, where a_i denotes the ith row of A. Other criteria are also in use.

A drawback of this approach is that it is difficult to choose a good value of the drop tolerance: usually, this is done by trial-and-error for a few sample matrices from a given application, until a satisfactory value of τ is found. In many cases, good results are obtained for values of τ in the range 10^{-4}–10^{-2}, but the optimal value is strongly problem dependent.

Another difficulty is that it is impossible to predict the amount of storage that will be needed to store the incomplete LU factors. An efficient, predictable algorithm is obtained by limiting the number of nonzeros allowed in each row of the triangular factors. Saad [248] has proposed the following *dual threshold* strategy: fix a drop tolerance τ and a number

TABLE I
Test Results for 2D Convection–Diffusion Problem

Precond.	ILU(0)	ILU(1)	ILUT(5×10^{-3}, 5)	ILUT(10^{-3}, 10)
Its	159	90	66	37
Density	1	1.4	2.2	3.6
Time	87	59	53	41

p of fill-ins to be allowed in each row of the incomplete L/U factors; at each step of the elimination process, drop all fill-ins that are smaller than τ times the 2-norm of the current row; of all the remaining ones, keep (at most) the p largest ones in magnitude. A variant of this approach allows in each row of the incomplete factors p nonzeros in addition to the positions that were already nonzeros in the original matrix A. This makes sense for irregular problems in which the nonzeros in A are not distributed uniformly.

The resulting preconditioner, denoted ILUT(τ, p), is quite powerful. If it fails on a problem for a given choice of the parameters τ and p, it will often succeed by taking a smaller value of τ and/or a larger value of p. The corresponding incomplete Cholesky preconditioner for SPD matrices, denoted ICT, can also be defined.

As an illustration, we report in Table I some results for a model convection–diffusion equation with homogeneous Dirichlet boundary conditions on the unit square, discretized by centered finite differences on a 400×400 grid. The Krylov subspace method used was Bi-CGSTAB. The initial guess was the zero vector, and the iteration was stopped when a reduction of the initial residual by five orders of magnitude had been achieved. For each preconditioner we report the number of iterations (Its), the density of the preconditioner (defined as the ratio of the number of nonzeros in the incomplete LU factors divided by the number of nonzeros in the coefficient matrix A), and the total solution time (measured in seconds). The runs were performed on one processor of an SGI Origin 2000. Note that going from ILU(0) to ILUT(10^{-3}, 10) resulted in a reduction by a factor of two in solution time, but at the expense of a significant increase in storage demand. It should be stressed, however, that this is a fairly easy model problem; for more difficult linear systems, ILUT can be expected to outperform ILU(0) much more dramatically than it does here.

In this example, adding fill-in improved the overall performance of the solver. However, there are also cases where allowing more fill-in makes the preconditioner worse. Indeed, especially for indefinite matrices, the quality of incomplete factorization preconditioners as measured by the corresponding convergence rate of the preconditioned iteration may not improve monotonically with the amount of fill allowed in the incomplete factors, and it can happen that a preconditioner gets worse before it gets better (see the discussion in [116, pp. 198–199]). Furthermore, a denser preconditioner may require fewer iterations to converge but more CPU time, due to the increased costs. Drop tolerance-based preconditioners may be difficult to use in a black-box fashion, at least until the user has accumulated some experience with their use on specific problems. This motivated the development by Jones and Plassmann [183, 184] of a variant of incomplete factorization preconditioning that does not require the selection of a drop tolerance and has predictable storage requirements. In this approach, a fixed number n_k of nonzeros is allowed in the kth row (or column) of the incomplete Cholesky factor $\bar{L}$; n_k is simply the number of nonzeros that were originally in the kth row of the lower triangular part of A. However, the original sparsity pattern is ignored, and only the nonzeros with the largest magnitude are kept. In this way a preconditioner is obtained that has the same storage requirements as the no-fill incomplete Cholesky preconditioner but better convergence properties in many cases. This strategy has the advantage of being a black-box method, but convergence can be slow on difficult problems. A natural extension, investigated in [204], is to allow additional fill in the factor $\bar{L}$. For a certain integer $p \geq 0$, the number of nonzeros that are retained in the kth row (or column) of $\bar{L}$ is $n_k + p$. This strategy, which reduces to that of Jones and Plassmann for $p = 0$, can improve performance dramatically at the expense of somewhat higher—but still predictable—storage requirements. The preconditioner is no longer entirely a black box (since p must be specified), but only one parameter must be supplied compared with two for Saad's ILUT. Note that exactly the same preconditioner can be obtained from ILUT by setting the drop tolerance to 0; however, the implementation is simplified if no drop tolerance test is needed. The optimal value of p is of course problem dependent, but a value of $p \approx 5$ will do in many cases [204].

For many second-order, self-adjoint, scalar elliptic PDEs discretized with finite differences or finite elements it can be shown that the spectral condition number of the discrete operator grows as $\mathcal{O}(h^{-2})$ for $h \to 0$, where h denotes the size of the mesh (or some average measure of it for irregular problems) and the number of CG iterations to achieve a prescribed reduction in some norm of the error is proportional to h^{-1}. When no fill-ins (or small levels of fill) are allowed in the incomplete factors, the condition number of the preconditioned operator still behaves as $\mathcal{O}(h^{-2})$ (only with a much smaller multiplicative constant) and the number of iterations still scales as h^{-1}, as in the absence of preconditioning. In some cases, the situation can be improved by means of a technique called *modified* incomplete Cholesky (MIC) factorization. In its simplest form, the idea is to force the preconditioner A to have the same row sums as the original matrix A; that is, $Me = Ae$, where e denotes a column vector with entries all equal to 1. This can be accomplished by adding the dropped fill-ins to the diagonal entries (*pivots*) in the course of the elimination. For certain classes of matrices, including diagonally dominant ones, it can be shown that the pivots remain strictly positive (that is, no breakdown can occur) and that the condition number of the preconditioned matrix behaves as $\mathcal{O}(h^{-1})$. Thus, the estimated number of preconditioned CG iterations becomes $\mathcal{O}(h^{-1/2})$, a significant improvement over the unmodified method. Modified incomplete Cholesky (or MILU) methods were considered in special cases by Buleev (cf. [179]), by Dupont *et al.* [126], and by Axelsson [9] and for more general problem classes by Gustafsson [164]. In some cases (e.g., if a red–black ordering of the grid points is adopted) it is necessary to perturb the diagonal entries of A by a quantity of the order h^2 prior to performing the modified incomplete factorization in order to avoid breakdowns (see [129]). Perturbing the diagonal entries also allows one to control the size of the largest eigenvalue for some model problems (see [71, p. 80] for a discussion and references). Another variant is to *relax* the modification by multiplying the dropped fill by a relaxation factor between 0 and 1 prior to adding it to the diagonal entries (see, e.g., [217] for a discussion of relaxed incomplete Cholesky factorizations and additional references).

In spite of the large improvements that are possible for model problems through the idea of modification, modified incomplete factorizations are not as widely used as unmodified ones, possibly due to the fact that modified methods are more likely to break down on nonmodel problems. Also, even when they do not break down, modified methods tend to perform poorly on nonmodel problems, in part because of their higher sensitivity to rounding errors (see [279]). Nevertheless, the idea of forcing the preconditioner to agree with the coefficient matrix on some selected "trial" vector(s) is quite fruitful and is an important ingredient in the construction of effective preconditioning methods.

3.2. *Existence and Stability Issues*

When designing reliable preconditioners, the delicate issue of the existence and stability of the preconditioner must be addressed. This subsection is devoted to a discussion of such questions for incomplete factorization methods.

3.2.1. The symmetric positive definite case. Recall that a square matrix A is said to admit the LU factorization if it can be written as $A = LU$, where L is lower triangular and U is unit upper triangular (i.e., upper triangular with diagonal entries equal to 1). The LU factorization of a nonsingular matrix, when it exists, is unique. It is well known [156] that the LU factorization of a matrix A exists if and only if all leading principal submatrices of A are nonsingular, and that for any nonsingular matrix A there exists a permutation matrix P such that PA admits the LU factorization. Furthermore, A is SPD if and only if it can be factored as $A = LL^T$, with L lower triangular with positive diagonal elements (the Cholesky factorization). The situation is more complicated in the case of incomplete factorizations. As mentioned, incomplete factorization of a general matrix A can fail due to the occurrence of zero pivots, regardless of whether A admits the LU factorization or not. Moreover, trouble (in the form of numerical instabilities) can be expected in the presence of small pivots. As noted in [245], an incomplete factorization may fail due to zero pivots even if pivoting for stability is used. In the SPD case, a breakdown occurs each time a nonpositive pivot is encountered. The conjugate gradient method necessitates both A and the preconditioner $M = \bar{L}\bar{L}^T$ to be SPD, and a zero or negative pivot in the incomplete Cholesky process would result in a preconditioner which is not positive definite or not well defined.

The existence of an incomplete factorization (without pivoting) can be established for certain classes of matrices. In [215] the existence of the incomplete Cholesky factorization was proved, for arbitrary choices of the sparsity pattern $\mathcal{S}$, for the class of M-matrices. In this case, the pivots in the incomplete process are bounded below by the exact pivots, and since the exact pivots for an M-matrix are strictly positive [48], no breakdown is possible. This existence result was extended shortly thereafter to a somewhat larger class (that of H-matrices with positive diagonal entries) by several authors [213, 243, 283].

On the other hand, incomplete factorization can fail for a general SPD matrix. Not surprisingly, failures are frequent for matrices that are nonsymmetric and/or indefinite.

Unfortunately, many important applications lead to matrices that are not M- or even H-matrices. This lack of robustness of incomplete factorization is one of the main reasons iterative methods have not been widely used in industrial applications, where reliability is paramount.

Pivot breakdown has been recognized as a problem since the early days of incomplete factorization preconditioning, and a number of remedies have been proposed to deal with it, particularly in the SPD case. Significant progress for both the SPD and the general case has been made in recent years; some of these techniques are reviewed next.

In the SPD case, three general strategies are in use to guarantee the existence of an incomplete Cholesky factorization (ICF) of A. In increasing order of complexity, these are as follows

• Increase the diagonal dominance of A by diagonal shifts (either locally or globally, before or during the factorization).

• Replace A with a "nearby" M-matrix $\hat{A}$, compute an ICF of $\hat{A} \approx \bar{L}\bar{L}^T$, and use that as a preconditioner for $Ax = b$.

• Do not modify A but recast the factorization in a form that avoids breakdowns.

Some of these techniques are also applicable if A is nonsymmetric but positive definite (that is, $x^T Ax > 0$, if $x \neq 0$) or if A is indefinite with a few negative eigenvalues and an approximate factorization of A of the form $A \approx \bar{L}\bar{L}^T$ is sought.

The simplest fix is to replace a negative or small pivot at step i with some (positive) quantity p_i. This *dynamic* local diagonal shift strategy was suggested by Kershaw [190]. It was motivated by the hope that if only a few of the pivots are unstable (i.e., nonpositive), the resulting factorization might still yield a satisfactory preconditioner. The choice of a replacement value for a nonpositive pivot is no simple matter. If the p_i chosen is too large, the preconditioner will be inaccurate; i.e., $\|A - \bar{L}\bar{L}^T\|_F$ is large. Here $\|\cdot\|_F$ denotes the Frobenius matrix norm (the square root of the sum of the squares of the matrix entries). If the p_i chosen is too small, the triangular solves with L and L^T can become unstable; i.e., $\|I - \bar{L}^{-1}A\bar{L}^{-T}\|_F$ is large. Some heuristics can be found in [190, 277] and in the optimization literature [154, 260] (see also [204]).

Unfortunately, it is frequently the case that even a handful of pivot shifts will result in a poor preconditioner. The problem is that if a nonpositive pivot occurs, the loss of information about the true factors due to dropping has already been so great that no "local" trick can succeed in recovering a good preconditioner—it's too late.[2]

A somewhat better strategy was suggested by Manteuffel in [213]. If an incomplete factorization of A fails due to pivot breakdown, a *global* diagonal shift is applied to A prior to reattempting the incomplete factorization. That is, the incomplete factorization is carried out on $\hat{A} = A + \alpha\,\mathrm{diag}(A)$, where $\alpha > 0$ and $\mathrm{diag}(A)$ denotes the diagonal part of A. If this fails, α is increased and the process is repeated until an incomplete factorization is successfully computed.

Clearly, there exists a value α^* for which the incomplete factorization of $\hat{A}$ is guaranteed to exist: for instance, one can take the smallest α that makes $\hat{A} = A + \alpha\,\mathrm{diag}(A)$ diagonally dominant. Because diagonally dominant matrices are H-matrices, the incomplete factorization of $\hat{A}$ exists. However, the best results are usually obtained for values of α that are much smaller than α^* and larger than the smallest α for which $\hat{A}$ admits an incomplete factorization. The obvious disadvantage of this approach is the difficulty in picking a "good" value of α. Because α is chosen on the basis of a trial-and-error strategy, this approach can be expensive.

A popular technique in the structural engineering community is the robust incomplete Cholesky factorization developed by Ajiz and Jennings [2]. This method is based on the idea of *stabilized cancellation*. Whenever an element l_{ij} of the incomplete Cholesky factor L is dropped, $|l_{ij}|$ is added to the corresponding diagonal entries a_{ii} and a_{jj}. In some cases it is desirable to multiply $|l_{ij}|$ by a scaling factor before adding it to the diagonal, so as to ensure that the diagonal modification is proportional to the size of the diagonal entry being modified (see [2, 172]). In this way, the incomplete Cholesky factor L is the *exact* Cholesky factor of a perturbed matrix $\hat{A} = A + C$, where C, the *cancellation matrix*, is positive semidefinite. Thus, *no breakdown can occur*. Moreover, the splitting

[2] This point is convincingly argued by Lin and Moré [204].

TABLE II
Comparison of Standard and Ajiz–Jennings ICF on Shell Problem

Precond.	Density	Its	P-time	It-time
ICT(10^{-3}, 30)	1.90	43	1.11	0.65
AJ(10^{-3})	1.83	125	0.77	1.82

$A = \bar{L}\bar{L}^T - C = \hat{A} - C$ is convergent, in the sense that $\rho(I - \hat{A}^{-1}A) < 1$. Here, $\rho(\cdot)$ denotes the spectral radius of a matrix. Thus, the triangular solves with the incomplete factor $\bar{L}$ and its transpose cannot become unstable.

Although reliable, this strategy may result in preconditioners of low quality, generally inferior to a standard incomplete Cholesky factorization with comparable density whenever the latter exists without breakdowns. Indeed, if many fill-ins are dropped from the factors, large diagonal modifications are performed, reducing the accuracy of the preconditioner. The results in Table II illustrate this fact for a linear system arising from the finite element analysis of shells. This is matrix S2RMQ4M1 from the CYLSHELL collection in the Matrix Market [225]. This problem corresponds to a shell of characteristic thickness $t/R = 10^{-2}$ discretized with a 30×30 uniform quadrilateral mesh. The stiffness matrix A has order $n = 5,489$ and the lower triangular part of A contains $nz = 143,300$ nonzero entries. The matrix has a condition number $\kappa(A) \approx 1.2 \times 10^8$ (see [35] for further details). We compare a dual drop tolerance-based incomplete Cholesky factorization (denoted ICT in the table) with the Ajiz–Jennings robust incomplete Cholesky preconditioner (denoted AJ). Prior to computing the preconditioner, the coefficient matrix is symmetrically scaled by its diagonal and a reverse Cuthill–McKee ordering [150] is applied. Using a tolerance $\tau = 10^{-3}$ and $p = 30$ in ICT we obtain preconditioners with similar density, but the quality of the Ajiz–Jennings preconditioner is clearly inferior. We note that for this problem, simple diagonal preconditioning takes 1609 iterations and 8.56 seconds to converge.

In the Ajiz–Jennings approach, the modification of diagonal entries is done *dynamically*, as dropping occurs in the course of the incomplete factorization process. Another approach, known as *diagonally compensated reduction* of positive off-diagonal entries, consists of modifying the matrix A *before* computing an incomplete factorization. In the simplest variant of this method [15], positive off-diagonal entries are set to zero and added to the corresponding diagonal entries. The resulting matrix satisfies $\hat{A} = A + C$, with C positive semidefinite, and has nonpositive off-diagonal entries. Hence, it is a Stieltjes matrix (symmetric M-matrix) and an incomplete factorization of $\hat{A}$ can be computed without breakdowns; the resulting preconditioner $M = \bar{L}\bar{L}^T$ is then applied to the original linear system $Ax = b$.

Preconditioners based on diagonally compensated reduction give good results on moderately ill-conditioned problems, such as certain solid elasticity problems [258] and diffusion problems posed on highly distorted meshes [32], but are ineffective for more difficult problems arising from the finite element analysis of thin shells and other structures [32, 172, 259].

A more sophisticated approach has been proposed by Tismenetsky [271], with some improvements and theory contributed by Kaporin [186] (see also [268]). This method was originally described in terms of rank-one updates to the active submatrix. However, to show that the method cannot suffer pivot breakdowns, it is preferable to interpret it as a bordering scheme. At step $k+1$ the exact decomposition is

$$\begin{bmatrix} A_k & v_k \\ v_k^T & \alpha_{k+1} \end{bmatrix} = \begin{bmatrix} L_k & 0 \\ y_k^T & \delta_{k+1} \end{bmatrix} \begin{bmatrix} L_k^T & y_k \\ 0 & \delta_{k+1} \end{bmatrix},$$

where

$$A_k = L_k L_k^T, \quad y_k = L_k^{-1} v_k, \quad \delta_{k+1} = \sqrt{\alpha_{k+1} - y_k^T y_k}.$$

When dropping is applied to the y_k vector, an incomplete Cholesky factorization is obtained (a special case of Chow and Saad's ILUS preconditioner [95]). A breakdown occurs if $y_k^T y_k \geq \alpha_{k+1}$. It can be shown that in Tismenetsky's method the pivot δ_{k+1} is computed as

$$\delta_{k+1} = \sqrt{\alpha_{k+1} - 2y_k^T y_k + y_k^T L_k^{-1} A_k L_k^{-T} y_k}, \tag{7}$$

which reduces to $\delta_{k+1} = \sqrt{\alpha_{k+1} - y_k^T y_k}$ in the absence of dropping. However, breakdowns are now impossible even in the presence of dropping, since the expression under square root in (7) is

$$\begin{bmatrix} y_k^T L_k^{-1} & -1 \end{bmatrix} \begin{bmatrix} A_k & v_k \\ v_k^T & \alpha_{k+1} \end{bmatrix} \begin{bmatrix} L_k^{-T} y_k \\ -1 \end{bmatrix} > 0.$$

Once the pivot δ_{k+1} has been computed, a drop tolerance can be safely applied to remove unwanted fill-ins from y_k. This strategy is rather expensive in terms of preconditioner construction costs and incurs high intermediate storage requirements, but it produces high-quality preconditioners which outperform virtually all other known incomplete factorization techniques in terms of convergence rates (see the results in [186, 271]). Tismenetsky's idea, which unfortunately has attracted surprisingly little attention, shows that it is possible to develop breakdown-free incomplete factorizations of positive definite matrices without the need for any matrix modifications by a clever recasting of the underlying factorization process. The price to pay is represented by higher setup and intermediate storage costs. These can be reduced to acceptable levels by combining Tismenetsky's idea with the method of Ajiz and Jennings to preserve the breakdown-free property, which is accomplished by the use of two drop tolerances. Briefly, some of the (smaller) intermediate fill-ins that would have been retained are dropped, for example by applying a second drop tolerance test to the intermediate fill-ins. The larger this second tolerance, the closer the method becomes to the Ajiz–Jennings technique. Some form of diagonal compensation (in the style of Ajiz–Jennings) is then needed in order to avoid pivot breakdowns. Variants of this approach have been tested in [186, 268, 271]. In particular, [186] uses a second drop tolerance that is a simple function of the first one, so that the user only needs to specify one tolerance. Generally speaking, these methods display convergence rates which are comparable with those of the "full" Tismenetsky method while at the same time requiring much less storage and setup time. However, these techniques are still quite expensive in time and space. In [44], it is shown that the total amount of storage needed to compute the Tismenetsky–Kaporin preconditioner can be comparable to that needed in computing a *complete* factorization with a suitable fill-reducing ordering, such as minimum degree. This is a serious drawback of the method in applications involving large matrices.

In summary, it can be said that existing solutions to the robustness (existence) problem for incomplete factorization preconditioners for general SPD matrices tend to fall into one of two camps: simple and inexpensive fixes that result in low-quality preconditioners in terms of rates of convergence, or sophisticated, expensive strategies that yield high-quality preconditioners. The recent paper [44] introduces a new preconditioning strategy, called RIF (for robust incomplete factorization), that strikes a compromise between these two extremes. The basis for the RIF method is the observation that the so-called "root-free Cholesky factorization" $A = LDL^T$ can be computed by means of an A-orthogonalization process applied to the unit basis vectors $e_1, e_2, \dots, e_n$. This is simply the Gram–Schmidt process with respect to the inner product generated by the SPD matrix A. This algorithm was first described in [144]. The observation that A-orthogonalization of the unit basis vectors is closely related to (symmetric) Gaussian elimination can be found in the celebrated paper by Hestenes and Stiefel [171] on the conjugate gradient method (see pp. 425–427). Because this algorithm costs twice as much as the Cholesky factorization in the dense case, A-orthogonalization is never used to factor matrices. However, as noted in [38], A-orthogonalization also produces the inverse factorization $A^{-1} = ZD^{-1}Z^T$ (with Z unit upper triangular and D diagonal) and this fact has been exploited to construct factored sparse approximate inverse preconditioners (see [32, 38, 191] and the discussion in Section 5.1.2).

