FAST ITERATIVE SOLVER FOR FINITE ELEMENTS USING INCOMPLETE ELIMINATION[1]

JAN MANDEL[2]

Computational Mathematics Group
Department of Mathematics
University of Colorado at Denver
Denver, Colorado 80217-3364

Keywords. Large-Scale Linear Systems, Preconditioning, Domain Decomposition, Multigrid, Hierarchical Bases, p-Version Finite Element Method

Abstract. The preconditioned conjugate gradients method is used with a special preconditioning for large sparse systems of linear equations arising from finite elements. First the system is transformed by the elimination of se'ected nonzeros. The transformed system is then preconditioned by a block diagonal matrix that includes a coarse discretization of the same problem. The selection of the block diagonal is adaptive, which makes the method flexible and efficient for problems with a complicated geometry. The method is illustrated on systems arising from the p-version finite element method for three-dimensional elasticity and a test implementation is described. Computational results show that the iterative solver outperforms significantly the state-of-the art direct sparse solver used in MSC/PROBE for real-world test problems with distorted elements both in terms of CPU time and storage. The advantage of the iterative solver increases with the size of the problem.

---

[1] Paper presented at the 1991 MSC World Users' Conference, Los Angeles, California, March 11-15, 1991.

[2] The work of the author reported here was partially supported by the National Science Foundation under grant DMS-8704169. The test problems and the use of MSC/PROBE were provided by the MacNeal-Schwendler Corporation.

1. Introduction. The solution of large, sparse systems of linear equations

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

presents a bottleneck for increasing the precision of three-dimensional finite element models. Nonlinear problems are also commonly reduced to the solution of a series of linear problems (1) by techniques such as Newton's method and its variants, which necessitates a repeated solution of linear systems (1).

While the use of computer resources for the generation of stiffness matrices is proportional to the number of elements, the solution of the resulting large, sparse systems of equations grows relatively more expensive due to *fill-in* as the model becomes larger: the triangular factors of the stiffness matrix have many more nonzero elements than the stiffness matrix itself. Various sparse matrix techniques have been developed to make direct solvers based on variants of Gaussian elimination more efficient [11], even on advanced vector architectures [3]. Unfortunately, fill-in cannot be avoided but only minimized by an ordering of variables. For symmetric, positive definite systems, elimination is stable without pivoting and so the operations performed do not depend on the numerical values of the data of the problem; thus performance of direct methods is independent of the numerical data in the symmetric, positive definite case.

Iterative methods do not involve any fill-in, but they use the original data of the problem or data structures that can be stored in about the same amount of memory as the original data. Because of this advantage, iterative methods are gaining acceptance in the finite element community [16]. The performance of iterative methods, however, does depend on the numerical values of the data of the problem to be solved. An important goal is thus to develop *robust* iterative methods that perform well for a wide range of data and exploit the special properties of the data at hand. Iterative methods gain efficiency by the use of information about the specific problem to be solved. *Multigrid methods* (see, for example [23]) are especially efficient when the system (1) is a discretization of an elliptic partial differential equation and several discretizations with larger step size (or coarser elements) of the same problem are available. A similar method using a hierarchy of discretizations for different degres p was also studied [12]. The main problem with application of multigrid ideas here is that the solver must be able to handle arbitrary distorted elements and irregular meshes.

The basic iterative method considered here is the method of *preconditioned conjugate gradients* [9]. For a description of this method, see §3 below. In each step, preconditioned conjugate gradients call for the evaluation of matrix-vector product Ax and the solution of an auxiliary system

$$Cx = r \tag{2}$$

for a given right-hand side r. The matrix C is called a *preconditioner*, and its choice is problem dependent. Some common choices for C are the diagonal of A or an incomplete LU decomposition of A [22, 24].

The present method uses a specific, problem dependent choice of the preconditioner C, which gives fast convergence of the iterations. The choice of C is *adaptive* to handle efficiently distorted elements, which occur often in practice. The efficiency of this approach depends on the structure of the system to be solved and on the choice of C. The method can be parallelized on the element level. It is related to multigrid methods and also to certain domain decomposition methods [8]. In the application to the p-version finite element method, each element is treated as a subdomain. For general criteria of applicability, see §6.

An early formulation of the method was presented in the conference paper [18]; for more theory and some other related methods, see also [19, 20].

The paper is organized as follows: §2 summarizes some basic facts about the p-version finite element method. The preconditioned conjugate gradients method is recalled in §3. §4 presents a formulation of the method and some theoretical results. Computational results are given in §5, and §6 discusses further applications and extensions.

2. The p-Version Finite Element Method and MSC/PROBE. The p-version in three dimensions as implemented in MSC/PROBE uses hierarchical serendipity elements of order p up to 8. Except for linear basis functions, the degrees of freedom are not associated with nodes. The basis functions for the reference brick element $\hat{K} = [-1, +1]^3$ are of the form

$$(3) \qquad (x \pm 1)(y \pm 1)(z \pm 1),$$

$$(4) \qquad (x-1)(y-1)(1-z^2)L_n(z), \qquad n = 0, \ldots, p-2,$$

$$(5) \qquad (x-1)(1-y^2)(1-z^2)L_m(y)L_n(z), \qquad m, n \geq 0, \quad m+n \leq p-4,$$

$$(6) \qquad (1-x^2)(1-y^2)(1-z^2)L_l(x)L_m(y)L_n(z), \qquad l, m, n \geq 0, \quad l+m+n \leq p-6,$$

associated with nodes, edges, faces, and the interior, respectively, with obvious permutations of the variables and changes of -1 to $+1$. In (4)–(6), L_n are polynomials of order n. Basis functions for other types of elements are defined similarly. For more details, see [6, 7, 26, 27].

The present method is invariant to the particular choice of the polynomials L_n. It can be applied whenever the basis functions split into groups associated with the nodes, with each edge, each face, and with the interior as in (3)–(6). For example, functions associated with an edge are zero on all faces not adjacent to that edge.

3. Preconditioned Conjugate Gradients. Let A and C be $N \times N$ *symmetric positive definite* matrices. Define the norms

$$\|x\| = \sqrt{x^T x}, \qquad \|x\|_A = \sqrt{x^T A x},$$

and the *relative condition number* κ of C and A as

$$\kappa = \frac{\max\limits_{x \neq 0} \frac{x^T A x}{x^T C x}}{\min\limits_{x \neq 0} \frac{x^T A x}{x^T C x}}.$$

FIG. 1. *The preconditioned conjugate gradient algorithm*

Step 1. Given $\varepsilon > 0$, $x_0 = 0$, let $k = 0$ and

$$\begin{aligned} r_0 &= b - Ax_0 = b \\ h_0 &= C^{-1}r_0 \\ p_0 &= h_0 \end{aligned}$$

Step 2. Do

$$\begin{aligned} \alpha_k &= \frac{r_k^T p_k}{(Ap_k)^T p_k} \\ x_{k+1} &= x_k + \alpha_k p_k \\ r_{k+1} &= r_k - \alpha_k A p_k \\ h_{k+1} &= C^{-1} r_{k+1} \\ \beta_{k+1} &= \frac{r_{k+1}^T h_{k+1}}{r_k^T h_k} \\ p_{k+1} &= h_{k+1} + \beta_k p_k \end{aligned}$$

Step 3. Let c_k be an estimate of the condition number of $C^{-1}A$. If $c_k \frac{\|h_{k+1}\|}{\|h_0\|} \leq \varepsilon$, exit; else $k = k + 1$ and go to Step 2.

Note that κ is also the ratio of the extreme eigenvalues of $C^{-1}A$.

The preconditioned conjugate gradients algorithm for the solution of $Ax = b$ is in Figure 1. This particular version of the algorithm follows the routine SPCG in CGCODE [1]. In Step 3, c_k is an estimate of κ, obtained from the sequence p_k by exploiting the relationship between conjugate gradients and Lanczos algorithm for the matrix $C^{-1}A$. It is the ratio of extreme eigenvalues of a tridiagonal matrix built using the coefficients β_k. This estimate is recomputed every few steps and always before the exit is actually taken. Let x^* be the solution of (1). The algorithm thus attempts to have the relative error on exit satisfy

$$\frac{\|x_k - x^*\|}{\|x^*\|} \leq \varepsilon,$$

using the estimate c_k of κ as an (admittedly very rough) estimate of $\|C^{-1}A\|\|(C^{-1}A)^{-1}\|$.

An alternative stopping criterion can be based on the bound

$$\frac{\|x_k - x^*\|_A}{\|x^*\|_A} \leq \sqrt{\kappa \frac{r_k^T h_k}{r_0^T h_0}}, \tag{7}$$

for the relative error in the energy norm. Again, the estimate c_k of the condition number κ is used rather than the unknown value κ. For more details, see [2].

In exact arithmetic, this algorithm terminates and gives an exact solution in at most N steps [14]. We are interested in it as an iterative method, performing much fewer than N steps. The behavior of the error depends on the distribution of the eigenvalues of $C^{-1}A$, see, for example, [30]. An upper bound on the error can be derived in terms of κ, see [17, p. 187]:

$$\|x_k - x^*\|_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|x^*\|_A.$$

For more details and derivation of the preconditioned conjugate gradients algorithm, see, for example [9, 14].

4. Formulation of the Iterative Method. The key to a fast preconditioned conjugate gradients method is thus the design of an efficient preconditioner, which is not too expensive to apply and gives a low value of the condition number κ. Of course, these goals contradict each other: the lowest possible $\kappa = 1$ requires that $C = A$. One thus needs to strike a balance between the two objectives.