As shown in [44], A-orthogonalization can be used to compute an approximate factorization $A \approx \bar{L}\bar{D}\bar{L}^T$, with $\bar{L}$ unit lower triangular[3] and $\bar{D}$ diagonal and positive definite. The algorithm is inherently stable, and no pivot breakdown is possible. To describe this algorithm, we begin by recalling the A-orthogonalization process. Initially, $z_j = e_j (1 \le j \le n)$; here e_j denotes the n-vector having all entries equal to 0 except for the jth one, which is equal to 1. Then the (right-looking) algorithm consists of the nested loop

$$z_i \leftarrow z_i - \frac{\langle Az_j, z_i\rangle}{\langle Az_j, z_j\rangle} z_j, \tag{8}$$

where $j = 1, 2, \dots, n$ and $i = j + 1, \dots, n$. On completion, letting $Z = [z_1, z_2, \dots, z_n]$ and $D = \mathrm{diag}(d_1, d_2, \dots, d_n)$, with $d_j = \langle Az_j, z_j\rangle$, we obtain the inverse upper–lower factorization $A^{-1} = ZD^{-1}Z^T$. A left-looking variant of this scheme also exists.

To obtain a sparse preconditioner, a dropping rule is applied to the z_i vectors after each update step (see [32]). Of course, in order to have an efficient algorithm, the loop has to be carefully implemented by exploiting sparsity in A and in the z_i vectors. In particular, most of the inner products $\langle Az_j, z_i\rangle$ are structurally zero and they need not be computed, and the corresponding update (8) can be skipped. In this way, it is generally possible to compute a sparse approximate inverse preconditioner $A^{-1} \approx ZD^{-1}Z^T$ quite cheaply, typically with $\mathcal{O}(n)$ complexity [32]. Because the pivots d_j are computed as $d_j = \langle Az_j, z_j\rangle$, with $z_j \neq 0$ (since the jth entry of z_j is equal to 1), this preconditioner, which is usually referred to as SAINV, is well defined for a general SPD matrix. Variants of SAINV have been shown to be reliable in solving highly ill-conditioned linear systems [32, 36, 191].

Consider now the exact algorithm (with no dropping) and write $A = LDL^T$, with L unit lower triangular and D diagonal. Observe that L in the LDL^T factorization of A and the

[3]A *unit* triangular matrix is a triangular matrix with ones on the diagonal.

inverse factor Z satisfy

$$AZ = LD, \quad \text{or} \quad L = AZD^{-1},$$

where D is the diagonal matrix containing the pivots. This easily follows from

$$Z^T AZ = D \quad \text{and} \quad Z^T = L^{-1}.$$

Recall that $d_j = \langle Az_j, z_j \rangle$; by equating corresponding entries of AZD^{-1} and $L = [l_{ij}]$ we find that

$$l_{ij} = \frac{\langle Az_j, z_i \rangle}{\langle Az_j, z_j \rangle}, \quad i \geq j. \tag{9}$$

Hence, the L factor of A can be obtained as a by-product of the A-orthogonalization process, at *no extra cost*. We mention that the relationship (9), which can be found in [171], has also been independently noted in [55].

The algorithm requires some intermediate storage for the sparse z_i vectors while they are needed to compute the multipliers (9). At step j of the algorithm ($1 \leq j \leq n$) we need to store, besides the j computed columns of the incomplete factor L, the remaining $n - j$ columns $z_{j+1}, \ldots, z_n$ of Z. Notice that these sparse vectors can have nonzero entries only within the first j positions, besides the 1 in position k of z_k. As j increases, there are fewer and fewer such vectors that need to be kept in temporary storage, but they tend to fill in. Assuming a uniform distribution of nonzeros in both L and Z, a back-of-the-envelope calculation shows that in terms of storage, the "high-water mark" is reached for $j \approx n/2$, that is, midway through the A-orthogonalization process, after which it begins to decrease. The total storage can be estimated to be approximately 25% more than the storage required by the final incomplete factor L. In practice, we found that this is often an overestimate, unless a very small drop tolerance is used (which is not practical anyway). This is due to the fact that the bandwidth of A and Z can be reduced by suitable permutations, and therefore the nonzeros are not uniformly distributed. Hence, with a careful implementation and using suitable dynamic data structures, the proposed algorithm incurs negligible intermediate storage requirements.

In principle, the incomplete factorization based on A-orthogonalization needs two drop tolerances; one for the incomplete A-orthogonalization process, to be applied to the z vectors, and a second one to be applied to the entries of L. Note that the latter is simply postfiltration: once a column of L has been computed, it does not enter the computation of the remaining ones. In practice we found that this postfiltration is usually not necessary, and indeed it can be harmful to the quality of the preconditioner (see [44]). Therefore, postfiltration in L is not recommended, and dropping is applied only to the z vectors.

In Table III we report results for different incomplete factorization preconditioners applied to solving a linear system with coefficient matrix BCSSTK25 from [122]. This matrix, which arises in the finite element analysis of a skyscraper, has order $n = 15{,}439$ and its lower triangular part contains $nnz = 133{,}840$ nonzeros. In the table we present results for diagonally preconditioned CG (denoted "JCG"), the Ajiz–Jennings robust incomplete Cholesky method (denoted "AJ"), the Tismenetsky–Kaporin method of [186] (denoted "TK"), and the RIF method just described. On this problem, standard incomplete Cholesky factorization methods break down for almost all reasonable choices of the parameters controlling fill-in.

TABLE III
Comparison of Robust Incomplete Factorizations on Problem BCSSTK25

Precond.	Density	Its	P-time	It-time
JCG	—	9296	—	65.0
AJ	1.39	897	0.26	13.6
TK	1.28	527	1.73	7.67
RIF	1.37	588	1.22	8.92

The results in Table III are fairly typical (see [44] for extensive numerical testing). What the table does not show is that the best method in terms of CPU time, the Tismenetsky–Kaporin algorithm, needs large amounts of intermediate storage to compute the incomplete factorization. For this problem, the storage needed is about 70% of the space needed to store the *complete* Cholesky factor of the matrix (reordered with minimum degree). In contrast, the Ajiz–Jennings and the RIF preconditioner need only about 13% of the storage needed for the complete factor.

Concluding, we can say that there exist now several solutions to the problem of devising reliable incomplete factorization preconditioners for general SPD matrices. While these methods do not exhibit optimal complexity, in the sense that the condition number of the preconditioned system still grows as the discretization parameter h tends to zero, robust incomplete factorizations are remarkably versatile, general, and easy to use. Moreover, they can be incorporated into more sophisticated preconditioners, including multilevel methods, which are often closer to being optimal.

3.2.2. The general case. The state of the art is less satisfactory when A is not SPD. This fact should not come as a surprise, for in this case even the existence of a stable *complete* factorization without pivoting is not guaranteed. Even if the incomplete factorization process can be completed without breakdowns, there is no guarantee in general that it will result in a useful preconditioner. Especially for nonsymmetric problems, the incomplete factors $\bar{L}$ and $\bar{U}$ may happen to be very ill conditioned, even for a well-conditioned A. Application of the preconditioner, which involves a forward substitution for $\bar{L}$ and a backsubstitution for $\bar{U}$, may introduce large errors that can destroy the effectiveness of the preconditioner. This phenomenon was noticed by van der Vorst [277], analyzed for a class of discrete convection–diffusion equations by Elman [131], and investigated empirically on a large number of general sparse matrices by Chow and Saad [94]. In a nutshell, instability in the ILU factors is related to lack of diagonal dominance. The two types of instability (zero or small pivots and unstable triangular solves) can occur independently of one another, although they are frequently observed together in matrices that are strongly nonsymmetric and far from diagonally dominant. Matrices of this type arise frequently in CFD, for instance in the numerical solution of the Navier–Stokes equations.

The quality of an incomplete factorization $A \approx \bar{L}\bar{U}$ can be gauged, at least in principle, by its *accuracy* and *stability*. Accuracy refers to how close the incomplete factors of A are to the exact ones and can be measured by $N_1 = \|A - \bar{L}\bar{U}\|_F$. For some classes of problems, including symmetric M-matrices, it can be shown [14, 125] that the number of preconditioned CG iterations is almost directly related to N_1, so that improving the accuracy of the incomplete factorization by allowing additional fill will result in a decrease of the number of iterations to converge. However, for more general problems (and especially for highly nonsymmetric and indefinite matrices) this is not necessarily the case, and N_1 alone may be a poor indicator of the quality of the preconditioner.

Stability refers to how close the preconditioned matrix $(\bar{L}\bar{U})^{-1}A$ is to the identity matrix I and is measured by $N_2 = \|I - (\bar{L}\bar{U})^{-1}A\|_F$ (or by $N_2 = \|I - A(\bar{L}\bar{U})^{-1}\|_F$ for right preconditioning). Letting $R = \bar{L}\bar{U} - A$ (the residual matrix), we get $I - (\bar{L}\bar{U})^{-1}A = (\bar{L}\bar{U})^{-1}R$ and it becomes clear that an incomplete factorization can be accurate (small N_1) but unstable (large N_2). Indeed, if either $\bar{L}^{-1}$ or $\bar{U}^{-1}$ has very large entries (unstable, or ill-conditioned triangular solves), then N_2 can be many orders of magnitude larger than N_1. In this case, failure of the preconditioned iteration is to be expected (see [39, 94]). Thus, for general matrices N_2 is a more telling measure of preconditioning quality than N_1. Although it is not practical to compute N_2, even a rough estimate can be quite useful. As recommended by Chow and Saad in [94], an inexpensive way of detecting ill conditioning in the ILU factors is to compute $\|(\bar{L}\bar{U})^{-1}e\|_\infty$, where e denotes the n-vector with all entries equal to 1. Note that this amounts to solving a linear system with the factored coefficient matrix $\bar{L}\bar{U}$ and e as the right-hand side. This is, of course, only a lower bound for $\|(\bar{L}\bar{U})^{-1}\|_\infty$, but it has been found to be quite useful in practice. A very large value of this quantity will reveal an ill conditioned, and therefore likely useless, preconditioner and will indicate the need to compute a better one.

As shown in the next subsection, instabilities in the preconditioner construction and application can often be avoided by suitably preprocessing the coefficient matrix A. These preprocessings consist of a combination of symmetric permutations, nonsymmetric ones, and scalings aimed at improving the conditioning, diagonal dominance, and structure of the coefficient matrix.

Finally, *a priori* information on the nature of the problem can be incorporated into a general-purpose preconditioner like ILU to obtain a stable and effective preconditioner. For example in [209, 210], good results were obtained in the solution of complex symmetric indefinite systems arising from the Helmholtz equation using variants of incomplete factorization methods that make clever use of available knowledge of the spectrum of the discrete Helmholtz operator. This shows that purely algebraic methods, such as ILU preconditioners, can be made more competitive by exploiting available information about a particular class of problems.

3.3. *The Effect of Ordering*

Incomplete factorization preconditioners are sensitive to the ordering of unknowns and equations. Reorderings have been used to reduce fill-in (as with sparse direct solvers), to introduce parallelism in the construction and application of ILU preconditioners, and to improve the stability of the incomplete factorization. In most cases, reorderings tend to affect the rate of convergence of preconditioned Krylov subspace methods.

If P and Q are permutation matrices, system (1) can be replaced by the equivalent (reordered) one

$$PAQy = Pb, \quad x = Qy, \tag{10}$$

and an incomplete factorization of PAQ is computed. Except for special cases, there is no simple relationship between the incomplete LU factors of PAQ and those of A. If A is structurally symmetric (i.e., $a_{ij} \neq 0$ if and only if $a_{ji} \neq 0$ for all $i, j \in \mathbf{n}$), then it is usually desirable to preserve this property by using symmetric permutations only; that is, $Q = P^T$. Note that symmetric permutations preserve the set of diagonal entries and do not affect the eigenvalues of A. Matrices that arise from the discretization of second-order PDEs are usually structurally symmetric, or nearly so. If A is not structurally symmetric, permutations based on the structure of the symmetrized matrix $A + A^T$ are often employed.

The effects of reorderings on the convergence of preconditioned Krylov subspace methods have been studied by a number of authors, mostly experimentally, and are still the subject of some debate (see, e.g., [21, 33, 34, 39, 43, 60, 64, 65, 87, 108, 114, 115, 123–125, 127, 128, 132, 169, 178, 201, 212, 296] and references therein). In general, the ordering used and the corresponding preconditioned iteration interact in rather complex and not-well-understood ways. Here we will confine ourselves to a brief discussion of those points on which some degree of consensus has been reached. In this subsection we consider orderings that have been originally developed for sparse direct solvers; block and parallel orderings are discussed in Sections 3.4 and 4, respectively.

Sparse matrix reorderings have been in use for a long time in connection with direct solution methods [121, 150]. Classical ordering strategies include bandwidth- and profile-reducing orderings, such as reverse Cuthill–McKee (RCM) [103], Sloan's ordering [263], and the Gibbs–Poole–Stockmeyer ordering [152]; variants of the minimum degree ordering [4, 151, 207]; and (generalized) nested dissection [149, 206].

In Fig. 1 we show the sparsity pattern of a simple five-point finite difference discretization of a diffusion operator corresponding to four orderings of the grid points: lexicographical, RCM, red–black, and nested dissection.

These orderings are based solely on the structure (graph) of the matrix and not on the numerical values of the matrix entries. For direct solvers based on complete matrix factorizations this is justified, particularly in the SPD case, where pivoting for numerical stability is unnecessary. However, for incomplete factorizations, the effectiveness of which is strongly affected by the size of the dropped entries, orderings based on graph information only may result in poor preconditioners. Thus, it is not surprising that some of the best orderings for direct solvers, such as minimum degree or nested dissection, often perform poorly when used with incomplete factorizations.

This phenomenon has been pointed out, for SPD matrices arising from diffusion-type problems in 2D, by Duff and Meurant [125]. They showed that when a minimum degree ordering of the grid points was used, the rate of convergence of conjugate gradients preconditioned with a no-fill incomplete Cholesky factorization was significantly worse than when a natural (lexicographical) ordering was used. The reason for this is that even though fewer nonzeros are likely to be dropped with a minimum-degree ordering (which is known to result in very sparse complete factors), the average size of the fill-ins is much larger, so that the norm of the remainder matrix $R = A - \bar{L}\bar{L}^T$ ends up being larger with minimum-degree than with the natural ordering (see [217, pp. 465–466]). However, fill-reducing orderings fare much better if some amount of fill-in is allowed in the incomplete factors. As shown in [125], minimum degree performs no worse than the natural ordering with a drop tolerance-based IC. Similar remarks apply to other orderings, such as nested dissection and red–black; RCM was found to be equivalent or slightly better than the natural ordering, depending on whether a level-of-fill or a drop tolerance approach was used. Hence, at least for this class of problems, there is little to be gained with the use of sparse matrix orderings. For finite element matrices, where a "natural" ordering of the unknowns may not exist, the authors

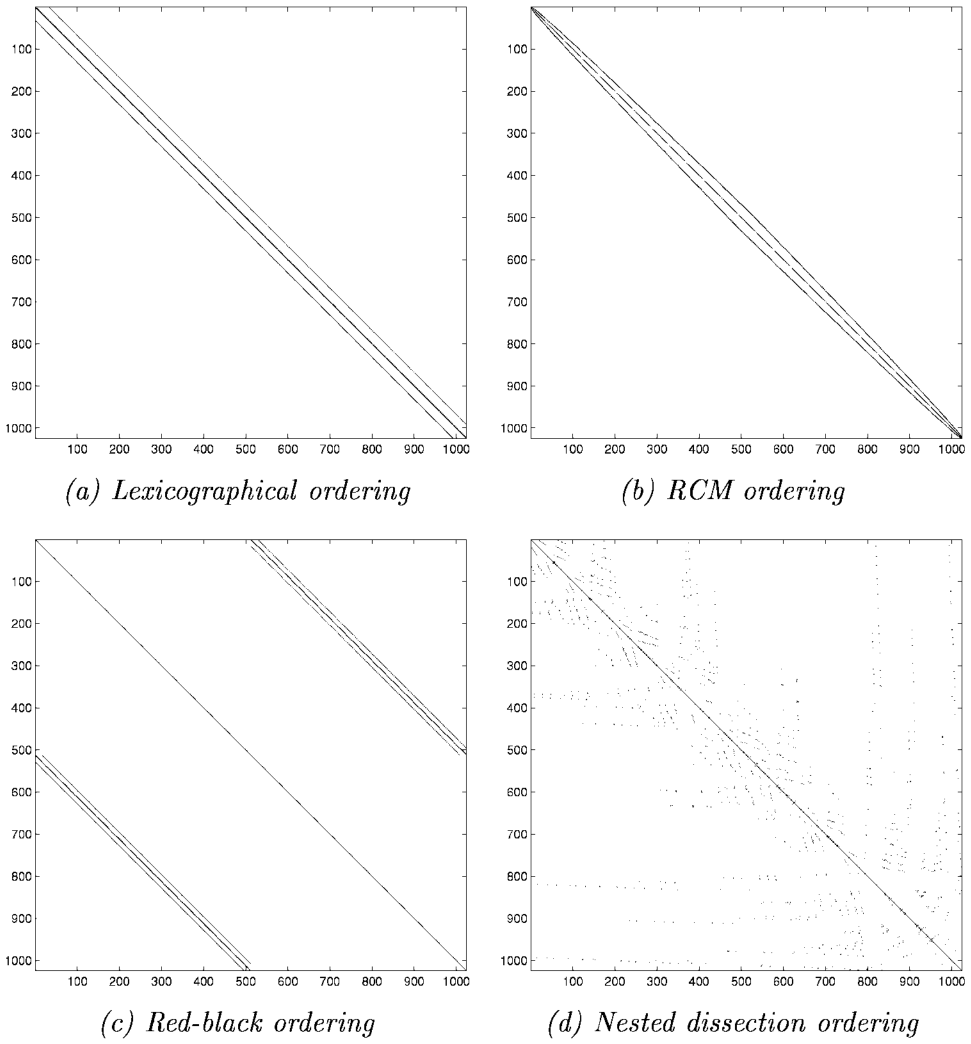

(a) Lexicographical ordering
(b) RCM ordering
(c) Red-black ordering
(d) Nested dissection ordering

FIG. 1. Matrix patterns, discrete diffusion operator with different orderings.

of [125] recommend the use of RCM. In [27, 98, 288] variants of RCM were shown to be beneficial in IC preconditioning of strongly anisotropic problems arising in oil reservoir simulations. An intuitive explanation of the good performance of RCM with IC preconditioning has been recently proposed [65]. It should be mentioned that minimum degree may be beneficial for difficult problems requiring a fairly accurate incomplete factorization and thus large amounts of fill (see, e.g., [239]).

The situation is somewhat different for nonsymmetric problems. In this case, matrix reorderings can improve the performance of ILU-preconditioned Krylov subspace solvers very significantly. This had been pointed out by Dutto, who studied the effect of ordering on GMRES with ILU(0) preconditioning in the context of solving the compressible Navier–Stokes equations on unstructured grids [127]. In systematic studies [34, 39] it was found experimentally that RCM gave the best results overall, particularly with ILUT preconditioning applied to finite difference approximations of convection-dominated convection–diffusion

TABLE IV
ILUT Results for Different Orderings, Convection–Diffusion Problem

Ordering	NO	RCM	RB	ND
Its	37	26	32	50
Density	3.6	2.8	2.4	2.5
Time	41	28	31	49

equations.[4] RCM was often better than other orderings not only in terms of performance but also in terms of robustness, with the smallest number of failures incurred for RCM. Given the low cost of RCM reordering, the conclusion in [39] was that it should be used as the default ordering with incomplete factorization preconditioners if some amount of fill-in is allowed.

As an example, we report in Table IV the results obtained for the same 2D convection–diffusion problem used to generate the results in Table I ($n = 160{,}000$ unknowns). We used ILUT(10^{-3}, 10) preconditioning of Bi-CGSTAB, with the following orderings: lexicographical (NO), reverse Cuthill–McKee (RCM), red–black (RB), and nested dissection (ND). The results for minimum-degree ordering were very similar to those with nested dissection and are not reported here.

We see from these results that red–black is better than the natural ordering when used with ILUT, as already observed by Saad [249]. However, here we find that RCM performs even better, leading to a significant reduction both in fill-in and iteration count, with consequent reduction in CPU time. On the other hand, nested dissection does poorly on this problem.

All the orderings considered so far are based solely on the sparsity structure of the matrix. For problems exhibiting features such as discontinuous coefficients, anisotropy, strong convection, and so on, these orderings can yield poor results. Matrix orderings that take into account the numerical value of the coefficients have been considered by several authors. D'Azevedo *et al.* [108] have developed an ordering technique, called the minimum discarded fill (MDF) algorithm, which attempts to produce an incomplete factorization with small remainder matrix R by minimizing, at each step, the size of the discarded fill. This method can be very effective but it is too expensive to be practical for meshes having high connectivity, such as those arising in 3D finite element analysis. Somewhat cheaper variants of MDF have been developed [109], but these are still fairly expensive while being less effective than MDF.