4.1. Preconditioning as a Transformation. Let $C^{-1} = MM^T$ for some M. Then the system (1) is equivalent to

$$\tilde{A}\tilde{x} = \tilde{b}, \tag{8}$$

where

$$\tilde{A} = MAM^T, \qquad \tilde{b} = Mb. \tag{9}$$

After solving for $\tilde{x}$ from (8), the solution x can be obtained from

$$x = M^T\tilde{x}. \tag{10}$$

The preconditioned conjugate gradients algorithm with A and C as above is equivalent to the conjugate gradients without preconditioning (that is, $C = I$) applied to $\tilde{A}$ in the place of A (see, for example, [14]).

4.2. Description of the Algorithm. The present method uses a transformation of the form (9) *and* $C \neq I$. The transformation matrix will be denoted $\bar{M}$ to avoid conflict with the notation in (9). Such a method is of course mathematically equivalent to one which uses only transformation or only preconditioning, but the use of both transformation and preconditioning results in extra flexibility in the formulation of the method and in its implementation.

The scheme of the whole iterative method is outlined in Figure 2. We now give a more complete description of the steps in Figure 2.

Step 1. Transformation by incomplete elimination. This step consists of a sequence of the following elementary block transformations. Consider the matrix A written in a 3×3

FIG. 2. *The iterative method*

Step 1. Transform the system $Ax = b$ to $\tilde{A}\tilde{x} = \tilde{b}$, with $\tilde{A} = \bar{M}A\bar{M}^T$, $\tilde{b} = \bar{M}b$. Keep the transformation matrix $\bar{M}$ (in a product form).

Step 2. Select the preconditioner C as a generalized block diagonal of the transformed matrix $\tilde{A}$.

Step 3. Assemble the matrix C and compute the Cholesky decomposition $C = LL^T$.

Step 4. Apply the preconditioned conjugate gradients method to the transformed system $\tilde{A}\tilde{x} = \tilde{b}$, using the known Cholesky decomposition of C from Step 3.

Step 5. Recover x by back transformation $x = \bar{M}^T\tilde{x}$.

symbolic block form

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix} \tag{11}$$

and the transformation matrix X, defined by

$$X = \begin{pmatrix} I & 0 & 0 \\ -A_{21}A_{11}^{-1} & I & 0 \\ 0 & 0 & I \end{pmatrix}. \tag{12}$$

Using the symmetry of A, we have

$$XAX^T = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} - A_{21}A_{11}^{-1}A_{12} & A_{23} - A_{21}A_{11}^{-1}A_{13} \\ A_{31} & A_{32} - A_{31}A_{11}^{-1}A_{12} & A_{33} \end{pmatrix}. \tag{13}$$

Thus, the first block of variables was eliminated from the second block but not from the third.

The block form (11) is to be understood in a generalized sense; the blocks of variables can form an arbitrary disjoint decomposition of the set $\{1, \ldots, N\}$.

Note that one could eliminate the first block of variables from the third as well as the second block; a succession of N such steps, each with the first block consisting of just one variable eliminated from all the rest, is equivalent to Gussian elimination, and reduces A to a diagonal matrix. In (13), we intentionally do not eliminate the first block completely from the rest of the system (that is, we do not force the blocks A_{31} and A_{13} to be zero), because we do not wish to change the block A_{33} and inevitably introduce new nonzero entries there.

This incomplete elimination was called *partial orthogonalization* of degrees of freedom in the first block to those in the second block in [18, 20], based on an interpretation of the matrix XAX^T as the stiffness matrix with respect to a new basis.

The algorithm proceeds by taking these incomplete elimination steps in the following succession:

1. All interior degrees of freedom corresponding to basis functions (6) with $m+n+l > 0$ are eliminated from the system.
2. For each face of the finite element structure, the face degrees of freedom (5) with $n + m > 0$ are eliminated from the degrees of freedom (4) with $n > 0$ for the adjacent edges.

This process does not introduce any fill-in, and it results in a transformed stiffness matrix with zero blocks corresponding to the face-edge interactions between basis functions of higher order than quadratic. There are, however, still nonzero blocks corresponding to the face-face and edge-edge interactions.

The transformation matrix $\bar{M}$ in Step 1 in Figure 2 is kept as the collection of the elementary transformation matrices X from (12), to be applied in the order in which they are generated.