Several coefficient-sensitive orderings, including weighted variants of RCM and others based on minimum spanning trees (MST) and single-source problem (SSP), were studied by Clift and Tang in [98], with mixed results. For drop tolerance-based incomplete Cholesky preconditioning, the MST- and SSP-based orderings resulted in improvements on the order of 25% in solution time over RCM for SPD systems arising from finite difference and finite element approximations of diffusion-type problems. To our knowledge, these methods have not been tested on non-self-adjoint problems.

An important recent advance is the development of efficient, coefficient-sensitive nonsymmetric reorderings aimed at permuting large entries to the main diagonal of a general

[4] These matrices are numerically nonsymmetric, but structurally symmetric.

sparse matrix (see [33, 54, 123, 124, 232]). Incomplete factorization preconditioners often fail on general sparse matrices that lack nice properties such as symmetry, positive definiteness, diagonal dominance, and so forth. Thus, failure rates are high for matrices that are highly unstructured, nonsymmetric (structurally as well as numerically), and indefinite, i.e., with eigenvalues that have both positive and negative real parts. Such matrices arise frequently in the simulation of chemical engineering processes, in economic modeling, in management science, in the analysis of circuits and power system networks, and elsewhere. These problems are very different from the ones arising from the numerical solution of scalar elliptic partial differential equations and can cause serious difficulties for standard iterative methods and preconditioners. Due to the presence of many zero entries on the main diagonal, ILU preconditioners may not even be defined or unstable. For this reason, practitioners in the above-mentioned areas have had to rely almost exclusively on direct solvers.

In [232], Olschowka and Neumaier introduced new permutation and scaling strategies for Gaussian elimination. The idea was to preprocess the coefficient matrix so as to obtain an equivalent system with a matrix which is more diagonally dominant. This preprocessing reduces the need for partial pivoting, thereby speeding up the solution process, and has a beneficial effect on the accuracy of the computed solution. Although the focus in [232] is on dense systems, the sparse case and the case of incomplete factorizations were also briefly discussed. These and other heuristics have been further developed and efficiently implemented in the sparse setting by Duff and Koster (Harwell Subroutine Library MC64; see [123, 124]). Some evidence of the usefulness of these preprocessings in connection with sparse direct solvers and for ILU preconditioning has been provided in [123, 124].

A systematic experimental study of the use of these permutation and scaling algorithms on linear systems from disciplines such as chemical engineering, economics, circuits, and CFD was carried out in [33]. The nonsymmetric permutations and scalings were combined with symmetric ones (such as RCM, minimum degree, etc.) to further improve the quality of the preconditioner. That is, the original linear system (1) was first replaced by

$$(PQD_1AD_2P^T)y = PQD_1b, \quad x = D_2P^Ty, \tag{11}$$

where D_1, D_2 are row and column scalings, Q is a row permutation that moves large entries to the main diagonal, and P represents a symmetric permutation based on the structure of $\hat{A} + \hat{A}^T$, where $\hat{A} = QD_1AD_2$. Then, an incomplete factorization of the scaled and permuted matrix $PQD_1AD_2P^T$ was computed. The experiments in [33] indicate that these preprocessings, and particularly the use of permutations for finding a *maximum product transversal*, for which the product of the magnitude of the diagonal entries is maximized, allow the computation of stable and effective preconditioners in many cases.

For PDE problems these preprocessings were found to be effective in cases where the coefficient matrix had many zeros on the main diagonal, as in the presence of constraints on some of the variables, e.g., the pressure in the linearized form of the Navier–Stokes equations. A few sample results for ILUT-preconditioned Bi-CGSTAB are given in Table V for linear systems arising in CFD (LNS3937, VENKAT25), metal forming (SLIDE), chemical engineering (BAYER10), and economics (ORANI678); the last two problems are not from PDEs. Except for SLIDE, which was provided by Ivan Otero of the Lawrence Livermore National Laboratory, these matrices are available from the University of Florida Sparse Matrix Collection [107]. In the table, n denotes the order of the matrix and nnz the number

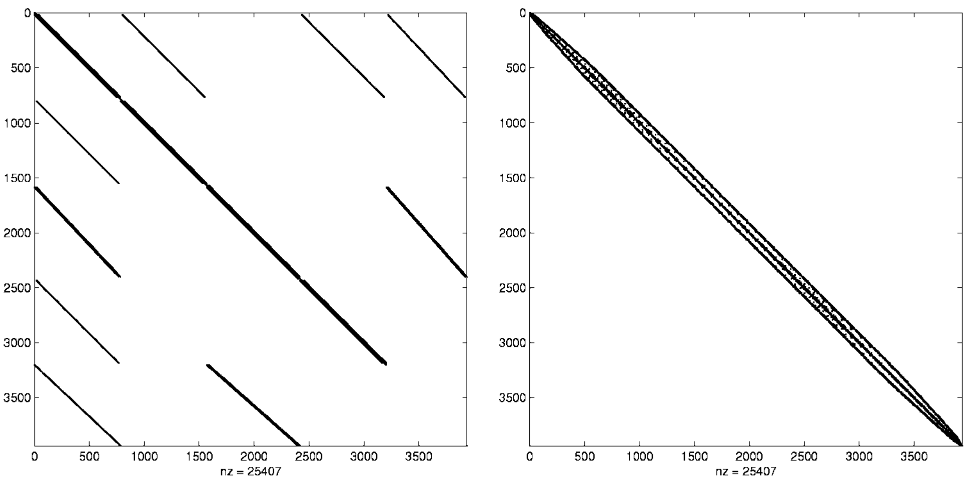
TABLE V
Test Results for ILUT with MC64 Preprocessing

Matrix	n	nnz	MC64	Density	Its	Tot-time
LNS3937	3937	25407	0.07	2.59	24	0.53
SLIDE	20191	1192535	0.98	0.09	491	69.6
VENKAT25	62424	1717792	1.43	1.05	98	53.8
BAYER10	13436	94926	0.40	1.22	65	3.81
ORANI678	2529	90158	0.11	0.09	14	0.38

of nonzero entries; under "MC64" we report the time required by the preprocessing (nonsymmetric permutations and row/columns scalings). Note that the cost of the preprocessing is small compared to the total solution costs. The results for SLIDE and VENKAT25 were obtained on one processor of an SGI Origin 2000, the remaining ones on one processor of a Sun Enterprise 450. Because of zero pivots, ILUT cannot solve any of these problems. Even for large amounts of fill, ILUT with partial pivoting (ILUTP) still cannot solve LNS3937 and BAYER10. After preprocessing with the maximum diagonal product permutation and corresponding scalings, all these problems can be easily solved by ILUT with reasonable density of the incomplete factors. The symmetric permutation used was RCM except for BAYER10, where minimum degree was used (see [33] for additional results).

In Fig. 2 we show the sparsity pattern for matrix LNS3937, which arises from a finite difference discretization of the (linearized) Navier–Stokes equations, with the original ordering and after preprocessing with MC64 (maximum product transversal) followed by RCM reordering.

In some cases it may be advantageous to combine dynamic (partial) pivoting with the static pivoting strategies in MC64. Additional papers on the use of various forms of pivoting and condition estimation to improve the robustness of ILU preconditioners include [51, 52, 296]. In summary, it can be stated that these recent advances have considerably improved



(a) LNS3937, original ordering *(b) LNS3937, MC64/RCM ordering*

FIG. 2. Matrix patterns, matrix LNS3937.

the reliability of preconditioned iterative solvers applied to general sparse matrices, thus opening the door to the use of iterative methods in application areas which were previously the exclusive domain of direct solution methods.

3.4. *Block Algorithms*

A standard technique to improve performance in dense matrix computations is to use blocking (see, e.g., [116]). By partitioning the matrices and vectors into blocks of suitable size (which usually depends on the target architecture) and by making such blocks the elementary entities on which the computations are performed, high-level BLAS (basic linear algebra subroutines) can be used for cache efficiency on current architectures with a hierarchical memory structure. As a result of such fine-tuning, computational rates near the theoretical peak are possible for many dense linear algebra calculations.

In contrast, computations involving sparse matrices are much more difficult to optimize, particularly when the matrices are irregularly populated, and computational rates are typically only a fraction of the theoretical peak. This is largely caused by the presence of indirect addressing in the innermost loops, e.g., in multiplying a sparse matrix by a dense vector. With large unstructured matrices, most floating point operations are scattered around the memory and are out-of-cache.

The situation can sometimes be improved by extending the use of blocking to sparse matrices. Block variants of incomplete factorization preconditioning have been used for many years in the solution of block tridiagonal linear systems arising from the discretization of partial differential equations on structured grids (see, e.g., [13, 100, 179, 195, 275], and the relevant chapters in [12, 217]). Here the blocks arise from some natural partitioning of the problem (grid lines, planes, or subdomains) and they are usually large and sparse. For structured grid problems in 2D, efficient band solvers can be used to invert these blocks, leading to good performance in many cases. Effective 3D variants have been recently developed (see, e.g., [211] and references therein).

General block incomplete factorizations valid for arbitrary partitionings of the matrix into blocks have been considered in [11]. Computational experience with block incomplete factorization (sometimes referred to as BILU) has been reported in [82, 92, 93]; available software include the BPKIT package described in [92].

Of particular interest is the situation where the blocks are small and dense, as is the case when several variables are associated with a grid point, as with multicomponent problems described by systems of partial differential equations. If variables associated with the same node are numbered consecutively, the matrix has a natural block structure. The blocks will be dense if variables at the same node are all mutually coupled. In this case high-level BLAS can be used as computational kernels and indirect addressing can be removed from the innermost loops, e.g., in the execution of matrix–vector products. For cache-based architectures, this leads to fairly good performance (see, e.g., [36, 226]).

The block structure can either be naturally present in the matrix, or it must be imposed. As already mentioned, matrices with a natural block form often arise when the finite element method is used to discretize a partial differential equation or a system of them. In other cases there may be no natural block structure to exploit. However, it is still possible to use block algorithms by imposing a suitable blocking on the matrix, for instance by reordering the matrix using row and column permutations. A natural goal of the permutation should be to result in a block partitioned matrix with fairly dense blocks. To this end, any band/profile-minimizing heuristic like RCM can be used. Another possibility is to use the PABLO (parameterized block ordering) algorithm and its variants (see [88, 233]). These schemes result in block partitioned forms in which the dense blocks tend to be clustered around the main diagonal of the matrix.

Yet another possibile way to construct the blocks is to use the graph compression procedure attributed to Ashcraft [8] (see also [252]). This method gave excellent results in applications to solid and structural mechanics problems in [36]. This block construction is based on the properties of the underlying graph. The aim of the graph compression is to find *cliques* in the undirected graph of the matrix. Consider the graph $G_A = (V, E)$ of the symmetric matrix $A = [a_{ij}]$. $V = \{1, \ldots, |V|\}$ is the set of vertices that correspond to rows and columns of the matrix. $E \subseteq V \times V$ is the set of its edges, where $(i, j) \in E$ if and only if the entry a_{ij} is nonzero. Then the adjacency set of a vertex v is

$$\text{adj}(v) = \{u \mid (v, u) \in E\}.$$

Vertices $u \in V$ and $v \in V$ are adjacent if and only if $v \in \text{adj}(u)$ (this is the same condition as $u \in \text{adj}(v)$). A *clique* in G_A is a set of vertices which are all mutually adjacent. The task of finding blocks as large and dense as possible in a matrix A is equivalent to that of finding all cliques which are maximal with respect to inclusion in the graph G_A. (More accurately, we are after all cliques of indistinguishable nodes; see [8].) Of course, this task could be accomplished by brute force, comparing the adjacency sets of the vertices in G_A. However, this would result in an unnecessarily time-consuming procedure. In [8], some basic enhancements of this procedure are described. Before actual computation and comparison of the adjacency sets take place, a preprocessing phase is performed. The candidates for inclusion into actual blocks are tested with respect to two easily computed additional quantities: vertex degree (size of an adjacency set) and vertex checksum (a simple function of adjacent vertices). These tests enable one to compute and compare adjacency sets in a few cases only. This contributes to the low cost of the graph compression routine.

The graph compression procedure was proposed as a tool for improving the performance of sparse direct solvers based on triangular factorizations of the system matrix. In this context, the compression contributes in a crucial way to reducing the symbolic overhead associated with the decomposition (reordering, elimination tree manipulation, symbolic factorization, etc.) As for the numeric phase of the factorization, it represents a useful complement to another important blocking strategy used in modern direct solvers, namely, the supernodal technique [208]. Note that the supernodal strategy is actually based on the structure of the triangular factors of A, which are usually much less sparse than the system matrix itself. For information on the use of supernodes in incomplete factorization preconditioners, see [226].

In Table VI we report some performance measurements on two different computers, an SGI Origin 2000 and a Compaq Alpha server, for two block preconditioners, B-SAINV (a block sparse approximate inverse preconditioner to be described later) and a block variant of the Ajiz–Jennings incomplete Cholesky factorization (denoted B-AJ in the table). In these algorithms, blocks are dropped from the incomplete factors if they are smaller than a prescribed drop tolerance in norm (we used the infinity norm, but other choices are possible). The efficiency of the block methods is studied by recording the Mflop rates for a sequence of cylindrical shell problems starting from a 30×30 element mesh having $n = 5489$ unknowns to a fine discretization by 180×180 element mesh with $n = 194{,}939$

TABLE VI
Cylindrical Shell, Mflop Rates

			Origin 2000[a]		AlphaServer GS140[b]	
Mesh	n	nnz	B-SAINV	B-AJ	B-SAINV	B-AJ
30×30	5489	112505	173	128	217	180
40×40	9719	201605	160	128	134	110
50×50	15149	316505	139	109	101	90
60×60	21779	457205	101	92	81	78
70×70	29609	623705	83	80	76	75
80×80	38639	816005	77	73	75	70
90×90	49136	1034105	73	67	75	65
100×100	60299	1278005	71	67	74	62
140×140	118019	2511605	65	57	68	53
180×180	194939	4158005	65	62	67	53

[a] Compiled with -mips4 -64 -r12k -OPT : Olimit = 8000 -03 command.
[b] Compiled with -fkapargs = '-o = 5 -so = 3' command.

unknowns. In all cases the number of nonzeros in the preconditioner is approximately the same as the number of nonzeros in the lower triangular part of the stiffness matrix, denoted by nnz. Here the typical block size is six, as there are three translational and three rotational degrees of freedom at each node. Graph compression was used to identify the matrix blocks. The maximum performance of the SGI Origin 2000 R12k processor is 600 Mflops; thus, for problems small enough to fit in the processor's external cache (8 Mb = one million double precision real numbers) we can achieve over 25% of the processor's maximum performance.[5] For bigger problems we can see a rather rapid decay of performance with respect to flop rates. It is interesting that the flop rate for the preconditioner construction phase is scarcely affected by the dimension of the problem and was typically between 200 and 212 Mflops on the Origin and between 260 and 310 Mflops on the Alpha server, regardless of problem size.

Note that the sparse approximate inverse preconditioner, the application of which requires matrix–vector products rather than triangular solves, results in higher Mflop rates than the block incomplete factorization preconditioner; however, the difference becomes small for larger problems.

In general, block incomplete factorizations appear to be more robust than standard (point) ones with respect to breakdowns and often result in improved rates of convergence for difficult problems [36, 82]. Nevertheless, instabilities can happen in the block case as well, and safeguarding mechanisms have been devised to deal with such difficulties. Consider a matrix A partitioned in block form as

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \ddots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix}. \tag{12}$$

[5] For the Compaq AlphaServer GS140 EV6 processor, the external cache is 4 MB.

Here A_{ij} has order $n_i \times n_j$, where $1 \le n_i \le n$, p is the block dimension of the matrix, and n is its dimension. Block incomplete factorization algorithms involve block updates of the form

$$A_{ij} \leftarrow A_{ij} - A_{ik}A_{kk}^{-1}A_{kj}, \tag{13}$$

and instabilities could arise if any of the *pivot blocks* A_{kk} becomes very ill conditioned (nearly singular). For special classes of matrices it can be shown that no instabilities arise (see [12]), but for general sparse matrices no such result holds, and some form of stabilization becomes necessary. One stabilization strategy, described in [94], uses a singular value decomposition (SVD) of the pivot blocks,

$$A_{kk} = U\Sigma V,$$

where Σ is the diagonal matrix of singular values $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_{n_k}$ and U and V are orthogonal matrices. If any of the singular values are smaller than $\alpha\sigma_1$, they are replaced by $\alpha\sigma_1$; here $0 < \alpha \le 1$ is a parameter. The resulting modified pivot block has spectral condition number bounded above by $1/\alpha$.

4. PARALLEL ILU METHODS

Until recently, ILU preconditioners were widely believed to be ill suited for implementation on parallel computers with more than a few processors. The reason is that Gaussian elimination, on which ILU techniques are based, offers limited scope for parallelization. Furthermore, the forward and backward triangular solves that form the preconditioning operation are highly sequential in nature, and parallelism in these operations is not readily apparent.

In the case of no-fill ILU or incomplete Cholesky preconditioning, a good degree of parallelism can be achieved through graph coloring techniques. Given a graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, the *graph coloring problem* is to construct a partition $(C_1, C_2, \ldots, C_c)$ of the set V such that all vertices in the same part C_i form an independent set; i.e., vertices in the same subset are not connected by any edge. The minimum number of colors necessary for a graph, $\chi = c_{min}$, is the *chromatic number* of the graph. The relevance of the chromatic number from a computational point of view is that all unknowns in the same subset can be solved in parallel. Thus, the number of inherent sequential steps is greater than or equal to the chromatic number of the graph.

Let A be a structurally symmetric $n \times n$ matrix, and let the corresponding undirected graph be G_A. For arbitrary A, finding the chromatic number of G_A is NP-hard, but in practice a suboptimal coloring suffices. For the five-point stencil discretization, it is easy to find a two-coloring of the graph, commonly called *red–black* (RB) coloring. For red–black coloring, the degree of parallelism in applying an IC(0) or ILU(0) preconditioner is $n/2$ (n is the number of unknowns), which justifies considering this ordering strategy [182]. The problem with this approach is that a no-fill preconditioner obtained with RB may result in poor convergence. For SPD matrices arising from finite difference discretizations of two-dimensional elliptic problems, Duff and Meurant [125] have shown, by way of experiments, that RB reordering has a negative effect on the convergence of conjugate gradient preconditioned with IC(0). Poole and Ortega [238] have observed similar convergence behavior for multicolorings. As already mentioned (see Table IV), RB ordering may result in good convergence rates when some fill-in is allowed, but then the degree of parallelism in the application of the preconditioner is drastically reduced.

Domain decomposition techniques can be used to introduce parallelism in incomplete factorization methods. Additive Schwarz methods (ASM) [264] derive a preconditioner by decomposing the problem domain into a number of possibly overlapping subdomains, (approximately) solving each subdomain, and summing the contributions of the subdomain solves. Unknowns corresponding to grid points in the overlap of two or more regions require a special treatment, often being updated through some kind of averaging scheme. Fill-in is allowed only within subdomains, not between them. Variants of ASM are obtained by varying the amount of overlap and the subdomain solvers. When the subdomains are solved approximately using an incomplete factorization, the resulting preconditioner can be thought of as a "parallel ILU" strategy (see [18] for an efficient implementation). Like block Jacobi, ASM has good parallelism and locality, but these advantages could be offset by a high iteration count of the underlying Krylov subspace method. In this case, a coarse grid correction can often be used to keep the number of iterations from growing, leading to a two- (or multi-) level method (see [264]). The role of the coarse grid correction is to recover some of the global information that is lost when all fill-in between subdomains (matrix subblocks) is ignored.

A comparison of multicoloring techniques and (one level) domain decomposition-based incomplete Cholesky and ILU methods is presented in [34], where several diffusion and convection–diffusion problems in 2D and 3D were used for the numerical experiments. It was found that for most 2D problems, ASM preconditioning with ILU solves on the subdomains outperformed multicoloring-based parallel ILU methods, which suffered from high iteration counts. For 3D problems, the degradation in convergence rates suffered by multicoloring ILU was more contained, and the two methods gave comparable performance. This was true both for symmetric and for mildly nonsymmetric (diffusion dominated) problems. For convection-dominated problems in 2D both ILU with RB ordering and ASM performed rather poorly. However, in 3D, ILU with RB ordering was found to be more robust than additive Schwarz with ILU subdomain solves (see [34]).

There are, however, orderings that are highly parallel and do not lead to any degradation of convergence rates. For simple 2D problems on regular grids, such orderings (the so-called *van der Vorst orderings*) have been described in [278] (see also [125]). These orderings are based on domain decomposition with a "local" ordering of the nodes within each subdomain, that is, an ordering that gives numbers that are not too far from neighboring grid points (or unknowns). Furthermore, fill-in between subdomains is not completely ignored, as in more traditional ASM-type methods.