Step 2. Selection of block diagonal preconditioner and adaptive strategies for distorted elements. Step 1 results in the system $\tilde{A}\tilde{x} = \tilde{b}$, which is equivalent to the original system, and has most face-edge interactions eliminated. The next step is to ignore as many other interactions as possible. In the simple case (no special treatment of distorted elements), the preconditioner C is selected to be the block diagonal of $\tilde{A}$ with the following blocks (some of which may be empty):

1. The first block consists of all degrees of freedom corresponding to linear and quadratic basis functions, that is, nodal basis functions (3)–(6) with $n = m = l = 0$.
2. For each edge, one block of the remaining degrees of freedom, corresponding to functions (4) with $n > 0$.
3. For each face, one block of the remaining degrees of freedom, corresponding to functions (5) with $n > 0$ or $m > 0$.
4. For each element, one block of the remaining degrees of freedom, corresponding to functions (6) with $n > 0$, $m > 0$, or $l > 0$.

The reason for including variables corresponding to the quadratic functions in the first block was explained in [18]. For three-dimensional problems, the convergence of the method suffers if the first block consists of linear degrees of freedom only. The first diagonal block of C is actually the stiffness matrix of the same problem using quadratic elements.

The selection of C as above gives satisfactory convergence rates if the elements are not distorted. In essence, this selection of C ignores all face-face, face-edge, and edge-edge interactions not resolved in Step 1, except for the quadratic functions included in the first block. For distorted elements, ignoring all those interactions gives poor convergence rates. A heuristic procedure has been devised to enlarge adaptively the matrix C to include some of those interactions. For example, for an element with the aspect ratio 1 : 20 : 20, the interaction between the long edges of a side with aspect ratio 1 : 20 is quite strong and should be included in C. The transformation in Step 1 has the property that if A_K is the local stiffness matrix of element K understood to be embedded in a zero matrix of the size

FIG. 3. *Adaptive construction of the preconditioner for distorted elements*

Step 1. Given element K and its transformed local stiffness matrix $\tilde{A}_K$, let V be the set of all edges and faces of K. Initialize a disjoint decomposition of the degrees of freedom in K into
1. all linear and quadratic degrees of freedom;
2. one block of the remaining degrees of freedom for each face and each edge.

Step 2. Estimate the relative condition number of $\tilde{A}_K$ and C_K for the current decomposition of degrees of freedom of element K. If acceptable, exit.

Step 3. Based on estimated strength of interaction between the existing blocks, coarsen the block decomposition of degrees of freedom of K by merging strongly connected blocks. If there are too few blocks already, let $C_K = \tilde{A}_K$ and exit. Otherwise go to Step 2.

of the global stiffness matrix A, then the *transformed local stiffness matrix*

$$\tilde{A}_K = \bar{M} A_K \bar{M}^T \tag{14}$$

has nonzero entries only for the degrees of freedom of element K. The adaptive heuristic procedure selects a local preconditioner C_K based on $\tilde{A}_K$. C is then assembled from C_K just as the global stiffness matrix $\tilde{A}$ could be assembled form the local matrices $\tilde{A}_K$:

$$\tilde{A} = \sum_K \tilde{A}_K, \qquad C = \sum_K C_K. \tag{15}$$

The adaptive procedure is outlined in Figure 3. It uses an estimator of the relative condition number or A_K and C_K, based on the theoretical principles introduced in §4.3 below. Note that the adaptive procedure can give up in some cases and select $C_K = \tilde{A}_K$.

Step 3. Assembly and Cholesky decomposition of C. The matrix constructed in Step 2 is well suited for solution by an envelope (variable band) method. If there are no distorted elements, C consists of a number of dense diagonal blocks for the faces and edges, and of one large, sparse block for the linear and quadratic degrees of freedom. An envelope method will take advantage of the block diagonal form of C automatically. If additional interactions are included in C, then C will not have a block diagonal structure, but, because distorted elements tend to occur in clusters, an ordering of variables can be found which results in a reasonably small envelope, and so the Cholesky decomposition of C can still be calculated efficiently.

Step 4. Preconditioned conjugate gradients. The method from Figure 1 is used with the transformed matrix $\tilde{A}$ playing the role of A.

Step 5. Back transformation of the solution. To apply (10), the matrices X^T from (13) are applied in the opposite order than in which they were generated.

4.3. Theoretical Results and Estimates of Convergence. Let inequality between symmetric matrices mean that their difference is positive semidefinite. The principal theoretical observation here is that if for all elements K,

$$m_1 C_K \leq \tilde{A}_K \leq m_2 C_K, \tag{16}$$

then by summation over all elements and using (15), one has

$$m_1 C \leq \tilde{A} \leq m_2 C, \tag{17}$$

which implies that the relative condition number of $\tilde{A}$ and C is at most m_2/m_1. The inequality (16) can be satisfied with $m_1 > 0$, $m_2 < \infty$ if and only if $\tilde{A}_K$ and C_K have the same nullspace [19]. This is fortunately the case here, because the nullspace of A_K and thus also of $\tilde{A}_K$ consists of vectors corresponding to small rigid body motions, which form a subspace of all linear functions. The ratio m_2/m_1 can be calculated numerically. If the nullspaces of $\tilde{A}_K$ and C_K coincide and $\lambda_{1,K}$ and $\lambda_{2,K}$ are the minimal nonzero eigenvalue and the maximal eigenvalue of the generalized eigenvalue problem

$$\tilde{A}_K u_K = \lambda C_K u_K, \tag{18}$$

then (17) holds with

$$m_1 = \min_K \lambda_{1,K}, \qquad m_2 = \max_K \lambda_{2,K},$$

see [19]. This simple observation is used in the present method as follows:

1. The specific version of the partial elimination algorithm and the construction of C described in §4.2 were selected based on the calculation of $\lambda_{2,K}/\lambda_{1,K}$ in many representative situations [18].
2. The adaptive selection of C_K in Figure 3 uses an estimate of $\lambda_{2,K}/\lambda_{1,K}$ from submatrices of $\tilde{A}_K$ and C_K. Calculating $\lambda_{1,K}$ and $\lambda_{2,K}$ exactly would be prohibitively expensive.

For a closely related method for two-dimensional problems, it can be shown that [4]

$$m_2/m_1 \leq \text{ const. } \log^2 p.$$

For elements with high aspect ratios in two dimensions, the condition number $\lambda_{2,K}/\lambda_{1,K}$ grows like the square of the aspect ratio, and making C_K a larger block diagonal part of $\tilde{A}_K$ was proved to inhibit this growth [15, 21].

5. Computational Results. In this section, we describe the test implementation of the algorithm and report on the results of computations.

5.1. Notes on a Test Implementation. The purpose of the test implementation was to assess the viability of the method and provide a tool for experimentation. The emphasis was on reliability of the results, flexibility, and programmer's convenience rather than efficiency, although gross inefficiencies were avoided wherever possible.

The code was written in FORTRAN 77 with the use of the C language macro preprocessor, and it has about 10,000 lines of source code before macro expansions. The code makes also use of routines from the packages LINPACK, EISPACK, CGCODE, and SPARSPAK.

The program uses a simple stack memory management scheme so that it can handle problems of an arbitrary size. Element types are described using tables given as input data. Arbitrary elements in any number of dimensions are allowed. The topology and connectivity data are handled in in-core data structures.

The global stiffness matrix is kept as a collection of local stiffness matrices and is never assembled. The transformations are reflected in the data of the local stiffness matrices, and only the pieces of the global stiffness matrix needed to calculate the transformations are temporarily assembled. The local stiffness matrices are kept in temporary files and/or memory buffers. The data structure for the transformation matrix $\bar{M}$ is kept in several files.

The preconditioner C is kept in core. For the Cholesky decomposition of C, the variables are first reordered using SPARSPAK implementation of the RCM algorithm [13]. A special version of the envelope Cholesky had to be used, because the envelope algorithm from [13] resulted in excessive paging. The new routine splits the Cholesky decomposition into a number of stages. At each stage, the routine accesses only a "window" in the envelope at random and the rest of the data is processed sequentially. The size of the window is given as a parameter, which should be smaller than the amount of RAM available to the program.

Keeping the matrix C in core limits the size of the problem that can be handled by the test implementation. This was no problem on a dedicated SUN-4 computer, where disk space can be easily converted into virtual memory. A production code based on this method could identify the diagonal blocks of C, store them in a file, and use an out-of-core solver for the largest blocks if necessary. See also §6.6 for other possibilities.

5.2. Comparison with a Multifrontal Direct Solver. The results of computational tests are in Table 1. In all tests problems, the material was isotropic with Poisson ratio 0.3. The relative precision for the stopping criterion was chosen to be 10^{-4}, which is better than the precision of the finite element approximation. The relative precision in the energy norm was monitored using (7), and it was better than 10^{-3} in all cases.

The direct solver was the multi-frontal solver distributed with MSC/PROBE. The data for the largest problem (crank2, 376 elements) and $p = 8$ could not be obtained because of disk space limitations. Also, the disk space required by the direct solver for this problem at $p = 6$ and $p = 7$ exceeded available space, while the iterative solver could be run without any problems.