Parallel ILU preconditioners of this type have been recently developed by several researchers (see [177, 178, 212]). The importance of this work consists of showing that it is possible to develop highly parallel implementations of general ILU preconditioners with arbitrary levels of fill-in without having to pay a price in terms of rates of convergence. These new ILU algorithms have been tested on scalar elliptic PDE problems in two and three space dimensions. The theoretical analysis and experimental results in [177, 178] indicate that the proposed algorithms exhibit good scalability up to several hundred processors [177, 178]. It was also shown in [177] that these methods are clearly superior to one-level ASM preconditioning, mostly because the rate of convergence is scarcely affected by the number of processors being used (for a fixed problem size).

Very brifley, these algorithms consist of the following steps: graph partitioning (e.g., using the highly efficient techniques described in [188]), incomplete elimination of interior nodes in a subdomain before boundary nodes, and coloring the subdomains to process the boundary nodes in parallel. For ILU(0), such an algorithm has been described by Saad [251, pp. 374–376]. When fill-in is allowed, fill between subdomains has to be dealt with. The crucial fact is that it is often possible to allow enough fill-in between subdomains to preserve a high rate of convergence, while at the same time discarding enough of it to retain a high degree of parallelism. Thus, a scalable parallel preconditioner can be achieved. So far the effectiveness of these techniques has been demonstrated only for fairly simple model problems with regular geometries and discretizations. However, we expect these techniques to prove effective also for more realistic problems, including unstructured ones.

A somewhat different approach to parallel block overlapped incomplete factorization preconditioning has been introduced by Kaporin and Konshin [187] and tested on real-life linear elasticity problems. The preconditioner suffers from only moderate growth in the number of iterations as the number of processors (subdomains) grows, thus achieving fairly good parallel scalability.

Other recent papers on parallel ILU preconditioning include [115, 165, 189, 220, 285]. It has now become clear that high parallelism can be achieved with ILU preconditioners, albeit considerable sophistication and ingenuity are needed if one is to develop efficient implementations. In the following section we discuss an alternative class of preconditioning methods which have more natural parallelism.

5. SPARSE APPROXIMATE INVERSES

Preconditioning techniques based on sparse approximate inverses have been vigorously developed in recent years. The common idea underlying this class of algorithms is that a sparse matrix $M \approx A^{-1}$ is explicitly computed and used as a preconditioner for Krylov subspace methods for the solution of (1). The main advantage of this approach is that the preconditioning operation can easily be implemented in parallel, since it consists of matrix–vector products. Further, the construction and application of preconditioners of the approximate inverse type tend to be immune from such numerical difficulties as pivot breakdowns and instability, and indeed these techniques have been shown to be remarkably robust in practice.

Approximate inverse techniques rely on the assumption that for a given sparse matrix A, it is possible to find a sparse matrix M which is a good approximation, in some sense, of A^{-1}. However, this is not at all obvious, since the inverse of a sparse matrix is usually dense. More precisely, it can be proved that the inverse of an irreducible sparse matrix is structurally full. This means that for a given irreducible sparsity pattern, it is always possible to assign numerical values to the nonzeros in such a way that all entries of the inverse will be nonzero (see [120]). Nevertheless, it is often the case that many of the entries in the inverse of a sparse matrix are small in absolute value, thus making the approximation of A^{-1} with a sparse matrix possible. For instance, a classical result of Demko *et al.* [112] states that if A is a banded symmetric positive definite matrix, then the entries of A^{-1} are bounded in an exponentially decaying manner along each row or column. More precisely, there exist $0 < \rho < 1$ and a constant C such that

$$|[A^{-1}]_{ij}| \leq C\rho^{|i-j|} \quad \text{for all } i, j = 1, 2, \ldots, n.$$

This result can be regarded as a discrete analogue of the decay of the Green's function of a second-order, self-adjoint differential operator (see also [216]). The numbers ρ and C depend on the bandwidth and on the spectral condition number of A. For matrices having a large bandwidth and/or a high condition number, C can be very large and ρ very close to 1, so that the decay could actually be so slow to be virtually imperceptible. On the other hand, if A is strongly diagonally dominant, then the entries of A^{-1} can be shown to decay rapidly. Diagonal dominance, however, is not necessary for sparse approximate inverse preconditioners to perform well.

5.1. *Overview of Algorithms*

Not surprisingly, there exist several completely different algorithms for computing a sparse approximate inverse, with each approach having its own strengths and limitations. In the following we restrict ourselves to a discussion of those methods that have been shown to be especially competitive in terms of performance and robustness. A more complete overview, including developments up to 1999, can be found in [42].

It is customary to distinguish between two basic types of approximate inverses, depending on whether the preconditioner $M \approx A^{-1}$ is expressed as a single matrix or as product of two or more matrices. The latter type of preconditioners are known as factored sparse approximate inverses, and they are of the form

$$M = M_U M_L, \quad \text{where } M_U \approx U^{-1}, \quad \text{and} \quad M_L \approx L^{-1},$$

where L and U are the lower and upper triangular factors of A (we are assuming here that A has an LU factorization).

Within each class there are several different techniques, depending on the algorithm used to compute the approximate inverse or approximate inverse factors. At present, there are two main approaches: Frobenius norm minimization, and incomplete (bi-) conjugation.

5.1.1. Frobenius norm minimization. This class of approximate inverse techniques was the first to be proposed and investigated, back in the early 1970s (see Benson [28]). Other early papers include [31, 145]. The basic idea is to compute a sparse matrix $M \approx A^{-1}$ as the solution of the constrained minimization problem

$$\min_{M \in \mathcal{S}} \|I - AM\|_F,$$

where $\mathcal{S}$ is a set of sparse matrices and $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. Since

$$\|I - AM\|_F^2 = \sum_{j=1}^{n} \|e_j - Am_j\|_2^2,$$

where e_j denotes the jth column of the identity matrix, the computation of M reduces to solving n independent linear least-squares problems, subject to sparsity constraints.

Notice that the above approach produces a right approximate inverse. A left approximate inverse can be computed by solving a constrained minimization problem for $\|I - MA\|_F = \|I - A^T M^T\|_F$. This amounts to computing a right approximate inverse for A^T and taking the transpose of the resulting matrix. In the case of nonsymmetric matrices, the distinction between left and right approximate inverses can be important. Indeed, there are situations where it is difficult to compute a good right approximate inverse but easy to find a good left approximate inverse. Furthermore, when A is nonsymmetric and ill conditioned, a matrix $M \approx A^{-1}$ may be a poor right approximate inverse but a good left approximate inverse. In the following discussion, we shall assume that a right approximate inverse is being computed.

In early papers, the constraint set $\mathcal{S}$, consisting of matrices with a given sparsity pattern, was prescribed at the outset. Once $\mathcal{S}$ is given, the computation of M is straightforward, and it is possible to implement such computation efficiently on a parallel computer. In a distributed memory environment, the coefficient matrix A can be distributed among the processors before the computation begins, and the construction of M is a local process which can be done with little communication among processors.

When the sparsity pattern is fixed in advance, the construction of the preconditioner can be accomplished as follows. The nonzero pattern is a subset $\mathcal{G} \subseteq \{(i, j) \mid 1 \leq i, j \leq n\}$ such that $m_{ij} = 0$ if $(i, j) \notin \mathcal{G}$. Thus, the constraint set $\mathcal{S}$ is simply the set of all real $n \times n$ matrices with nonzero pattern contained in $\mathcal{G}$. Denote by m_j the jth column of M $(1 \leq j \leq n)$. For a fixed j, consider the set $\mathcal{J} = \{i \mid (i, j) \in \mathcal{G}\}$, which specifies the nonzero pattern of m_j. Clearly, the only columns of A that enter the definition of m_j are those whose index is in $\mathcal{J}$. Let $A(:, \mathcal{J})$ be the submatrix of A formed from such columns, and let $\mathcal{I}$ be the set of indices of nonzero rows of $A(:, J)$. Then we can restrict our attention to the matrix $\hat{A} = A(\mathcal{I}, \mathcal{J})$, to the unknown vector $\hat{m}_j = m_j(\mathcal{J})$, and to the right-hand-side $\hat{e}_j = e_j(\mathcal{I})$. The nonzero entries in m_j can be computed by solving the (small) unconstrained least-squares problem

$$\|\hat{e}_j - \hat{A}\hat{m}_j\|_2 = \min.$$

This least-squares problem can be solved, for instance, by means of the QR factorization of $\hat{A}$. Clearly, each column m_j can be computed, at least in principle, independently of the other columns of M. Note that due to the sparsity of A, the submatrix $\hat{A}$ will contain only a few nonzero rows and columns, so each least-squares problem has small size and can be solved efficiently by dense matrix techniques.

The main difficulty with this approach is the choice of $\mathcal{S}$, that is, how to pick a sparsity pattern for M that will result in a good preconditioner. For simple problems, it is common to impose on M the sparsity pattern of the original matrix A. However, as the following example shows, this does not work well in general.

In Fig. 3 we show sparsity patterns for a nonsymmetric matrix A of order $n = 1509$ with $nnz = 45{,}090$ nonzeros from Barnard and Clay [24]. This matrix comes from an application of the ALE3D method to a solid deformation problem. On the left we show the pattern of the original matrix, and on the right the sparsity pattern of an approximate inverse with $n = 70{,}572$ nonzeros obtained with the algorithm in [40]. Using the approximate inverse M as a preconditioner for the Bi-CGSTAB method applied to $Ax = b$ results in convergence in 39 iterations. (Without preconditioning, Bi-CGSTAB does not converge in 10,000 iterations.) For this problem, using an approximate inverse with the same sparsity pattern as A does not work, because A^{-1} has large entries outside the pattern of A.

Because for general sparse matrices it is difficult to prescribe a good nonzero pattern for M, several authors have developed adaptive strategies which start with a simple initial guess (for example, a diagonal matrix) and successively augment this pattern until a criterion of the type $\|e_j - Am_j\|_2 < \varepsilon$ is satisfied for a given $\varepsilon > 0$ (for each j), or a maximum number of nonzeros in m_j has been reached. Such an approach was first proposed by Cosgrove *et al.*

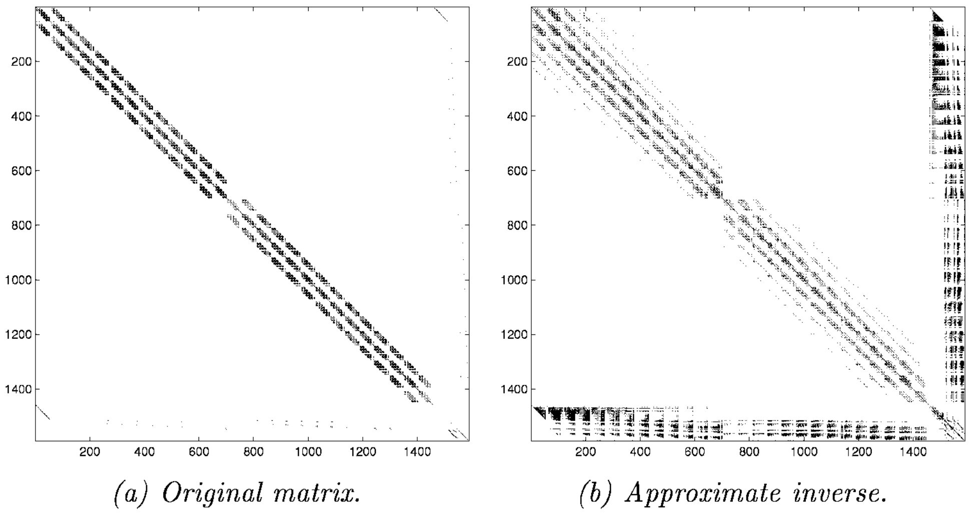


FIG. 3. Sparsity patterns, matrix ALE3D.

[102]. Slightly different strategies were also considered by Grote and Huckle [163] and by Gould and Scott [158].

The most successful of these approaches is the one proposed by Grote and Huckle, hereafter referred to as the SPAI preconditioner [163]. The algorithm runs as follows.

ALGORITHM 5.1. SPAI algorithm:
For every column m_j of M:

(1) Choose an initial pattern $\mathcal{J}$.

(2) Determine the row indices $\mathcal{I}$ of the corresponding nonzero entries and the QR decomposition of $\hat{A} = A(\mathcal{I}, \mathcal{J})$. Then compute the solution $\hat{m}_j$ of the least-squares problem $\|\hat{e}_j - \hat{A}\hat{m}_j\|_2 = \min$ and its residual $r = \hat{e}_j - \hat{A}\hat{m}_j$.

While $\|r\|_2 > \varepsilon$:

(3) Set $\mathcal{L}$ equal to the set of indices ℓ for which $r(\ell) \neq 0$.

(4) Set $\tilde{\mathcal{J}}$ equal to the set of all new column indices of A that appear in all $\mathcal{L}$ rows but not in $\mathcal{J}$.

(5) For each $k \in \tilde{\mathcal{J}}$ compute the norm ρ_k of the new residual via the formula

$$\rho_k^2 = \|r\|_2^2 - \frac{(r^T A e_k)^2}{\|A e_k\|_2^2}$$

and delete from $\tilde{\mathcal{J}}$ all but the most profitable indices.

(6) Determine the new indices $\tilde{\mathcal{I}}$ and update the QR factorization of $A(\mathcal{I} \cup \tilde{I}, \mathcal{J} \cup \tilde{\mathcal{J}})$. Then solve the new least-squares problem, compute the new residual $r = e_j - Am_j$, and set $\mathcal{I} = \mathcal{I} \cup \tilde{\mathcal{I}}$ and $\mathcal{J} = \mathcal{J} \cup \tilde{\mathcal{J}}$.

This algorithm requires the user to provide an initial sparsity pattern, and several parameters. These parameters are the tolerance ε on the residuals, the maximum number of new nonzero entries actually retained at each iteration (that is, how many of the most profitable indices are kept at step (5)), and also the maximum number of iterations for the loop (3)–(6). The latter two parameters together determine the maximum number of nonzeros allowed in each column of the approximate inverse.

The parallelization of this adaptive algorithm, as can be expected, is highly nontrivial and requires a high degree of ingenuity. Details of parallel implementations of SPAI can be found in [23, 24].

Unfortunately, the setup time for the adaptive SPAI preconditioner is often very high, even in parallel (see [42, 23]). Therefore, some effort has been put into finding ways to reduce the construction cost of SPAI. This was the motivation for Chow and Saad's development of the MR (for Minimal Residual) method [96]. In this algorithm, the exact minimization of $\|I - AM\|_F$ is replaced by an approximate minimization obtained by performing a few iterations of a minimal residual-type method applied to $Am_j = e_j$. No sparsity pattern needs to be prescribed in advance, as large entries emerge automatically after a few iterations, and small ones are dropped. The resulting preconditioner, however, appears to be less robust than SPAI [42].

Another line of research is to develop techniques for automatically choosing a good sparsity pattern. A simple idea is to take the sparsity pattern of M to be that of A^k, where k is a positive integer, $k \geq 2$. This approach can be justified in terms of the Neumann series expansion of A^{-1}. While the approximate inverses corresponding to higher powers of A are often better than the one corresponding to $k = 1$, they may be too expensive to compute and apply. In many cases of practical interest, it is possible to considerably reduce storage and computational work by working with *sparsified* matrices. Dropping entries below a prescribed threshold in A produces a new matrix $\hat{A} \approx A$; using the structure of $\hat{A}^2$ often results in a good sparsity pattern for M [90]. Postfiltration (i.e., *a posteriori* removal of small entries) is used to reduce the cost of applying the preconditioner, usually without adversely affecting the rate of convergence. Some care must be exercised to retain entries in A if their dropping would result in a singular M (see [90]). Other techniques for pattern selection have been proposed by Huckle [174], who paid particular attention to the case when A was strongly nonsymmetric.

The parallel performance and scalability of sparse approximate inverses based on Frobenius norm minimization with prescribed sparsity patterns has been studied by Chow [91]. Chow's code ParaSails (for parallel sparse approximate inverse least-squares preconditioner) has been used to solve finite element elasticity problems with more than 4 million equations on 1000 processors of ASCI Blue Pacific (IBM SP), and it has also been tested on anisotropic diffusion problems with up to 216 million equations.[6]

A factored approximate inverse preconditioner based on Frobenius norm minimization has been introduced by Kolotilina and Yeremin [196]. This is known as the FSAI algorithm. The motivation for this work was that when A is SPD, the preconditioner obtained by Frobenius norm minimization of $I - AM$ (or $I - MA$) is not SPD in general and cannot be used with the conjugate gradient method. To remedy this situation, Kolotilina and Yeremin proposed computing a sparse lower triangular matrix $G_L \approx L^{-1}$, where L is the Cholesky factor of A. Now the preconditioner is given by $M = G_L^T G_L$ and is guaranteed to be SPD as long as G_L has no zero diagonal entries.

Remarkably, the entries of G_L can be computed directly from A, without any knowledge of the Cholesky factor L itself; moreover, the algorithm for computing G_L is naturally parallel and cannot break down for a general positive definite A. The FSAI technique can be briefly described as follows. Let $\mathcal{S}_L$ be a prescribed lower triangular sparsity pattern which includes the main diagonal. Then a lower triangular matrix $\hat{G}_L$ is computed by

[6] The ParaSails code is available online at http://www.llnl.gov/casc/parasails/.

solving the matrix equation

$$(A\hat{G}_L)_{ij} = \delta_{ij}, \quad (i, j) \in \mathcal{S}_L.$$

Here $\hat{G}_L$ is computed by columns: each column requires the solution of a small "local" SPD linear system, the size of which is equal to the number of nonzeros allowed in that column. The diagonal entries of $\hat{G}_L$ are all positive. Define $\hat{D} = (\mathrm{diag}(\hat{G}_L))^{-1}$ and $G_L = \hat{D}^{\frac{1}{2}}\hat{G}_L$; then the preconditioned matrix $G_L A G_L^T$ is SPD and has diagonal entries all equal to 1. Furthermore, the resulting sparse inverse factor G_L can be shown to be optimal, in the sense that $X = G_L$ minimizes the quantity $\|I - XL\|_F$ over all lower triangular matrices with sparsity pattern $\mathcal{S}_L$, with the additional constraint that the entries of the preconditioned matrix XAX^T be all equal to 1 (see [196] and the related paper [185]; see [194] for improvements of the basic FSAI algorithm).

As before, the main issue is the selection of a good sparsity pattern for G_L. A common choice is to allow nonzeros in G_L only in positions corresponding to nonzeros in the lower triangular part of A. More advanced (but still simple) pattern selection strategies have been studied by Chow [90]. Parallel implementations of FSAI have been described in [47, 91, 111, 140]. Recently, Huckle [175, 176] developed a dynamic version of FSAI which does not require a sparsity pattern to be prescribed in advance.

The FSAI algorithm is robust for general SPD matrices and has performed well on difficult problems from structural mechanics [32, 197]. Improvements to the original FSAI algorithm have been proposed [141, 194]. Although the method can be extended to the nonsymmetric case, it has been found to be unreliable for general sparse matrices (see [42]), where it is generally outperformed by the AINV algorithm discussed in the next subsection.

It is interesting to note that sparse approximate inverse techniques have been successfully applied not only to the solution of sparse linear systems, but also to large *dense* systems, including those arising from boundary element and fast multipole methods. Generally speaking, the best results have been obtained with a fairly simple approximate inverse based on Frobenius norm minimization with respect to a prescribed sparsity pattern (see, for example, [1, 3, 57, 74, 75, 84–86, 105, 106, 193, 205]). An important observation is that in some problems, a good sparse approximate inverse can be computed even if the coefficient matrix A is not explicitly available (as with the fast multipole method). As a result, it is now possible to solve dense complex linear systems from scattering calculations with over 1 million unknowns in just a couple of hours of CPU time on a 16-processor Compaq SC Alpha server [76].

5.1.2. Incomplete biconjugation: The AINV algorithm. Factored approximate inverse preconditioners for general sparse matrices can be efficiently constructed by means of a generalization of the Gram–Schmidt process known as *biconjugation*. Biconjugation provides a way to compute a triangular factorization of A^{-1} (rather than of A), working only with the matrix A. If A admits the factorization $A = LDU$, where L is unit lower triangular, D is diagonal, and U is unit upper triangular, then A^{-1} can be factored as $A^{-1} = U^{-1}D^{-1}L^{-1} = ZD^{-1}W^T$, where $Z = U^{-1}$ and $W = L^{-T}$ are unit upper triangular matrices. Note that in general, the inverse factors Z and W will be rather dense. For instance, if A is an irreducible band matrix, they will be completely filled above the main diagonal. Factored sparse approximate inverse preconditioners can be constructed by computing sparse approximations $\bar{Z} \approx Z$ and $\bar{W} \approx W$. The factored approximate inverse is then

$$M = \bar{Z}\bar{D}^{-1}\bar{W}^T \approx A^{-1},$$

where $\bar{D}$ is a nonsingular diagonal matrix, $\bar{D} \approx D$. The FSAI method described above can be adapted to compute such approximations. A more robust approach, however, is the one based on biconjugation. This approach, hereafter referred to as the AINV method, is described in detail in [38, 40]. The AINV method does not require that the sparsity pattern be known in advance and is applicable to matrices with general sparsity patterns. The construction of the AINV preconditioner is based on an algorithm which computes two sets of vectors $\{z_i\}_{i=1}^n$, $\{w_i\}_{i=1}^n$ which are A-biconjugate, i.e., such that $w_i^T A z_j = 0$ if and only if $i \neq j$. Given a real nonsingular $n \times n$ matrix A, there is a close relationship between the problem of inverting A and that of computing two sets of A-biconjugate vectors $\{z_i\}_{i=1}^n$ and $\{w_i\}_{i=1}^n$. If

$$Z = [z_1, z_2, \ldots, z_n]$$

is the matrix whose ith column is z_i and

$$W = [w_1, w_2, \ldots, w_n]$$

is the matrix whose ith column is w_i, then

$$W^T A Z = D = \begin{pmatrix} p_1 & 0 & \ldots & 0 \\ 0 & p_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & p_n \end{pmatrix},$$

where $p_i = w_i^T A z_i \neq 0$. It follows that W and Z are necessarily nonsingular and

$$A^{-1} = Z D^{-1} W^T = \sum_{i=1}^{n} \frac{z_i w_i^T}{p_i}. \tag{14}$$

Hence, the inverse of A is known if two complete sets of A-biconjugate vectors are known. Note that there are infinitely many such sets. Matrices W and Z whose columns are A-biconjugate can be explicitly computed by means of a biconjugation process applied to the columns of any two nonsingular $n \times n$ matrices $W^{(0)}$, $Z^{(0)}$. A computationally convenient choice is to let $W^{(0)} = Z^{(0)} = I$: the biconjugation process is applied to the unit basis vectors. In order to describe the procedure, let a_i^T and c_i^T denote the ith row of A and A^T, respectively (i.e., c_i is the ith column of A). The basic A-biconjugation procedure can be written as follows.