The last two columns of Table 2 give a comparison of CPU time and disk space/memory requirements for the direct and iterative solver, derived from the data in Table 1. The

TABLE 1
Comparison of Direct and Iterative Solver

				CPU Times (seconds)				Memory and Disk (MB)		
Problem	p	NDOF	IT	STIFF	ITER	RHS	DIR	STIFF	ITER	DIR
block18w	3	600	15	36	15	5	17	0.50	0.59	0.80
	4	1065	21	115	35	15	62	1.25	1.50	2.06
	5	1755	25	409	79	34	197	2.83	3.30	4.64
	6	2724	30	1325	208	80	474	5.85	6.90	9.47
	7	4026	35	3398	860	406	1217	11.26	13.68	17.80
	8	5679	43	8311	2305	1033	2638	20.25	25.36	31.28
block64	3	2175	7	150	80	13	252	2.27	2.56	5.80
	4	3795	8	523	213	54	1027	5.53	6.48	15.21
	5	6135	15	1752	537	214	3202	12.09	13.68	33.64
	6	9387	18	5290	1329	553	9391	24.30	27.42	75.35
	7	13743	20	13149	3131	1179	23391	45.67	52.62	147.68
	8	19395	21	31534	7173	2241	50903	81.14	96.40	253.15
crank	5	9687	25	6190	620	215	3314	18.02	20.72	38.86
	6	15090	29	17090	1340	460	8725	36.87	42.63	77.37
	7	22386	38	40241	3536	1085	20368	70.29	83.46	144.48
	8	31758	45	79136	12631	4841	44418	125.80	154.08	253.48
crank2	3	10992	14	2170	1943	223	5662	12.54	14.26	51.37
	4	19671	19	7144	6206	851	17996	30.77	36.13	124.31
	5	32478	25	20486	7529	1807	66743	67.78	76.98	309.11
	6	50541	27	56375	12285	4189		137.22	155.89	
	7	74988	32	119787	26490	9741		259.34	301.60	

Test problems:

block18w Block of $3 \times 3 \times 2$ elements, 14 bricks and 4 wedges. SUN SPARC 1, 12MB memory.

block64 Block of $8 \times 8 \times 8$ brick elements. SUN SPARC 1, 12MB memory.

crank Model of a crankshaft, 107 elements. SUN 4/280, 96MB memory.

crank2 Model of a crankshaft, 376 elements. SUN 4/280, 96MB memory.

p The degree of elements used.

NDOF Order of the system solved.

IT Number of iterations for estimated relative precision 10^{-4} in the ℓ^2 norm. The estimated relative precision in the energy norm from (7) was better than 10^{-3} in all cases.

CPU times are sums of user and system CPU times in seconds. The system time reflects I/O activity.

STIFF The CPU time for generating the stiffness matrix data by MSC/PROBE.

ITER The total CPU time for the iterative solver.

RHS The CPU time to solve for an additional right-hand side.

DIR The CPU time for the multi-frontal solver used in MSC/PROBE.

Memory and Disk Usage is the total size of all files and/or dynamically allocated virtual memory:

STIFF The total size of all local stiffness matrices.

ITER The total requirements of the iterative solver, including the local stiffness matrices.

DIR The total requirements of the direct solver. (The direct solver does not store the local stiffness matrices.)

TABLE 2
Relative Performance of Iterative Solver

			Proportions of CPU time in iterative solver			Improvement over direct solver	
Problem	p	NDOF	Trans-formation	Decompo-sition of C	Iterations	CPU time	Disk and memory
block18w	3	600	0.47	0.21	0.32	1.1	1.3
	4	1065	0.35	0.22	0.43	1.8	1.4
	5	1755	0.47	0.11	0.43	2.5	1.4
	6	2724	0.57	0.05	0.38	2.3	1.4
	7	4026	0.51	0.02	0.47	1.4	1.3
	8	5679	0.54	0.01	0.45	1.1	1.2
block64	3	2175	0.25	0.59	0.16	3.1	2.3
	4	3795	0.20	0.55	0.26	4.8	2.3
	5	6135	0.37	0.23	0.40	6.0	2.5
	6	9387	0.47	0.12	0.42	7.1	2.7
	7	13743	0.57	0.06	0.38	7.5	2.8
	8	19395	0.66	0.03	0.31	7.1	2.6
crank	5	9687	0.44	0.21	0.35	5.3	1.9
	6	15090	0.55	0.11	0.34	6.5	1.8
	7	22386	0.64	0.05	0.31	5.8	1.7
	8	31758	0.59	0.02	0.38	3.5	1.6
crank2	3	10992	0.07	0.82	0.11	2.9	3.6
	4	19671	0.04	0.82	0.14	2.9	3.4
	5	32478	0.12	0.64	0.24	8.9	4.0
	6	50541	0.27	0.39	0.34		
	7	74988	0.38	0.26	0.37		

The test problems are same as in Table 1.

Proportions of CPU are the proportions of the CPU time the iterative method spends in the main parts of the algorithm. The transformation time includes the adaptive procedure (Figure 3). The back transformation (Step 4 in Figure 2) is included in the time of the iterations.

Improvement over direct solver gives the ratios of the CPU time and disk space/memory requirements of the direct solver compared to those of the iterative solver.

iterative method was faster in all cases. The biggest speedup observed was 8.9 for the crank2 test problem and $p = 5$.

The amount of disk space and memory needed by the iterative solver is only slightly more than for all local stiffness matrices. On the other hand, the disk space requirements of the direct solver grow explosively due to fill-in.

The middle columns of Table 2 give a breakdown of the CPU time for the main parts of the iterative solver. For problems with a large number of elements, the memory and time requirements of the iterative solver are dominated by the decomposition of the preconditioner C, which essentially involves a direct solution of the same problem on quadratic elements. The highest proportion of time spent in the decomposition was for the largest test problem (crank2). The proportion of the time spent in the decomposition decreases with larger p, because C then consists of a smaller number of nonzeros of $\tilde{A}$. Only the iteration part has to be repeated for the solution for an additional right-hand side.

6. Extensions and Future Research. This section summarizes some directions for future development.

6.1. Nearly Incompressible Materials. The p-version finite element method is known to avoid locking for nearly incompressible materials [31] and good engineering solutions can be obtained by post-processing [25]. The iterative method deteriorates for nearly incompressible materials and requires more iterations and/or a bigger matrix C, which is more expensive to decompose. One possible way to handle this problem is to use for preconditioning elements with artificial pressure or equivalent nonconforming elements.

Highly anisotropic materials pose a similar problem, which could be possibly handled by adding more additional variables.

In any case, the adaptive procedure (Figure 3) should be developed to recognize such instances reliably and gradually switch to a direct solver, putting eventually $C = A$. The transformation should be of course avoided in such extreme cases. The performance of such a method would then depend on the difficulty of the problem; the method would be fast for nearly isotropic and well compressible materials, while it would automatically reduce to a direct solver for difficult problems.

6.2. Computation of Eigenvalues. The generalized eigenvalue problem

$$Au = \lambda Bu, \tag{19}$$

with B the mass matrix, is commonly solved using the Lanczos method, which requires solving repeatedly linear systems with the matrix $A - \mu B$ for given shifts μ [10]. These systems need to be solved exactly, which would make iterative methods prohibitively expensive. However, combining inverse iterations with preconditioned conjugate gradient iterations may be a way to adapt an iterative method for the solution of linear systems to solve (19), see [28, 29].

6.3. Considerations for Parallel Architectures. The present method is immediately parallelizable on the element level except for the decomposition of C. The transformation (Steps 1) requires the communication of parts of the matrices A_K only between

adjacent elements. (This results in complicated programming, so other possibilities are being investigated, see §6.5.) The adaptive procedure (Step 2) is completely parallel. Preconditioned conjugate gradients (Step 4) require evaluation of inner products and of the matrix-vector products $\tilde{A}\tilde{x}$ and $C^{-1}r$. For the matrix-vector product $\tilde{A}\tilde{x}$, one only needs to communicate the values of $\tilde{A}_K\tilde{x}_K$ between adjacent elements. The back transformation (Step 5) can use tranformation data stored locally.

The repeated evaluations of $C^{-1}r$ in Step 4 and the preceding decomposition of C in Step 3 are a bottleneck to parallelism. In a parallel computing environment, it may be more efficient to use several inner iterations of an easily parallelizable method to evaluate an approximation to $C^{-1}r$ in each step of preconditioned conjugate gradients rather than decompose C. For experience with parallel implementations of a related method in two dimensions, see [4, 5].

6.4. Combination with Incomplete LU Decomposition (ILU). The transformation by incomplete elimination is based on a similar principle as the well-known preconditioner by ILU. Incorporating the principles used here into the ILU framework could result in a superior method.

6.5. Element Parallel Transformation. The transformation by incomplete elimination operates on the *global* stiffness matrix, which requires significant communication between elements and results in somehow complicated programming. A version of the method is considered in which the transformation in a local stiffness matrix depends on the data of that matrix only.

6.6. Multilevel Process. The method requires in each step the solution of an auxiliary system, which is the same problem with quadratic elements. The time for Cholesky decomposition of this subsystem dominates the computation for a large number of elements and limits the size of the problem if the decomposition is done in core. A fast iterative method for the solution of this auxiliary problem should be considered for very large problems, involving even smaller auxiliary problems. Another possibility may be to use a version of ILU, possibly combined with the incomplete elimination into one incomplete factorization process, cf., §6.4.

6.7. Application to Other Problems. The approach presented here can be applied to a large class of symmetric, positive definite linear problems arising from the finite element method. The basic requirements for its applicability and efficiency are:

1. The elements have a large number of degrees of freedom, which are naturally associated with the interior, faces, edges, and nodes of the elements.
2. A small subset of basis functions can be found that spans the nullspace of the local stiffness matrix. The system of equations derived from this subset of the basis functions is then solved in each iteration.

The p-version finite element method satisfies these requirements. For other finite element methods, condition 1 can be satisfied by forming *macroelements* by aggregating a number of existing elements, and condition 2 by a transformation of the degrees of freedom of the macroelement.