ALGORITHM 5.2. Biconjugation algorithm:

```
(1) Let $w_i^{(0)} = z_i^{(0)} = e_i \quad (1 \le i \le n)$
(2) For $i = 1, 2, \ldots, n$ do
(3)     For $j = i, i+1, \ldots, n$ do
(4)         $p_j^{(i-1)} := a_i^T z_j^{(i-1)}$;   $q_j^{(i-1)} := c_i^T w_j^{(i-1)}$
```

```
(5)      End do
(6)      if $i = n$ go to (11)
(7)      for $j = i + 1, \ldots, n$ do
(8)          $z_j^{(i)} := z_j^{(i-1)} - \left(\frac{p_j^{(i-1)}}{p_i^{(i-1)}}\right) z_i^{(i-1)}; \quad w_j^{(i)} := w_j^{(i-1)} - \left(\frac{q_j^{(i-1)}}{q_i^{(i-1)}}\right) w_i^{(i-1)}$
(9)      End do
(10) End do
(11) Let $z_i := z_i^{(i-1)}$, $w_i := w_i^{(i-1)}$ and $p_i := p_i^{(i-1)}$, for $1 \le i \le n$. Return
     $Z = [z_1, z_2, \ldots, z_n]$, $W = [w_1, w_2, \ldots, w_n]$ and $D = \mathrm{diag}(p_1, p_2, \ldots, p_n)$.
```

This algorithm can be interpreted as a (two sided) generalized Gram–Schmidt orthogonalization process with respect to the bilinear form associated with A. If A is SPD, only the process for Z needs to be carried out (since in this case $W = Z$), and the algorithm is just a conjugate Gram–Schmidt process, i.e., orthogonalization of the unit vectors with respect to the "energy" inner product $\langle x, y\rangle := x^T Ay$.

It is easy to see that, in exact arithmetic, the above process can be completed without divisions by zero if and only if all the leading principal minors of A are nonzeros or, equivalently, if and only if A has an LU factorization [40]. In this case, if $A = LDU$ is the decomposition of A as a product of a unit lower triangular matrix L, a diagonal matrix D, and a unit upper triangular matrix U, it is easily checked that $Z = U^{-1}$ and $W = L^{-T}$ (D being exactly the same matrix in both factorizations). Because of the initialization chosen (step (1) in Algorithm 5.2), the z- and w-vectors are initially very sparse; however, the updates in step (8) cause them to fill in rapidly (see [40, 43, 64] for graph-theoretical characterizations of fill-in in Algorithm 5.2). Sparsity in the inverse factors is preserved by carrying out the biconjugation process incompletely, similar to ILU-type methods. Incompleteness can be enforced either on the basis of position, allowing the vectors $\bar{z}_i$ and $\bar{w}_i$ to have nonzero entries only in prescribed locations, or on the basis of a drop tolerance, whereby new fill-ins are removed if their magnitude is less than a prescribed threshold $\tau > 0$. Not surprisingly, the second strategy is much more robust and effective, particularly for unstructured problems (see [35, 42]). Because of incompleteness, the question arises as to whether the preconditioner construction can be performed without breakdowns (divisions by zero): we discuss this in Section 5.2. For SPD problems, there exists a variant of AINV, denoted SAINV (for stabilized AINV), that is breakdown free [32, 191].

The AINV/SAINV preconditioner has been extensively tested on a variety of symmetric and nonsymmetric problems in conjunction with standard Krylov subspace methods such as conjugate gradients (for symmetric positive definite matrices) and GMRES, Bi-CGSTAB, and TFQMR [146] (for nonsymmetric problems). The preconditioner has been found to be comparable to ILU methods in terms of robustness and rates of convergence, with ILU methods being somewhat faster on average on sequential computers (see [38, 40–42]).

Although the application of the AINV preconditioner on a parallel computer, as it consists of matrix–vector products, is easily parallelized, the incomplete A-biorthogonalization process used to construct the preconditioner is inherently sequential. One possible solution to this problem, adopted in [46], is to compute the preconditioner sequentially on one processor and then to distribute the approximate inverse factors among processors in a way that minimizes communication costs while achieving good load balance. This approach is justified in applications, such as those considered in [46], in which the matrices are small enough to fit on the local memory of one processor, and where the preconditioner can be reused a number of times. In this case the time for computing the preconditioner is negligible relative to the overall costs. When solving very large problems, however, it is desirable to be able to compute the preconditioner in parallel.

As shown in [37, 45], a fully parallel AINV algorithm can be achieved by means of graph partitioning. The idea can be illustrated as follows. If p processors are available, graph partitioning can be used to decompose the adjacency graph associated with the sparse matrix A (or with $A + A^T$ if A is not structurally symmetric) in p subgraphs of roughly equal size in such a way that the number of edge cuts is approximately minimized. Nodes which are connected by cut edges are removed from the subgraphs and put in the *separator set*. By numbering the nodes in the separator set last, a symmetric permutation P^TAP of A is obtained. The permuted matrix has the following structure:

$$P^TAP = \begin{pmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \cdots & C_p & A_S \end{pmatrix}.$$

The diagonal blocks $A_1, \dots, A_p$ correspond to the interior nodes in the graph decomposition and should have approximately the same order. The off-diagonal blocks B_i, C_i represent the connections between the subgraphs, and the diagonal block A_S the connections between nodes in the separator set. The order of A_S is equal to the cardinality of the separator set and should be kept as small as possible.

Let $P^TAP = LDU$ be the LDU factorization of P^TAP (which we assume to exist). For each diagonal block A_i, let $A_i = L_iD_iU_i$ be the corresponding LDU factorization. Then it is easy to see that

$$L^{-1} = \begin{pmatrix} L_1^{-1} & & & & \\ & L_2^{-1} & & & \\ & & \ddots & & \\ & & & L_p^{-1} & \\ F_1 & F_2 & \dots & F_p & L_S^{-1} \end{pmatrix},$$

where $F_i := -L_S^{-1}C_iA_i^{-1}(1 \le i \le p)$ and L_S^{-1} is the inverse of the unit lower triangular factor of the *Schur complement* matrix

$$S = A_S - \sum_{i=1}^{p} C_iA_i^{-1}B_i.$$

We are assuming here that S has the LDU factorization $S = L_SD_SU_S$. Likewise,

$$U^{-1} = \begin{pmatrix} U_1^{-1} & & & & E_1 \\ & U_2^{-1} & & & E_2 \\ & & \ddots & & \vdots \\ & & & U_p^{-1} & E_p \\ & & & & U_s^{-1} \end{pmatrix},$$

where $E_i := -A_i^{-1} B_i U_S^{-1} (1 \le i \le p)$ and U_S^{-1} is the inverse of the unit upper triangular factor of S. It is important to observe that L^{-1} and U^{-1} preserve a good deal of sparsity, since fill-in can occur only within the nonzero blocks. The matrix D is simply defined as

$$D = \operatorname{diag}(D_1, D_2, \ldots, D_p, D_S).$$

Hence, we can write the (generally dense) inverse $(P^T AP)^{-1}$ of $P^T AP$ as a product of sparse matrices L^{-1}, U^{-1} and D^{-1}. In practice, however, the inverse factors L^{-1} and U^{-1} contain too many nonzeros. Since we are only interested in computing a preconditioner, we just need to compute sparse approximations to L^{-1} and U^{-1}.

This is accomplished as follows. With graph partitioning, the matrix is distributed so that processor P_i holds A_i, C_i, and $B_i (1 \le i \le p)$. One of the processors, marked P_S, should also hold A_S. Each processor then computes sparse approximate inverse factors $\bar{Z}_i$, $\bar{D}_i$, and $\bar{W}_i$ such that $\bar{Z}_i \bar{D}_i^{-1} \bar{W}_i^T \approx A_i^{-1}$ using the AINV algorithm. Once this is done, each processor computes the product $S_i := C_i \bar{Z}_i \bar{D}_i^{-1} \bar{W}_i^T B_i \approx C_i A_i^{-1} B_i$. Until this point the computation proceeds in parallel with no communication. The next step is the accumulation of the (approximate) Schur complement $\hat{S} := A_S - \sum_{i=1}^{p} S_i$ on processor P_S. This accumulation is done in $k = \log_2 p$ steps with a fan-in across the processors.

As soon as $\hat{S}$ is computed, processor P_S computes a factored sparse approximate inverse $\bar{Z}_S \bar{D}_S^{-1} \bar{W}_S^T \approx \hat{S}^{-1}$ using the AINV algorithm. This is a sequential bottleneck and explains why the size of the separator set must be kept small. Once the approximate inverse factors of $\hat{S}$ are computed, they are broadcast to all remaining processors. (Actually, the preconditioner application can be implemented in such a way that only the $\bar{W}_S$ factor needs to be broadcast.) Notice that because only matrix–vector products are required in the application of the preconditioner, there is no need to form $-\bar{Z}_i \bar{D}_i^{-1} \bar{W}_i^T B_i \bar{Z}_S \approx E_i$ or $-\bar{W}_S^T C_i \bar{Z}_i \bar{D}_i^{-1} \bar{W}_i^T \approx F_i$ explicitly. In this way, a factored sparse approximate inverse of $P^T AP$ is obtained.

Note that this is a two-level preconditioner, in the sense that the computation of the preconditioner involves two phases. In the first phase, sparse approximate inverses of the diagonal blocks A_i are computed. In the second phase, a sparse approximate inverse of the approximate Schur complement $\hat{S}$ is computed. Without this second step the preconditioner would reduce to a block Jacobi method with inexact block solves (in the terminology of domain decomposition methods, this is additive Schwarz with inexact solves and no overlap). As already mentioned, for a fixed problem size, the rate of convergence of this preconditioner tends to deteriorate as the number of blocks (subdomains) grows. Hence, assuming that each block is assigned to a processor in a parallel computer, this method would not be scalable. However, the approximate Schur complement phase provides a global exchange of information across the processors, acting as a "coarse grid" correction in which the "coarse grid" nodes are interface nodes (i.e., they correspond to vertices in the separator set). As we will see, this prevents the number of iterations from growing as the number of processors grows. As long as the cardinality of the separator set is small compared to the cardinality of the subdomains (subgraphs), the algorithm is scalable in terms of parallel efficiency. Indeed, in this case the application of the preconditioner at each step of a Krylov subspace method such as GMRES or Bi-CGSTAB is easily implemented in parallel with relativeley little communication needed.

As a fairly typical example of the type of performance achieved by the parallel AINV preconditioner, we report some results obtained with the Fortran/MPI implementation[7]

[7] The code can be obtained by contacting the author.

TABLE VII
Parallel AINV Results for Neutron Diffusion Problem

p	2	4	8	16	32	64
P-time	11.6	5.89	3.24	1.79	1.12	0.94
It-time	227.1	113.3	56.9	26.7	13.6	11.7
Its	157	157	156	157	156	157

first described in [37]. As a first example we consider a linear system with $n = 804{,}609$ unknowns provided by James Warsa (Transport Methods Group, Los Alamos National Laboratory.) This problem arises from the finite element discretization of neutron diffusion in a water–iron reactor (in 2D) and is a widely used benchmark in nuclear engineering. In Table VII we report the number of PCG iterations (under "Its") and corresponding preconditioner setup times (under "P-time") and iteration times (under "It-time"). The results were obtained on a SGI Origin 2000 system using $p = 2, 4, \ldots, 64$ processors. Timings are in seconds.

We have solved the same problem using an efficient parallel PCG code with AMG preconditioning developed by Wayne Joubert at Los Alamos. We have found that for this problem, AINV is significantly faster. For example, the AMG solve time for $p = 32$ processors was found to be 39 s, of which approximately 30 s were spent in the setup phase. In contrast, the setup times for AINV are very small. This problem is not large enough for the high setup costs of AMG to be amortized, unless the preconditioner can be reused over repeated solves.

As a second example, we show results (Table VIII) for a problem arising in ocean modeling (barotropic equation) with $n = 370{,}000$ unknowns and approximately 3.3 million nonzero entries provided by John Dukowicz (also of LANL). For this problem, several multigrid methods have been tried without much success, but sparse approximate inverses appear to be effective (see also [265]).

AINV preconditioning has been used and further developed by several researchers (see, for example, [46, 47, 54, 64, 66, 67, 191, 217–219]). For applications to quantum chemistry (linear scaling electronic structure calculations), see Challacombe [78, 79].

5.2. *Existence and Stability Issues*

Regardless of whether the sparsity pattern is fixed or adaptively determined, the SPAI algorithm can always be carried to completion, in the sense that no breakdown can occur in the course of its construction, as long as the coefficient matrix A is nonsingular (see [89] for an application of SPAI to a class of singular linear systems). In principle, the resulting preconditioner could be singular for certain choices of the parameters; however, this is

TABLE VIII
Parallel AINV Results for Barotropic Equation

p	2	4	8	16	32
P-time	13.4	7.0	3.72	1.91	1.15
It-time	119.2	59.3	25.9	11.9	6.8
Its	189	189	189	202	187

unlikely to happen in practice. The application of the preconditioner itself, requiring only matrix–vector products, is a stable operation. Hence, no instability of the kind that occasionally destroys the effectiveness of ILU-type techniques can occur, and it has been observed that SPAI is able to solve some difficult problems for which standard ILU techniques fail (see, e.g., [23]).

Factored approximate inverses, on the other hand, are not always well defined and may be subject to breakdown in the construction phase. Existence and stability can be established for certain classes of matrices. The FSAI preconditioner is always well defined and can be computed in a stable way, if A is SPD, but may fail to exist for more general matrices [196].

The existence (and stability) theory for the AINV preconditioner is similar to that for ILU-type methods. In [38] it is proved that a sufficient condition is that A be an H-matrix, similar to ILU. It should be kept in mind, however, that the applicability of AINV goes well beyond these nice classes of problems, as is the case for ILU-type methods. In the general case, diagonal modifications may be necessary. However, it has been shown in [33, 54] that static and dynamic pivoting strategies can be used to prevent breakdowns, yielding reliable preconditioners even for very difficult problems.

The AINV algorithm may fail (due to pivot breakdowns) for a general SPD matrix. Such breakdowns have been observed in problems from structural analysis [35], which are SPD but far from having the M-matrix property. Indeed, such matrices often have fairly large positive off-diagonal entries. In this case, negative pivots may occur, and the preconditioner fails to be SPD. Replacing these negative pivots, which can be large in absolute value, by arbitrary positive quantities leads to poor preconditioners. Fortunately, there is a simple modification of the original AINV algorithm that can be shown to be completely breakdown free; this is the SAINV (for stabilized AINV) algorithm developed independently in [32, 191]. This algorithm is based on the A-orthogonalization process (8) described in Section 3.2.1.

5.3. *The Effect of Ordering*

Approximate inverse preconditioners in nonfactored form (like SPAI) are largely insensitive to the orderings of the equations and unknowns (see [43, 163]). This is because the inverse of the matrix $P^T AP$ obtained by a permutation of the rows and columns of A is just $P^T A^{-1} P$. Hence, permutations do not affect the number or the size of nonzero entries in A^{-1}. On the other hand, factored sparse approximate inverses are sensitive to the ordering of the equations and unknowns. Indeed, for a sparse matrix A the amount of *inverse fill*, which is defined as the number of structurally nonzero entries in the inverse triangular factors, is strongly dependent on the ordering of A. Also, a factored (approximate) inverse may or may not exist, depending on the ordering of A.

Detailed studies of the effect of ordering on the AINV preconditioner have been presented in [43, 64], while a comparison of different orderings used with the FSAI algorithm can be found in [141]. Using basic results from graph theory [153], it can be shown that orderings like minimum degree and generalized nested dissection tend to minimize the amount of inverse fill. Conversely, bandwidth-reducing orderings, such as reverse Cuthill–McKee, tend to produce inverse factors that are completely dense. Therefore, it can be expected that minimum degree or nested dissection should reduce the time needed for computing a sparse approximate inverse preconditioner. However, the magnitude of the entries in the inverse factors is also impacted by the ordering. Fill-reducing orderings produce entries in the

TABLE IX
AINV Results for Different Orderings, Problem UTM3060

Ordering	NO	RCM	MMD
Its	1106	412	116
Density	9.56	8.96	2.81
P-time	3.10	2.26	0.80
It-time	25.5	10.1	2.08

inverse factors that are larger, on average, than those corresponding to bandwidth-reducing orderings [43]. Discarding many such entries could result in poor rates of convergence, as is often the case with ILU-type methods (compare the results in Table IV). Extensive numerical experiments in [43, 64] show that this is not the case, and that orderings like minimum degree perform very well in practice, much better than RCM. This is good news also from the standpoint of parallelizing the AINV preconditioner construction phase. Indeed, fill-reducing orderings like MMD and nested dissection result in short and wide elimination trees, which are well-suited for parallel execution [37, 43, 64].

In Table IX we show some results for matrix UTM3060 from the Matrix Market. This is a rather ill-conditioned nonsymmetric matrix from plasma physics. It has order $n = 3060$ and it contains $nnz = 42,211$ nonzeros. Bi-CGSTAB with no preconditioning requires 4033 iterations to reduce the intitial residual by eight orders of magnitude; this takes 10.1 s on one processor of a Sun Starfire server. Preconditioning with the diagonal part of A makes things even worse (no convergence in 10,000 iterations). In the table we report results for AINV preconditioning with drop tolerance $\tau = 0.1$ for the original ordering (NO), multiple minimum degree (MMD), and reverse Cuthill–McKee (RCM). As usual, the orderings are computed using the graph of the symmetrized matrix $A + A^T$. The original and RCM orderings suffer from slow decay in the inverse factors, and it is clear that minimum degree is by far the best ordering among the ones tried. These results are fairly typical (see [43, 64] for extensive experiments).

In Fig. 4 we show the sparsity patterns for the UTM3060 matrix permuted by RCM and MMD, as well as the sparsity patterns of the incomplete inverse factors. For the latter, the lower triangular part corresponds to W^T and the upper triangular to Z.

For certain problems, for example those characterized by strong anisotropies, purely graph-based orderings may fail to capture important features of the problem. Better results can be obtained by taking into account the magnitude of the entries. Some promising weighted graph heuristics have been proposed in [64], but much work remains to be done.

In the case of highly nonsymmetric and indefinite problems, it was shown in [33] that the robustness and performance of approximate inverse preconditioners can be greatly enhanced by the nonsymmetric permutations and scalings implemented by Duff and Koster in the Harwell Subroutine Library code MC64 (see Section 3.3). This preprocessing makes the coefficient matrix more diagonally dominant and better conditioned, and the preprocessed matrix can be effectively preconditioned by techniques like AINV.

5.4. *Block Algorithms*

Block variants of sparse approximate inverse preconditioners have been studied by several authors. Challacombe [78, 79] has used a physics-based blocking of the matrix entries to

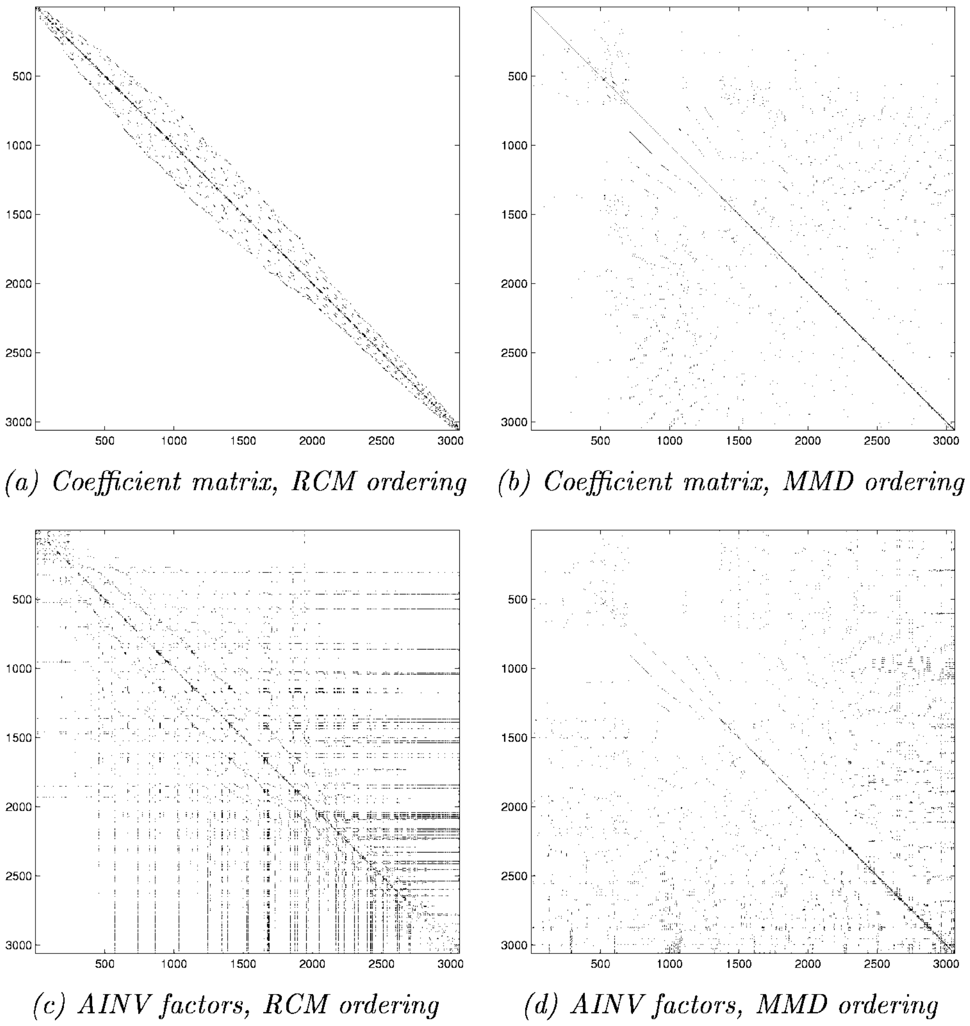


FIG. 4. Patterns for matrix UTM3060 and its AINV factors with different orderings.

achieve high performance when applying the AINV transformation in quantum chemistry applications; however, the approximate inverse itself was computed using the point version of AINV (Algorithm 5.2). Block variants of the AINV and SAINV preconditioners have been introduced and studied in [36], where they were used to solve tough problems in solid and structural mechanics. A breakdown-free block variant of AINV was also developed by Bridson and Tang [66].

A block variant of the SPAI preconditioner has been recently described [25]. It should be noticed that the effect of blocking is quite different for different preconditioners. For instance, for the case of the SPAI preconditioner, blocking has the effect of reducing the time to compute the preconditioner compared to the unblocked version, but at the cost of slower convergence rates. As explained in [25], this is due to the fact that the dropping strategy used in the block SPAI algorithm is less stringent, and thus less effective, than the one used in the scalar version of SPAI. This is not the case for AINV, for which the block version is typically faster than the point one, at least for problems that are endowed with a natural block structure. In all cases, however, blocking results in improved cache efficiency on hierarchical memory computers.

As an example, we describe the block AINV method of [36] and its stabilized variant (block SAINV). Assume that the system matrix A is SPD and has been partitioned in the block form (12). Let $m_i = \sum_{j<i} n_j$ be the offset of the ith block. Also, denote the block rows of A by A_i^T, $i = 1, \ldots, p$. That is,

$$A_i^T = [A_{i1}, \ldots, A_{ip}].$$

Note that the diagonal blocks A_{ii} are square symmetric positive definite matrices, and that $A_{ji} = A_{ij}^T$ for $i \neq j$.

The block AINV algorithm computes block partitioned Z and D directly from A based on a block A-orthogonalization process applied to blocks of columns of the identity matrix. Note that D is now block diagonal. Let E_i denote the $n \times n_i$ matrix with all zero rows except for rows $m_i + 1$ through $m_i + n_i$, which correspond to the rows of the $n_i \times n_i$ identity matrix I_{n_i}. The block A-orthogonalization procedure can be written as follows.

ALGORITHM 5.3. Block A-orthogonalization algorithm:

```
Let $Z_i^{(0)} = E_i \quad (1 \le i \le p)$;   $P_1^{(0)} = A_{11}$
For $i = 2, \ldots, p$ do
    For $j = 1, \ldots, i-1$ do
        $P_i^{(j-1)} := A_j^T Z_i^{(j-1)}$
        $Z_i^{(j)} = Z_i^{(j-1)} - Z_j^{(j-1)} \left[P_j^{(j-1)}\right]^{-1} P_i^{(j-1)}$
    End do
    $P_i^{(i-1)} = A_i^T Z_i^{(i-1)}$
End do
Let $Z_i := Z_i^{(i-1)}$ and $D_i := P_i^{(i-1)}$, for $1 \le i \le p$. Return
$Z = [Z_1, Z_2, \ldots, Z_p]$ and $D = \mathrm{diag}(D_1, D_2, \ldots, D_p)$.
```

The block AINV preconditioner $M = ZD^{-1}Z^T \approx A^{-1}$ is computed by applying this block generalized Gram–Schmidt process incompletely. Blocks with small norms (corresponding to positions above the block diagonal part of the triangular matrix Z) are dropped to preserve sparsity after the block updates for $Z_i^{(j)}$.

The inverses of the pivot blocks appearing in the update step for $Z_i^{(j)}$ above can be computed by means of a full triangular factorization. Notice that because of dropping, the pivot blocks in the incomplete process may not be positive definite, or even symmetric. Nevertheless, a stabilized and fully reliable version of the algorithm can be obtained from Algorithm 5.3 by replacing the expressions for $P_i^{(j-1)}$ and $P_i^{(i-1)}$ with $[Z_j^{(j-1)}]^T A Z_i^{(j-1)}$ and $[Z_i^{(i-1)}]^T A Z_i^{(i-1)}$, respectively. The pivot blocks are now guaranteed to be symmetric and positive definite, at the price of a somewhat higher construction cost.

Notice that the block AINV algorithms are rich in dense matrix–matrix operations; hence, BLAS-3 kernels can be used to attain high performance.

Computational experience indicates that the performance of the block AINV schemes can be significantly enhanced by explicitly applying a symmetric block Jacobi scaling to the coefficient matrix A prior to performing the block AINV process. This preprocessing is based on the computation of the Cholesky factorizations $A_{ii} = L_i L_i^T$ for $1 \le i \le p$,

TABLE X
Block AINV Results for S3RMQ4M1, Different Scalings

Preconditioner	Scaling	τ	Density	Iter	P-time	I-time	P + I
Block AINV	—	0.02	0.74	827	0.4	5.9	6.3
	—	0.01	2.09	506	1.0	5.8	6.8
	—	0.003	4.75	262	2.5	5.5	8.0
Block AINV	Jacobi	0.25	1.06	672	0.6	5.0	5.6
	Jacobi	0.15	2.53	370	1.2	4.8	6.0
	Jacobi	0.075	4.53	286	2.4	6.7	9.1
Block AINV	Block Jacobi	0.1	1.21	408	0.7	3.3	4.0
	Block Jacobi	0.05	1.75	320	0.7	3.3	4.0
	Block Jacobi	0.02	3.73	189	1.8	3.1	4.9

followed by the explicit formation of the block scaled matrix

$$\hat{A} = G^{-1}AG^{-T}, \quad \text{where } G = \operatorname{diag}(L_1, \ldots, L_p).$$

The additional costs incurred by this scaling are usually small compared to the total solution costs.

Table X reports results for the block AINV algorithm with different scalings applied to the matrix S3RMQ4M1, a very ill-conditioned thin shell problem (see [32, 35]). Ashcraft's graph compression algorithm [8] was used to identify the block structure, and the compressed graph was reordered with multiple minimum degree. In the table, "P-time" denotes the preconditioner setup time, "I-time" the PCG iteration time, and "P + I" the total solution time. All timings are in seconds. Also, τ denotes the different values of the drop tolerance used; "density" denotes the relative fill in the approximate inverse preconditioner (that is, the total number of nonzeros in the Z and D matrices, divided by the number of nonzeros in the lower triangular part of A), and "iter" is the number of PCG iterations. For this linear system, the block AINV preconditioner with block Jacobi scaling is approximately four times faster in terms of solution time than the point-stabilized AINV algorithm with point Jacobi scaling (not shown). The results in Table X show the benefit of using block diagonal scaling.

It should be mentioned that block variants of AINV (or SAINV) preconditioning appear to be superior to their point counterparts only for problems that possess a natural block structure, such as multicomponent problems in structural mechanics. Artificially imposing a block structure on, say, a scalar problem usually results in disappointing performance. Indeed, while the number of iterations and even the time per iteration may go down with a block algorithm, using blocks which are not completely dense introduces additional arithmetic overhead that tends to offset the gains in convergence rates and flops rate.

6. ALGEBRAIC MULTILEVEL VARIANTS

For a long time, the development of multilevel methods has taken place in parallel with and largely independently of that of general-purpose Krylov subspace methods and preconditioning techniques. This unfortunate situation has finally changed in the last few years. On one hand, it has become clear that the robustness of multigrid-like methods can be enhanced by coupling them with CG or other Krylov subspace acceleration techniques. Also, incomplete factorization preconditioners have been shown to provide good smoothers for multigrid (see, e.g., [290]). Recent studies [69, 70, 270] have shown that sparse approximate inverses can also provide effective and robust smoothers for both geometric and algebraic multigrid, especially for tough problems (see also [19, 29, 30, 234] for early work in this direction). Conversely, the ever-increasing size of the linear systems arising in countless applications has exposed the limitations of nonscalable methods (such as incomplete factorization and sparse approximate inverse preconditioners), and it is now generally agreed that the multilevel paradigm must somehow be incorporated into these techniques in order for these to remain competitive.

On the mathematical level, the difference between incomplete factorizations and methods like AMG is not as deep as it might appear at first sight, as all these algorithms can be interpreted as approximate Schur complement methods [16, 17, 104, 284]. This realization has prompted the development of a number of algebraic multilevel algorithms which are rooted in standard ILU or approximate inverse techniques but somehow attempt to achieve algorithmic scalability by mimicking multigrid methods.

There is at present a wide variety of such algorithms; an incomplete list of recent papers includes [20–22, 58, 113, 203, 228, 235, 242, 250, 254, 256, 257, 276, 295] for multilevel variants of ILU, and [53, 218, 219, 227] for multilevel sparse approximate inverse techniques. Combining approximate inverses and wavelet transforms is another way to improve scalability (see [67, 80, 99]). At present there does not seem to be any method that is clearly best in terms of performance and robustness, but the multilevel variants are generally superior to their one-level counterparts. Research on this class of techniques is continuing at an accelerated pace, leading to increasingly robust and powerful algorithms.

The development of efficient parallel versions of AMG and other algebraic multilevel methods has proven to be difficult due to the fact that the coarsening strategy used in the original AMG algorithm is highly sequential in nature. Recent papers [170, 198] show that progress is being made in this direction as well, but much work remains to be done, especially for indefinite problems, for problems in 3D, and for systems of partial differential equations.

Finally we mention the powerful FETI method, a family of scalable (or nearly scalable) algorithms that are being used to solve huge problems in structural mechanics and other applications of finite element analysis on massively parallel computers (see [135–137] and numerous subsequent papers by Farhat and collaborators). The FETI methods make use of knowledge of the underlying application to achieve optimality and cannot be considered purely algebraic methods, but the basic idea is general enough that it can be adapted to solve a wide variety of problems.

7. CONCLUSIONS

The development of efficient and reliable preconditioned iterative methods is the key for the successful application of scientific computation to the solution of many large-scale problems. Therefore it is not surprising that this research area continues to see a vigorous level of activity.

In this survey, we have attempted to highlight some of the developments that have taken place in recent years. Among the most important such developments, we emphasize the following:

- Improved robustness, frequently achieved by transferring techniques from sparse direct solvers (such as reorderings and scalings) to preconditioners.
- Improved performance through the use of blocking.
- Fully parallel versions of incomplete factorization methods with convergence rates equal to those of the standard, sequential versions.
- The emergence of a new class of general-purpose, parallel preconditioners based on sparse approximate inverses.
- The cross-fertilization between multilevel methods on one hand and incomplete factorization and sparse approximate inverse techniques on the other.
- The increasingly frequent use of iterative solvers in industrial applications.

There are many further important problems and ideas that we have not been able to address in this article, even within the relatively narrow context of purely algebraic methods. For example, nothing has been said about preconditioning techniques for the special linear systems arising in the solution of eigenvalue problems [192, 221, 262, 292]. Also, we have not been able to include a discussion of certain graph theoretically motivated techniques that show promise and have been given increased attention, known as Vaidya-type and support graph preconditioners [49, 56, 83]. Furthermore, we have hardly touched on the very important topic of software for preconditioned iterative methods.

In concluding this survey we stress the fact that in spite of recent progress, there are still important areas where much work remains to be done. While efficient preconditioners have been developed for certain classes of problems, such as self-adjoint, second-order scalar elliptic PDEs with positive definite operator, much work remains to be done for more complicated problems. For example, there is a need for reliable and efficient preconditioners for symmetric indefinite problems [223].

Ideally, an optimal preconditioner for problem (1) would result in an $\mathcal{O}(n)$ solution algorithm, would be perfectly scalable when implemented on a parallel computer, and would behave robustly over large problem classes. However, barring some unforeseen breakthrough, such a "Holy Grail" solver is unlikely to be around the corner; chances are that such a powerful and general algorithm will never be found [291]. On the other hand, incremental progress leading to increasingly more powerful and reliable preconditioners for specific types of problems is within reach and can be expected to continue for the foreseeable future.

ACKNOWLEDGMENTS

I thank Jerry Brackbill for inviting me to write this survey. Without his encouragement and great patience, this article would have never been completed. I am also greatly indebted to Miroslav Tůma, whose skillful implementation of many of the algorithms discussed in this paper is gratefully acknowledged. Finally, thanks to one of the anonymous referees for his (or her) careful reading of the manuscript.

REFERENCES

1. C. H. Ahn, W. C. Chew, J. S. Zhao, and E. Michielssen, Numerical study of approximate inverse preconditioner for two-dimensional engine inlet problems, *Electromagnetics* **19**, 131 (1999).
2. M. A. Ajiz and A. Jennings, A robust incomplete Choleski-conjugate gradient algorithm, *Int. J. Numer. Methods Eng.* **20**, 949 (1984).

3. G. Alléon, M. Benzi, and L. Giraud, Sparse approximate inverse preconditioning for dense linear systems arising in computational electromagnetics, *Numer. Algorithms* **16**, 1 (1997).
4. P. Amestoy, T. A. Davis, and I. S. Duff, An approximate minimum degree ordering algorithm, *SIAM J. Matrix Anal. Appl.* **17**, 886 (1996).
5. P. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, *SIAM J. Matrix Anal. Appl.* **23**, 15 (2001).
6. S. F. Ashby, P. N. Brown, M. R. Dorr, and A. C. Hindmarsh, A linear algebraic analysis of diffusion synthetic acceleration for the Boltzmann transport equation, *SIAM J. Numer. Anal.* **32**, 179 (1995).
7. S. F. Ashby, T. A. Manteuffel, and P. E. Saylor, Adaptive polynomial preconditioning for hermitian indefinite linear systems, *BIT* **29**, 583 (1989).
8. C. Ashcraft, Compressed graphs and the minimum degree algorithm, *SIAM J. Sci. Comput.* **16**, 1404 (1995).
9. O. Axelsson, A generalized SSOR method, *BIT* **13**, 443 (1972).
10. O. Axelsson, A survey of preconditioned iterative methods for linear systems of algebraic equations, *BIT* **25**, 166 (1985).
11. O. Axelsson, A general incomplete block matrix factorization method, *Linear Algebra Its Appl.* **74**, 179 (1986).
12. O. Axelsson, *Iterative Solution Methods* (Cambridge Univ. Press, Cambridge, 1994).
13. O. Axelsson, S. Brinkkemper, and V. P. Il'in, On some versions of incomplete block matrix factorization iterative methods, *Linear Algebra Its Appl.* **58**, 3 (1984).
14. O. Axelsson and V. Eijkhout, Vectorizable preconditioners for elliptic difference equations in three space dimensions, *J. Comput. Appl. Math.* **27**, 299 (1991).
15. O. Axelsson and L. Yu. Kolotilina, Diagonally compensated reduction and related preconditioning methods, *Numer. Linear Algebra Appl.* **1**, 155 (1994).
16. O. Axelsson and P. S. Vassilevski, Algebraic multilevel preconditioning methods, I, *Numer. Math.* **56**, 157 (1989).
17. O. Axelsson and P. S. Vassilevski, Algebraic multilevel preconditioning methods, II, *SIAM J. Numer. Anal.* **27**, 1569 (1990).
18. S. Balay, K. Buschelman, W. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. Smith, and H. Zhang, *PETSc User's Manual*, Report ANL 95/11, Revision 2.2.3 (Argonne National Laboratory, Argonne, IL, 2002).
19. R. N. Banerjee and M. W. Benson, An approximate inverse based multigrid approach to the biharmonic problem, *Int. J. Comput. Math.* **40**, 201 (1991).
20. R. E. Bank and R. K. Smith, The incomplete factorization multigraph algorithm, *SIAM J. Sci. Comput.* **20**, 1349 (1999).
21. R. E. Bank and R. K. Smith, An algebraic multilevel multigraph algorithm, *SIAM J. Sci. Comput.* **23**, 1572 (2002).
22. R. E. Bank and C. Wagner, Multilevel ILU decomposition, *Numer. Math.* **82**, 543 (1999).
23. S. T. Barnard, L. M. Bernardo, and H. D. Simon, An MPI implementation of the SPAI preconditioner on the T3E, *Int. J. High Perform. Comput. Appl.* **13**, 107 (1999).
24. S. T. Barnard and R. L. Clay, A portable MPI implementation of the SPAI preconditioner in ISIS++, in *Proceedings of the Eight SIAM Conference on Parallel Processing for Scientific Computing*, edited by M. Heath *et al.* (SIAM, Philadelphia, PA, 1997 [CD-ROM]).
25. S. T. Barnard and M. J. Grote, A block version of the SPAI preconditioner, in *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, edited by B. A. Hendrickson *et al.* (SIAM, Philadelphia, PA, 1999 [CD-ROM]).
26. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods* (SIAM, Philadelphia, PA, 1994).
27. G. A. Behie and P. A. Forsyth, Comparison of fast iterative methods for symmetric systems, *IMA J. Numer. Anal.* **3**, 41 (1983).

28. M. W. Benson, *Iterative Solution of Large Scale Linear Systems*, M.Sc. thesis (Lakehead University, Thunder Bay, Ontario, 1973).

29. M. W. Benson, Frequency domain behavior of a set of parallel multigrid smoothing operators, *Int. J. Comput. Math.* **36**, 77 (1990).

30. M. W. Benson, An approximate inverse based multigrid approach to thin domain problems, *Util. Math.* **45**, 39 (1994).

31. M. W. Benson and P. O. Frederickson, Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems, *Util. Math.* **22**, 127 (1982).

32. M. Benzi, J. K. Cullum, and M. Tůma, Robust approximate inverse preconditioning for the conjugate gradient method, *SIAM J. Sci. Comput.* **22**, 1318 (2000).

33. M. Benzi, J. C. Haws, and M. Tůma, Preconditioning highly indefinite and nonsymmetric matrices, *SIAM J. Sci. Comput.* **22**, 1333 (2000).

34. M. Benzi, W. D. Joubert, and G. Mateescu, Numerical experiments with parallel orderings for ILU preconditioners, *Electron. Trans. Numer. Anal.* **8**, 88 (1999).

35. M. Benzi, R. Kouhia, and M. Tůma, An assessment of some preconditioning techniques in shell problems, *Commun. Numer. Methods Eng.* **14**, 897 (1998).

36. M. Benzi, R. Kouhia, and M. Tůma, Stabilized and block approximate inverse preconditioners for problems in solid and structural mechanics, *Comput. Methods Appl. Mech. Eng.* **190**, 6533 (2001).

37. M. Benzi, J. Marín, and M. Tůma, A two-level parallel preconditioner based on sparse approximate inverses, in *Iterative Methods in Scientific Computation IV*, edited by D. R. Kincaid and A. C. Elster, IMACS Series in Computational and Applied Mathematics, Vol. 5, p. 167. (IMACS, New Brunswick, NJ, 1999).

38. M. Benzi, C. D. Meyer, and M. Tůma, A sparse approximate inverse preconditioner for the conjugate gradient method, *SIAM J. Sci. Comput.* **17**, 1135 (1996).

39. M. Benzi, D. B. Szyld, and A. van Duin, Orderings for incomplete factorization preconditioning of nonsymmetric problems, *SIAM J. Sci. Comput.* **20**, 1652 (1999).

40. M. Benzi and M. Tůma, A sparse approximate inverse preconditioner for nonsymmetric linear systems, *SIAM J. Sci. Comput.* **19**, 968 (1998).

41. M. Benzi and M. Tůma, Numerical experiments with two approximate inverse preconditioners, *BIT* **38**, 234 (1998).

42. M. Benzi and M. Tůma, A comparative study of sparse approximate inverse preconditioners, *Appl. Numer. Math.* **30**, 305 (1999).

43. M. Benzi and M. Tůma, Orderings for factorized approximate inverse preconditioners, *SIAM J. Sci. Comput.* **21**, 1851 (2000).

44. M. Benzi and M. Tůma, A robust incomplete factorization preconditioner for positive definite matrices, *Numer. Linear Algebra Appl.*, in press.

45. M. Benzi and M. Tůma, A parallel solver for large-scale Markov chains, *Appl. Numer. Math.* **41**, 135 (2002).

46. L. Bergamaschi, G. Pini, and F. Sartoretto, Approximate inverse preconditioning in the parallel solution of sparse eigenproblems, *Numer. Linear Algebra Appl.* **7**, 99 (2000).

47. L. Bergamaschi, G. Pini, and F. Sartoretto, Parallel preconditioning of a sparse eigensolver, *Parallel Comput.* **27**, 963 (2001).

48. A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, 3rd ed. (Academic Press, New York, 1979; reprinted by SIAM, Philadelphia, PA, 1994).

49. M. Bern, J. R. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo, Support graph preconditioners, submitted for publication.

50. E. Bodewig, *Matrix Calculus*, 2nd ed. (Interscience, New York/North-Holland, Amsterdam, 1959).

51. M. Bollhöfer, A robust ILU with pivoting based on monitoring the growth of the inverse factors, *Linear Algebra Its Appl.* **338**, 201 (2001).

52. M. Bollhöfer, A robust and efficient ILU that incorporates the growth of the inverse triangular factors, submitted for publication.

53. M. Bollhöfer and V. Mehrmann, Algebraic multilevel methods and sparse approximate inverses, *SIAM J. Matrix Anal. Appl.* **24**, 191 (2002).

54. M. Bollhöfer and Y. Saad, A factored approximate inverse preconditioner with pivoting, *SIAM J. Matrix Anal. Appl.* **23**, 692 (2002).

55. M. Bollhöfer and Y. Saad, On the relations between ILUs and factored approximate inverses, *SIAM J. Matrix Anal. Appl.* **24**, 219 (2002).

56. E. G. Boman, D. Chen, B. Hendrickson, and S. Toledo, Maximum-weight-basis preconditioners, submitted for publication.

57. Y. Y. Botros and J. L. Volakis, Preconditioned generalized minimal residual iterative scheme for perfectly matched layer terminated applications, *IEEE Microwave Guided Wave Lett.* **9**, 45 (1999).

58. E. F. F. Botta and F. W. Wubs, Matrix renumbering ILU: An effective algebraic multilevel ILU preconditioner for sparse matrices, *SIAM J. Matrix Anal. Appl.* **20**, 1007 (1999).

59. J. H. Bramble, *Multigrid Methods* (Longman Scientific & Technical, Harlow, UK, 1993).

60. C. W. Brand, An incomplete factorization preconditioning using repeated red/black ordering, *Numer. Math.* **61**, 433 (1992).

61. A. Brandt, Multi-level adaptive solutions to boundary-value problems, *Math. Comput.* **31**, 333 (1977).

62. A. Brandt, S. McCormick, and J. Ruge, Algebraic multigrid (AMG) for sparse matrix equations, in *Sparsity and Its Applications*, edited by D. J. Evans, p. 257 (Cambridge Univ. Press, Cambridge, UK, 1985).

63. C. Brezinski, *Projection Methods for Systems of Equations*, Studies in Computational Mathematics, Vol. 27 (North-Holland, Amsterdam, 1997).

64. R. Bridson and W.-P. Tang, Ordering, anisotropy and factored sparse approximate inverses, *SIAM J. Sci. Comput.* **21**, 867 (1999).

65. R. Bridson and W.-P. Tang, A structural diagnosis of some IC orderings, *SIAM J. Sci. Comput.* **22**, 1527 (2000).

66. R. Bridson and W.-P. Tang, Refining an approximate inverse, *J. Comput. Appl. Math.* **22**, 1527 (2000).

67. R. Bridson and W.-P. Tang, Multiresolution approximate inverse preconditioners, *SIAM J. Sci. Comput.* **23**, 463 (2001).

68. W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed. (SIAM, Philadelphia, PA, 2000).

69. O. Bröker and M. J. Grote. Sparse approximate inverse smoothers for geometric and algebraic multigrid, *Appl. Numer. Math.* **41**, 61 (2002).

70. O. Bröker, M. J. Grote, C. Mayer, and A. Reusken, Robust parallel smoothing for multigrid via sparse approximate inverses, *SIAM J. Sci. Comput.* **23**, 1395 (2001).

71. A. M. Bruaset, *A Survey of Preconditioned Iterative Methods* (Longman Scientific & Technical, Harlow, UK, 1995).

72. N. I. Buleev, A numerical method for solving two dimensional diffusion equations, *At. Energ.* **6**, 338 (1959) [in Russian].

73. N. I. Buleev, A numerical method for solving two- and three-dimensional diffusion equations, *Mat. Sb.* **51**, 227 (1960) [in Russian].

74. B. Carpentieri, I. S. Duff, and L. Giraud, Sparse pattern selection strategies for robust Frobenius-norm minimization preconditioners in electromagnetics, *Numer. Linear Algebra Appl.* **7**, 667 (2000).

75. B. Carpentieri, I. S. Duff, L. Giraud, and M. Magolu monga Made, *Sparse Symmetric Preconditioners for Dense Linear Systems in Electromagnetics*, CERFACS Technical Report TR/PA/01/35 (Toulouse, France, 2001).

76. B. Carpentieri, I. S. Duff, L. Giraud, and G. Sylvand, Combining fast multipole techniques and approximate inverse preconditioners for large calculations in electromagnetism, in *Activity Report of the Parallel Algorithms Project at CERFACS, January 2001–December 2001*, CERFACS Tecnical Report TR/PA/01/105 (Toulouse, France, 2001).

77. L. Cesari, Sulla risoluzione dei sistemi di equazioni lineari per approssimazioni successive, *Atti Accad. Nazionale Lincei R. Classe Sci. Fis. Mat. Nat.* **25**, 422 (1937).

78. M. Challacombe, A simplified density matrix minimization for linear scaling self-consistent field theory, *J. Chem. Phys.* **110**, 2332 (1999).

79. M. Challacombe, A general parallel sparse-blocked matrix multiply for linear scaling SCF theory, *Comput. Phys. Commun.* **128**, 93 (2000).

80. T. F. Chan, W.-P. Tang, and W.-L. Wan, Wavelet sparse approximate inverse preconditioners, *BIT* **37**, 644 (1997).

81. T. F. Chan and H. A. van der Vorst, Approximate and incomplete factorizations, in *Parallel Numerical Algorithms*, edited by D. E. Keyes, A. Sameh, and V. Venkatakrishnan, ICASE/LaRC Interdisciplinary Series in Science and Engineering, Vol. 4, p. 167 (Kluwer Academic, Dordrecht, 1997).

82. A. Chapman, Y. Saad, and L. Wigton, High-order ILU preconditioners for CFD problems, *Int. J. Numer. Methods Fluids* **33**, 767 (2000).

83. D. Chen and S. Toledo, Vaidya's preconditioners: implementation and experimental study, submitted for publication.

84. K. Chen, On a class of preconditioning methods for dense linear systems from boundary elements, *SIAM J. Sci. Comput.* **20**, 684 (1998).

85. K. Chen, Discrete wavelet transforms accelerated sparse preconditioners for dense boundary element systems, *Electron. Trans. Numer. Anal.* **8**, 138 (1999).

86. K. Chen, An analysis of sparse approximate inverse preconditioners for boundary integral equations, *SIAM J. Matrix Anal. Appls.* **22**, 1058 (2001).

87. M. P. Chernesky, On preconditioned Krylov subspace methods for discrete convection-diffusion problems, *Numer. Methods Partial Differential Equations* **13**, 321 (1997).

88. H. Choi and D. B. Szyld, Application of threshold partitioning of sparse matrices to Markov chains, in *Proceedings of the IEEE International Computer Performance and Dependability Symposium, IPDS'96*, p. 158 (IEEE Computer Soc. Press, Los Alamitos, CA, 1996).

89. R. Choquet and V. Perrier, Fast wavelet iterative solvers applied to the Neumann problem, *Aerospace Sci. Technol.* **4**, 135 (2000).

90. E. Chow, A priori sparsity patterns for parallel sparse approximate inverse preconditioners, *SIAM J. Sci. Comput.* **21**, 1804 (2000).

91. E. Chow, Parallel implementation and performance characteristics of sparse approximate inverse preconditioners with a priori sparsity patterns, *Int. J. High-Perform. Comput. Appl.* **15**, 56 (2001).

92. E. Chow and M. A. Heroux, An object-oriented framework for block preconditioning, *ACM Trans. Math. Software* **24**, 159 (1998).

93. E. Chow and Y. Saad, Approximate inverse techniques for block-partitioned matrices, *SIAM J. Sci. Comput.* **18**, 1657 (1997).

94. E. Chow and Y. Saad, Experimental study of ILU preconditioners for indefinite matrices, *J. Comput. Appl. Math.* **86**, 387 (1997).

95. E. Chow and Y. Saad, ILUS: An incomplete ILU preconditioner in sparse skyline format, *Int. J. Numer. Methods Fluids* **25**, 739 (1997).

96. E. Chow and Y. Saad, Approximate inverse preconditioners via sparse-sparse iterations, *SIAM J. Sci. Comput.* **19**, 995 (1998).

97. A. J. Cleary, R. D. Falgout, V. E. Henson, J. J. Jones, T. A. Manteuffel, S. F. McCormick, G. N. Miranda, and J. W. Ruge, Robustness and scalability of algebraic multigrid, *SIAM J. Sci. Comput.* **21**, 1886 (2000).

98. S. S. Clift and W.-P. Tang, Weighted graph based ordering techniques for preconditioned conjugate gradient methods, *BIT* **35**, 30 (1995).

99. A. Cohen and R. Masson, Wavelet methods for second-order elliptic problems, preconditioning, and adaptivity, *SIAM J. Sci. Comput.* **21**, 1006 (1999).

100. P. Concus, G. H. Golub, and G. Meurant, Block preconditioning for the conjugate gradient method, *SIAM J. Sci. Stat. Comput.* **6**, 220 (1985).

101. P. Concus, G. H. Golub, and D. P. O'Leary, A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations, in *Sparse Matrix Computations*, edited by J. R. Bunch and D. J. Rose, p. 309 (Academic Press, New York, 1976).

102. J. D. F. Cosgrove, J. C. Díaz, and A. Griewank, Approximate inverse preconditioning for sparse linear systems, *Int. J. Comput. Math.* **44**, 91 (1992).

103. E. Cuthill, Several strategies for reducing the bandwidth of matrices, in *Sparse Matrices and Their Applications*, p. 157, edited by D. J. Rose and R. A. Willoughby (Plenum, New York, 1972).

104. W. Dahmen and L. Elsner, Algebraic multigrid methods and the Schur complement, in *Robust MultiGrid Methods (Kiel 1988)*, Notes in Numerical Fluid Mechanics 23, p. 58 (Vieweg, Braunschweig, 1989).

105. E. Darve, The fast multipole method. I. Error analysis and asymptotic complexity, *SIAM J. Numer. Anal.* **38**, 98 (2000).

106. E. Darve, The fast multipole method: Numerical implementation, *J. Comput. Phys.* **160**, 195 (2000).

107. T. A. Davis, *University of Florida Sparse Matrix Collection*, NA Digest, Vol. 92, No. 42, October 16, 1994; NA Digest, Vol. 96, No. 28, July 23, 1996; and NA Digest, Vol. 97, No. 23, June 7, 1997, available at http://www.cise.ufl.edu/research/sparse/matrices, ftp://ftp.cise.ufl.edu/pub/faculty/davis/matrices.

108. E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang, Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems, *SIAM J. Matrix Anal. Appl.* **13**, 944 (1992).

109. E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang, Two variants of minimum discarded fill ordering, in *Iterative Methods in Linear Algebra*, edited by R. Beauwens and P. de Groen, p. 603 (Elsevier Science, North-Holland, 1992).

110. J. E. Dendy, Black box multigrid, *J. Comput. Phys.* **48**, 366 (1980).

111. E. de Doncker and A. K. Gupta, Coarse grain preconditioned conjugate gradient solver for large sparse systems, in *Proceedings of the Seventh SIAM Conference on Parallel Processing for Scientific Computing*, edited by J. Lewis, p. 472 (SIAM, Philadelphia, PA, 1995).

112. S. Demko, W. F. Moss, and P. W. Smith, Decay rates for inverses of band matrices, *Math. Comput.* **43**, 491 (1984).

113. J. C. Díaz and K. Komara, Incomplete multilevel Cholesky factorizations, *SIAM J. Matrix Anal. Appl.* **22**, 895 (2000).

114. S. Doi, On parallelism and convergence of incomplete LU factorizations, *Appl. Numer. Math.* **7**, 417 (1991).

115. S. Doi and T. Washio, Ordering strategies and related techniques to overcome the trade-off between parallelism and convergence in incomplete factorizations, *Parallel Comput.* **25**, 1995 (1999).

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

117. J. Douglas, Jr., and H. H. Rachford, Jr., On the numerical solution of heat conduction problems in two or three space variables, *Trans. Am. Math. Soc.* **82**, 421 (1956).

118. J. Drkošová, A. Greenbaum, M. Rozložník, and Z. Strakoš, Numerical stabilty of GMRES, *BIT* **35**, 309 (1995).

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

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

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

122. I. S. Duff, R. G. Grimes, and J. G. Lewis, *The Rutherford–Boeing Sparse Matrix Collection*, Technical Report RAL-TR-97-031 (Rutherford Appleton Laboratory, Chilton, UK, 1997).

123. I. S. Duff and J. Koster, The design and use of algorithms for permuting large entries to the diagonal of sparse matrices, *SIAM J. Matrix Anal. Appl.* **20**, 889 (1999).

124. I. S. Duff and J. Koster, On algorithms for permuting large entries to the diagonal of a sparse matrix, *SIAM J. Matrix Anal. Appl.* **22**, 973 (2001).

125. I. S. Duff and G. Meurant, The effect of ordering on preconditioned conjugate gradients, *BIT* **29**, 635 (1989).

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

127. L. C. Dutto, The effect of ordering on preconditioned GMRES algorithm, for solving the compressible Navier-Stokes equations, *Int. J. Numer. Methods Eng.* **36**, 457 (1993).

128. V. Eijkhout, Analysis of parallel incomplete point factorizations, *Linear Algebra Its Appl.* **154**, 723 (1991).

129. V. Eijkhout, Beware of unperturbed modified incomplete factorizations, in *Iterative Methods in Linear Algebra*, edited by R. Beauwens and P. de Groen, p. 583 (Elsevier Science, North-Holland, 1992).

130. S. Eisenstat, Efficient implementation of a class of conjugate gradient methods, *SIAM J. Sci. Stat. Comput.* **2**, 1 (1981).

131. H. C. Elman, A stability analysis of incomplete LU factorizations, *Math. Comput.* **47**, 191 (1986).

132. H. C. Elman and M. P. Chernesky, Ordering effects on relaxation methods applied to the discrete one-dimensional convection-diffusion equation, *SIAM J. Numer. Anal.* **30**, 1268 (1993).

133. M. Engeli, Th. Ginsburg, H. Rutishauser, and E. Stiefel, *Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems* (Birkhäuser, Basel/Stuttgart, 1959).

134. D. J. Evans, The use of pre-conditioning in iterative methods for solving linear systems with symmetric positive definite matrices, *J. Inst. Math. Its Appl.* **4**, 295 (1968).

135. C. Farhat and F.-X. Roux, A method of finite-element tearing and interconnecting and its parallel solution algorithm, *Int. J. Numer. Methods Eng.* **32**, 1205 (1991).

136. C. Farhat, J. Mandel, and F.-X. Roux, Optimal convergence properties of the FETI domain decomposition method, *Comput. Methods Appl. Mech. Eng.* **115**, 365 (1994).

137. C. Farhat and J. Mandel, The two-level FETI method for static and dynamic plate problems. Part I: An optimal iterative solver for biharmonic systems, *Comput. Methods Appl. Mech. Eng.* **155**, 129 (1998).

138. R. P. Fedorenko, A relaxation method for solving elliptic difference equations, *USSR Comput. Math. Math. Phys.* **1**, 1092 (1961).

139. R. P. Fedorenko, The speed of convergence of one iterative process, *USSR Comput. Math. Math. Phys.* **4**, 227 (1964).

140. M. R. Field, *An Efficient Parallel Preconditioner for the Conjugate Gradient Algorithm*, Hitachi Dublin Laboratory Technical Report HDL-TR-97-175 (Dublin, Ireland, 1997).

141. M. R. Field, *Improving the Performance of Factorised Sparse Approximate Inverse Preconditioner*, Hitachi Dublin Laboratory Technical Report HDL-TR-98-199 (Dublin, Ireland, 1998).

142. B. Fischer, *Polynomial Based Iteration Methods for Symmetric Linear Systems* (Wiley-Teubner, Chichester, 1996).

143. R. Fletcher, Conjugate gradient methods for indefinite systems, in *Numerical Analysis Dundee 1975*, edited by G. A. Watson, p. 73, Lecture Notes in Mathematics No. 506 (Springer, Berlin, 1976).

144. L. Fox L, H. D. Huskey, and J. H. Wilkinson, Notes on the solution of algebraic linear simultaneous equations, *Q. J. Mech. Appl. Math.* **1**, 149 (1948).

145. P. O. Frederickson, *Fast Approximate Inversion of Large Sparse Linear Systems*, Math. Report 7 (Lakehead University, Thunder Bay, Ontario, 1975).

146. R. Freund, A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems, *SIAM J. Sci. Comput.* **14**, 470 (1993).

147. R. Freund, G. H. Golub, and N. M. Nachtigal, Iterative solution of linear systems, *Acta Numer.* **1**, 57 (1991).

148. R. Freund and N. M. Nachtigal, QMR: A quasi-minimal residual method for non-Hermitian linear systems, *Numer. Math.* **60**, 315 (1991).

149. A. George, Nested dissection of a regular finite element mesh, *SIAM J. Numer. Anal.* **10**, 345 (1973).

150. A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems* (Prentice–Hall, Englewood Cliffs, NJ, 1981).

151. A. George and J. W. Liu, The evolution of the minimum degree algorithm, *SIAM Rev.* **31**, 1 (1989).

152. N. E. Gibbs, W. G. Poole, Jr., and P. K. Stockmeyer, An algorithm for reducing the bandwidth and profile of a sparse matrix, *SIAM J. Numer. Anal.* **13**, 236 (1976).

153. J. R. Gilbert, Predicting structure in sparse matrix computations, *SIAM J. Matrix Anal. Appl.* **15**, 62 (1994).

154. P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization* (Academic Press, London, 1981).

155. G. H. Golub and D. P. O'Leary, Some history of the conjugate gradient and Lanczos methods: 1948–1976, *SIAM Rev.* **31**, 50 (1989).

156. G. H. Golub and C. F. van Loan, *Matrix Computations*, 3rd ed. (Johns Hopkins Press, Baltimore, 1996).

157. G. H. Golub and Q. Ye, Inexact preconditioned conjugate-gradient method with inner-outer iteration, *SIAM J. Sci. Comput.* **21**, 1305 (1999).

158. N. I. M. Gould and J. A. Scott, Sparse approximate-inverse preconditioners using norm-minimization techniques, *SIAM J. Sci. Comput.* **19**, 605 (1998).

159. A. Greenbaum, Analysis of a multigrid method as an iterative method for solving linear systems, *SIAM J. Numer. Anal.* **21**, 473 (1984).

160. A. Greenbaum, *Iterative Methods for Solving Linear Systems* (SIAM, Philadelphia, PA, 1997).

161. A. Greenbaum, V. Pták, and Z. Strakoš, Any nonincreasing convergence curve is possible for GMRES, *SIAM J. Matrix Anal. Appl.* **17**, 465 (1996).

162. A. Greenbaum and Z. Strakoš, Predicting the behavior of finite precision Lanczos and conjugate gradient computations, *SIAM J. Matrix Anal. Appl.* **13**, 121 (1992).

163. M. Grote and T. Huckle, Parallel preconditioning with sparse approximate inverses, *SIAM J. Sci. Comput.* **18**, 838 (1997).

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

165. G. Haase, Parallel incomplete Cholesky preconditioners based on the nonoverlapping data distribution, *Parallel Comput.* **24**, 1685 (1998).

166. W. Hackbusch, *Multi-Grid Methods and Applications* (Springer-Verlag, Berlin, 1985).

167. W. Hackbusch, *Iterative Solution of Large Sparse Systems of Equations* (Springer-Verlag, New York, 1994).

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

169. M. Heniche, Y. Secretan, and M. Leclerc, Efficient ILU preconditioning and inexact-Newton-GMRES to solve the 2D steady shallow water equations, *Commun. Numer. Methods Eng.* **17**, 69 (2001).

170. V. E. Henson and U. M. Yang, BoomerAMG: A parallel algebraic multigrid solver and preconditioner, *Appl. Numer. Math.* **41**, 155 (2002).

171. M. R. Hestenes and E. L. Stiefel, Methods of conjugate gradients for solving linear systems, *J. Res. Natl. Bur. Stand.* **49**, 409 (1952).

172. I. Hladík, M. B. Reed, and G. Swoboda, Robust preconditioners for linear elasticity FEM analyses, *Int. J. Numer. Methods Eng.* **40**, 2109 (1997).

173. A. S. Householder, *The Theory of Matrices in Numerical Analysis* (Blaisdell, New York, 1964).

174. T. Huckle, Approximate sparsity patterns for the inverse of a matrix and preconditioning, *Appl. Numer. Math.* **30**, 291 (1999).

175. T. Huckle, Factorized sparse approximate inverses for preconditioning and smoothing, *Selčuk J. Appl. Math.* **1**, 63 (2000).

176. T. Huckle, Factorized sparse approximate inverses for preconditioning, in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001)*, (CSREA Press, Las Vegas, Nevada, 2001).

177. D. Hysom, *New Sequential and Scalable Parallel Algorithms for Incomplete Factor Preconditioning*, Ph.D. thesis (Dep. Computer Science, Old Dominion Univ., Norfolk, VA, 2001).

178. D. Hysom and A. Pothen, A scalable parallel algorithm for incomplete factor preconditioning, *SIAM J. Sci. Comput.* **22**, 2194 (2001).

179. V. P. Il'in, *Iterative Incomplete Factorization Methods* (World Scientific, Singapore, 1992).

180. C. G. J. Jacobi, Über eine neue Aufiösungsart der bei der Methode der kleinsten Quadrate vorkommenden linearen Gleichungen, *Astron. Nachrichten* **22**, 297 (1845).

181. O. G. Johnson, C. A. Micchelli, and G. Paul, Polynomial preconditioning for conjugate gradient calculations, *SIAM J. Numer. Anal.* **20**, 362 (1983).

182. M. T. Jones and P. E. Plassmann, Scalable iterative solution of sparse linear systems, *Parallel Comput.* **20**, 753 (1994).

183. M. T. Jones and P. E. Plassmann, An improved incomplete Cholesky factorization, *ACM Trans. Math. Software.* **21**, 5 (1995).

184. M. T. Jones and P. E. Plassmann, Algorithm 740: Fortran subroutines to compute improved incomplete Cholesky factorizations, *ACM Trans. Math. Software* **21**, 18 (1995).

185. I. E. Kaporin, New convergence results and preconditioning strategies for the conjugate gradient method, *Numer. Linear Algebra Appl.* **1**, 179 (1994).

186. I. E. Kaporin, High quality preconditioning of a general symmetric positive definite matrix based on its $U^T U + U^T R + R^T U$ decomposition, *Numer. Linear Algebra Appl.* **5**, 483 (1998).

187. I. E. Kaporin and I. N. Konshin, A parallel block overlap preconditioning with inexact submatrix inversion for linear elasticity problems, *Numer. Linear Algebra Appl.* **9**, 141 (2002).

188. G. Karypis and V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM J. Sci. Comput.* **20**, 359 (1998).

189. G. Karypis and V. Kumar, *Parallel Threshold-Based ILU Factorization*, Technical Report 061 (Dep. Computer Science/Army HPC Research Center, Univ. Minnesota, Minneapolis, 1998).

190. D. S. Kershaw, The incomplete Cholesky conjugate gradient method for the iterative solution of systems of linear equations, *J. Comput. Phys.* **26**, 43 (1978).

191. S. A. Kharchenko, L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin, A robust AINV-type method for constructing sparse approximate inverse preconditioners in factored form, *Numer. Linear Algebra Appl.* **8**, 165 (2001).

192. A. W. Knyazev, Preconditioned eigensolvers—an oxymoron? *Electron. Trans. Numer. Anal.* **7**, 104 (1998).

193. L. Yu. Kolotilina, Explicit preconditioning of systems of linear algebraic equations with dense matrices, *J. Sov. Math.* **43**, 2566 (1988).

194. L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin, Factorized sparse approximate inverse preconditionings. IV: Simple approaches to rising efficiency, *Numer. Linear Algebra Appl.* **6**, 515 (1999).

195. L. Yu. Kolotilina and A. Yu. Yeremin, On a family of two-level preconditionings of the incomplete block factorization type, *Sov. J. Numer. Anal. Math. Model.* **1**, 293 (1986).

196. L. Yu. Kolotilina and A. Yu. Yeremin, Factorized sparse approximate inverse preconditioning. I. Theory, *SIAM J. Matrix Anal. Appl.* **14**, 45 (1993).

197. L. Yu. Kolotilina and A. Yu. Yeremin, Factorized sparse approximate inverse preconditioning. II: Solution of 3D FE systems on massively parallel computers, *Int. J. High Speed Comput.* **7**, 191 (1995).

198. A. Krechen and K. Stüben, Parallel algebraic multigrid based on subdomain blocking, *Parallel Comput.* **27**, 1009 (2001).

199. C. Lanczos, Solutions of systems of linear equations by minimized iterations, *J. Res. Natl. Bur. Stand.* **49**, 33 (1952).

200. E. W. Larsen, Unconditionally stable diffusion-synthetic acceleration methods for the slab geometry discrete ordinates equations. Part I: Theory, *Nucl. Sci. Eng.* **82**, 47 (1982).

201. S. Le Borne, Ordering techniques for two- and three-dimensional convection-dominated elliptic boundary value problems, *Computing* **64**, 123 (2000).

202. S. L. Lee, *Krylov Methods for the Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*, Ph.D. thesis and Technical Report UIUCDS-R-93-1814 (Dep. Computer Science, Univ. Illinois at Urbana-Champaign, 1993).

203. N. Li, Y. Saad, and M. Sosonkina, p*ARMS: A Parallel Version of the Algebraic Recursive Multilevel Solver*, Technical Report UMSI-2001-100 (Minnesota Supercomputer Institute, Univ. Minnesota, Minneapolis, 2001).

204. C.-J. Lin and J. J. Moré, Incomplete Cholesky factorizations with limited memory, *SIAM J. Sci. Comput.* **21**, 24 (1999).

205. C.-J. Lin and R. Saigal, An incomplete Cholesky factorization for dense symmetric positive definite matrices, *BIT* **40**, 536 (2000).

206. R. J. Lipton, D. J. Rose, and R. E. Tarjan, Generalized nested dissection, *SIAM J. Numer. Anal.* **16**, 346 (1979).

207. J. W. H. Liu, Modification of the minimum degree algorithm by multiple elimination, *ACM Trans. Math. Software* **11**, 141 (1985).

208. J. W. H. Liu, E. G. Ng, and B. W. Peyton, On finding supernodes for sparse matrix computations, *SIAM J. Matrix Anal. Appl.* **14**, 242 (1993).

209. M. Magolu monga Made, Incomplete factorization based preconditionings for solving the Helmholtz equation, *Int. J. Numer. Methods Eng.* **50**, 1077 (2001).

210. M. Magolu monga Made, R. Beauwens, and G. Warzée, Preconditioning of discrete Helmholtz operators perturbed by a diagonal complex matrix, *Commun. Numer. Methods Eng.* **16**, 801 (2000).

211. M. Magolu monga Made and B. Polman, Experimental comparison of three-dimensional point and line modified incomplete factorization, *Numer. Algorithms* **23**, 51 (2000).

212. M. Magolu monga Made and H. A. van der Vorst, Parallel incomplete factorizations with pseudooverlapping subdomains, *Parallel Comput.* **27**, 989 (2001).

213. T. A. Manteuffel, An incomplete factorization technique for positive definite linear systems, *Math. Comput.* **34**, 473 (1980).

214. D. R. McCoy and E. W. Larsen, Unconditionally stable diffusion-synthetic acceleration methods for the slab geometry discrete ordinates equations. Part II: Numerical results, *Nucl. Sci. Eng.* **82**, 64 (1982).

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

216. G. A. Meurant, A review of the inverse of symmetric tridiagonal and block tridiagonal matrices, *SIAM J. Matrix Anal. Appl.* **13**, 707 (1992).

217. G. A. Meurant, *Computer Solution of Large Linear Systems*, Studies in Mathematics and Its Applications, Vol. 28 (North-Holland, Amsterdam, 1999).

218. G. A. Meurant, A multilevel AINV preconditioner, *Numer. Algorithms* **29**, 107 (2002).

219. G. A. Meurant, Numerical experiments with algebraic multilevel preconditioners, *Electron. Trans. Numer. Anal.* **12**, 1 (2001).

220. O. Yu. Milyukova, Parallel approximate factorization method for solving discrete elliptic equations, *Parallel Comput.* **27**, 1365 (2001).

221. R. B. Morgan, Preconditioning eigenvalues and some comparison of solvers, *J. Comput. Appl. Math.* **123**, 101 (2000).

222. V. A. Mousseau, D. A. Knoll, and W. J. Rider, Physics-based preconditioning and the Newton–Krylov method for non-equilibrium radiation diffusion, *J. Comput. Phys.* **160**, 743 (2000).

223. M. F. Murphy, G. H. Golub, and A. J. Wathen, A note on preconditioning for indefinite linear systems, *SIAM J. Sci. Comput.* **21**, 1969 (2000).

224. S. G. Nash, Ed., *A History of Scientific Computing* (Addison–Wesley, Reading, MA, 1990).

225. National Institute of Standards, *Matrix Market*, available online at http://math.nist.gov/MatrixMarket.

226. E. Ng, B. W. Peyton, and P. Raghavan, A blocked incomplete Cholesky preconditioner for hierarchical-memory computers, in *Iterative Methods in Scientific Computation IV*, edited by D. R. Kincaid and A. C. Elster, IMACS Series in Computational and Applied Mathematics, Vol. 5, 211 (IMACS, New Brunswick, NJ, 1999).

227. Y. Notay, Using approximate inverses in algebraic multilevel methods, *Numer. Math.* **80**, 397 (1998).

228. Y. Notay, A multilevel block incomplete factorization preconditioning, *Appl. Numer. Math.* **31**, 209 (1999).

229. Y. Notay, Optimal order preconditioning of finite difference matrices, *SIAM J. Sci. Comput.* **21**, 1991 (2000).

230. Y. Notay, Flexible conjugate gradient, *SIAM J. Sci. Comput.* **22**, 1444 (2000).

231. T. A. Oliphant, An extrapolation process for solving linear systems, *Q. Appl. Math.* **20**, 257 (1962).

232. M. Olschowka and A. Neumaier, A new pivoting strategy for Gaussian elimination, *Linear Algebra Its Appl.* **240**, 131 (1996).

233. J. O'Neil and D. B. Szyld, A block ordering method for sparse matrices, *SIAM J. Sci. Stat. Comput.* **11**, 811 (1990).

234. H. L. Ong, Fast approximate solution of large scale sparse linear systems, *J. Comput. Appl. Math.* **10**, 45 (1984).

235. A. Padiy, O. Axelsson, and B. Polman, Generalized augmented matrix preconditioning approach and its application to iterative solution of ill-conditioned algebraic systems, *SIAM J. Matrix Anal. Appl.* **22**, 793 (2001).

236. C. C. Paige and M. A. Saunders, Solution of sparse indefinite systems of linear equations, *SIAM J. Numer. Anal.* **12**, 617 (1975).

237. D. W. Peaceman and H. H. Rachford, Jr., The numerical solution of parabolic and elliptic differential equations, *J. Soc. Ind. Appl. Math.* **3**, 28 (1955).

238. E. L. Poole and J. M. Ortega, Multicolor ICCG methods for vector computers, *SIAM J. Numer. Anal.* **24**, 1394 (1987).

239. Y. Qu and J. Fish, Global-basis two-level method for indefinite systems. Part 2: Computational issues, *Int. J. Numer. Methods Eng.* **49**, 461 (2000).

240. A. Quarteroni and A. Valli, *Domain Decomposition Methods for Partial Differential Equations* (Clarendon, Oxford, 1999).

241. J. K. Reid, On the method of conjugate gradients for the solution of large sparse systems of linear equations, in *Large Sparse Sets of Linear Equations*, edited by J. K. Reid, p. 231 (Academic Press, New York, 1971).

242. A. Reusken, A multigrid method based on incomplete Gaussian elimination, *Numer. Linear Algebra Appl.* **3**, 369 (1996).

243. Y. Robert, Regular incomplete factorizations of real positive definite matrices, *Linear Algebra Its Appl.* **48**, 105 (1982).

244. J. W. Ruge and K. Stüben, Algebraic multigrid, in *Multigrid Methods*, edited by S. F. McCormick, p. 73 (SIAM, Philadelphia, PA, 1987).

245. Y. Saad, Preconditioning techniques for nonsymmetric and indefinite linear systems, *J. Comput. Appl. Math.* **24**, 89 (1988).

246. Y. Saad, Krylov subspace methods on supercomputers, *SIAM J. Sci. Stat. Comput.* **10**, 1200 (1989).

247. Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, *SIAM J. Sci. Comput.* **14**, 461 (1993).

248. Y. Saad, ILUT: A dual threshold incomplete LU factorization, *Numer. Linear Algebra Appl.* **1**, 387 (1994).

249. Y. Saad, Highly parallel preconditioners for general sparse matrices, in *Recent Advances in Iterative Methods*, edited by G. H. Golub, R. M. Luskin, and A. Greenbaum, p. 165, IMA Volumes in Mathematics and Its Applications, Vol. 60 (Springer-Verlag, New York, 1994).

250. Y. Saad, ILUM: A multi-elimination ILU preconditioner for general sparse matrices, *SIAM J. Sci. Comput.* **17**, 830 (1996).

251. Y. Saad, *Iterative Methods for Sparse Linear Systems* (PWS Publishing, Boston, 1996).

252. Y. Saad, *Finding Exact and Approximate Block Structures for ILU Preconditioning*, Technical Report UMSI-2001-93 (Minnesota Supercomputer Institute, Univ. Minnesota, Minneapolis, 2001).

253. Y. Saad and M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Stat. Comput.* **7**, 856 (1986).

254. Y. Saad and B. Suchomel, ARMS: An algebraic recursive multilevel solver for general sparse linear systems, *Numer. Linear Algebra Appl.* **9**, 359 (2002).

255. Y. Saad and H. A. van der Vorst, Iterative solution of linear systems in the 20th century, *J. Comput. Appl. Math.* **123**, 1 (2000).

256. Y. Saad and J. Zhang, BILUM: Block versions of multielimination and multilevel preconditioner for general sparse linear systems, *SIAM J. Sci. Comput.* **20**, 2103 (1999).

257. Y. Saad and J. Zhang, BILUTM: A domain-based multilevel block ILUT preconditioner for general sparse matrices, *SIAM J. Matrix Anal. Appl.* **21**, 279 (1999).

258. P. Saint-Georges, G. Warzee, Y. Notay, and R. Beauwens, High-performance PCG solvers for FEM structural analyses, *Int. J. Numer. Methods Eng.* **39**, 1133 (1996).

259. P. Saint-Georges, G. Warzee, Y. Notay, and R. Beauwens, Problem-dependent preconditioners for iterative solvers in FE elastostatics, *Comput. Struct.* **73**, 33 (1999).

260. R. B. Schnabel and E. Eskow, A new modified Cholesky factorization, *SIAM J. Sci. Stat. Comput.* **11**, 1136 (1990).

261. J. W. Sheldon, On the numerical solution of elliptic difference equations, *Math. Tables Other Aids Comput.* **9**, 101 (1955).

262. G. L. G. Sleijpen and F. W. Wubs, *Exploiting Multilevel Preconditioning Techniques in Eigenvalue Computations*, Preprint nr. 1117 (Dep. Mathematics, Univ. Utrecht, Utrecht, The Netherlands, 1999; revised version 2001).

263. S. W. Sloan, An algorithm for profile and wavefront reduction of sparse matrices, *Int. J. Numer. Methods Eng.* **23**, 239 (1986).

264. B. F. Smith, P. E. Bjørstad, and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations* (Cambridge Univ. Press, Cambridge/New York/Melbourne, 1996).

265. R. D. Smith, J. K. Dukowicz, and R. C. Malone, Parallel ocean general circulation modeling, *Phys. D* **60**, 38 (1992).

266. K. Stüben, Algebraic multigrid (AMG): Experiences and comparisons, *Appl. Math. Comput.* **13**, 419 (1983).

267. K. Stüben, A review of algebraic multigrid, *J. Comput. Appl. Math.* **128**, 281 (2001).

268. M. Suarjana and K. H. Law, A robust incomplete factorization based on value and space constraints, *Int. J. Numer. Methods Eng.* **38**, 1703 (1995).

269. D. B. Szyld and J. Vogel, FQMR: A flexible quasi-minimal residual method with inexact preconditioning, *SIAM J. Sci. Comput.* **23**, 363 (2001).

270. W.-P. Tang and W.-L. Wan, Sparse approximate inverse smoother for multigrid, *SIAM J. Matrix Anal. Appl.* **21**, 1236 (2000).

271. M. Tismenetsky, A new preconditioning technique for solving large sparse linear systems, *Linear Algebra Its Appl.* **154-156**, 331 (1991).

272. L. N. Trefethen and D. Bau, III, *Numerical Linear Algebra* (SIAM, Philadelphia, PA, 1997).

273. U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid* (Academic Press, London, 2000).

274. A. M. Turing, Rounding-off errors in matrix processes, *Q. J. Mech. Appl. Math.* **1**, 287 (1948).

275. R. Underwood, *An Approximate Factorization Procedure Based on the Block Cholesky Decomposition and Its Use with the Conjugate Gradient Method*, Report NEDO-11386 (General Electric Co., Nuclear Energy Div., San José, CA, 1976).

276. A. van der Ploeg, E. F. F. Botta, and F. W. Wubs, Nested grids ILU-decomposition (NGILU), *J. Comput. Appl. Math.* **66**, 515 (1996).

277. H. A. van der Vorst, Iterative solution methods for certain sparse linear systems with a nonsymmetric matrix arising from PDE-problems, *J. Comput. Phys.* **44**, 1 (1981).

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

279. H. A. van der Vorst, The convergence behaviour of preconditioned CG and CGS in the presence of rounding errors, in *Preconditioned Conjugate Gradient Methods*, edited by O. Axelsson and L. Yu. Kolotilina, p. 126, Lecture Notes in Mathematics, Vol. 1457 (Springer-Verlag, Berlin, 1990).

280. H. A. van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems, *SIAM J. Sci. Stat. Comput.* **12**, 631 (1992).

281. R. S. Varga, Factorizations and normalized iterative methods, in *Boundary Problems in Differential Equations*, edited by R. E. Langer, p. 121 (Univ. Wisconsin Press, Madison, 1960).

282. R. S. Varga, *Matrix Iterative Analysis* (Prentice–Hall, Englewood Cliffs, NJ, 1962).

283. R. S. Varga, E. B. Saff, and V. Mehrmann, Incomplete factorizations of matrices and connections with H-matrices, *SIAM J. Numer Anal.* **17**, 787 (1980).

284. P. S. Vassilevski, A block-factorization (algebraic) formulation of multigrid and Schwarz methods, *East-West J. Numer. Math.* **6**, 65 (1998).

285. C. Vuik, R. P. van Nooyen, and P. Wesseling, Parallelism in ILU-preconditioned GMRES, *Parallel Comput.* **24**, 1927 (1998).

286. E. Wachspress, *Iterative Solution of Elliptic Systems and Applications to the Neutron Diffusion Equations of Reactor Physics* (Prentice–Hall, Englewood Cliffs, NJ, 1966).

287. J. S. Warsa, T. A. Wareing, and J. E. Morel, Solution of the discontinuous P_1 equations in two-dimensional Cartesian geometry with two-level preconditioning, *SIAM J. Sci. Comput.*, in press.

288. J. W. Watts, III, A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation, *Soc. Petrol. Eng. J.* **21**, 345 (1981).

289. R. Weiss, *Parameter-Free Iterative Linear Solvers*, Mathematical Research, Vol. 97 (Akademie, Berlin, 1996).

290. G. Wittum, On the robustness of ILU-smoothing, *SIAM J. Sci. Stat. Comput.* **10**, 699 (1989).

291. P. H. Worley, Limits on parallelism in the numerical solution of linear PDEs, *SIAM J. Sci. Stat. Comput.* **12**, 1 (1991).

292. K. Wu, Y. Saad, and A. Stathopoulos, Inexact Newton preconditioning techniques for large symmetric eigenvalue problems, *Electron. Trans. Numer. Anal.* **7**, 202 (1998).

293. D. M. Young, *Iterative Methods for Solving Partial Difference Equations of Elliptic Type*, Ph.D. thesis (Harvard University, Cambridge, MA, 1950).

294. D. M. Young, *Iterative Solution of Large Linear systems* (Academic Press, New York, 1971).

295. J. Zhang, Sparse approximate inverse and multilevel block ILU preconditioning techniques for general sparse matrices, *Appl. Numer. Math.* **35**, 67 (2000).

296. J. Zhang, A multilevel dual reordering strategy for robust incomplete LU factorization of indefinite matrices, *SIAM J. Matrix Anal. Appl.* **22**, 925 (2001).