Currently considered areas of application include the Maxwell equations of electromagetics discretized by finite elements in MSC/EMAS.

Complex problems with Hermitean positive definite matrices do not present any significant problems, and the existing theory applies. The principles and algorithms presented in this paper can be also applied to nosymmetric and indefinite problems, but then the theory does not apply any more and the efficiency of the preconditioning in specific cases has yet to be investigated. Special versions of the conjugate gradients algorithm can be used for such problems [2].

7. Conclusion. The iterative method presented in this paper uses special properties of the system of linear equations arising from finite elements in three dimensions to achieve fast convergence and high efficiency. The method outperforms a state of the art direct sparse solver used in MSC/PROBE for large problems with complicated geometries and distorted elements.

Most computations parallelize naturally on the element level. Possible extensions of the method include the solution of eigenvalue problems, linear systems arising from singular perturbation problems such as elasticity for nearly incompressible or highly anisotropic materials, and general finite element systems for structural mechanics and the equations of electromagnetics.

REFERENCES

[1] S. Ashby, T. A. Manteuffel, and W. Joubert, *CGCODE.* Collection of conjugate gradients FORTRAN routines, Los Alamos National Laboratory, 1988.

[2] S. Ashby, T. A. Manteuffel, and P. E. Saylor, *A taxonomy for conjugate gradient methods,* SIAM J. Numer. Anal., 27 (1990), pp. 1542–1568.

[3] C. C. Ashcraft, R. G. Grimes, J. G. Lewis, B. W. Peyton, and H. D. Simon, *Progress in sparse matrix methods for large linear systems on vector supercomputers,* Int. J. of Supercomputer Applications, 1 (1987), pp. 10–30.

[4] I. Babuška, A. W. Craig, J. Mandel, and J. Pitkäranta, *Efficient preconditioning for the p-version finite element method in two dimensions.* SIAM J. Numer. Anal., to appear.

[5] I. Babuška and H. C. Elman, *Some aspects of parallel implementation of the finite element method on message passing architecture,* J. Comput. Appl. Math., 27 (1989), pp. 157–187.

[6] I. Babuška and M. Suri, *The p- and h-p versions of the finite element method, An overview,* Comput. Methods Appl. Mech. Engrg., 80 (1990), pp. 5–26. International Conference on Spectral and High Order Methods for partial Differential Equations, Como, Italy, June 1989.

[7] I. Babuška, B. A. Szabó, and I. N. Katz, *The p-version of the finite element method,* SIAM J. Numer. Anal., 18 (1981), pp. 515–545.

[8] J. H. Bramble, J. E. Pasciak, and A. H. Schatz, *The construction of preconditioners for elliptic problems by substructuring, I,* Math. Comp., 47 (1986), pp. 103– 134.

[9] 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, J. R. Bunch and D. J. Rose, eds., Academic Press, 1976, pp. 309–332.

[10] J. Cullum and R. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Volume 1: Theory,* Birkhaüser, Boston, 1985.

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

[12] S. Foresti, G. Brussino, S. Hassanzadeh, and V. Sonnad, *Multilevel solution method for the p-version of finite elements*, Computer Physics Comm., 53 (1989), pp. 349–355.

[13] J. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[14] G. H. Golub and C. F. Van Loan, *Matrix Computations*, John Hopkins University Press, second ed., 1989.

[15] G. S. Lett, *Iterative Solver for the p-Version Finite Element Method with Thin Rectangular Elements*, PhD thesis, Department of Mathematics, University of Colorado at Denver, 1990.

[16] J. Lewis and D. Pierce, *Recent research in iterative methods at Boeing*, Computer Physics Communications, 53 (1989), pp. 213–21. Practical Iterative Methods for Large Scale Computations, Proceedings of the Minnesota Supercomputer Institute Workshop, 23-25 Oct. 1988.

[17] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, New York, 1973.

[18] J. Mandel, *Hierarchical preconditioning and partial orthogonalization for the p-version finite element method*, in Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, T. F. Chan, R. Glowinski, J. Periaux, and O. B. Widlund, eds., Philadelphia, 1990, SIAM, pp. 141–156.

[19] ———, *Iterative solvers by substructuring for the p-version finite element method*, Comput. Methods Appl. Mech. Engrg., 80 (1990), pp. 117–128. International Conference on Spectral and High Order Methods for Partial Differential Equations, Como, Italy, June 1989.

[20] ———, *Two-level domain decomposition preconditioning for the p-version finite element method in three dimensions*, Int. J. Numer. Methods Engrg., 29 (1990), pp. 1095–1108.

[21] J. Mandel and G. S. Lett, *Domain decomposition preconditioning for p-version finite elements with high aspect ratios*. Applied Numer. Anal., to appear.

[22] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473–497.

[23] S. F. McCormick, ed., *Multigrid Methods*, vol. 3 of Frontiers in Applied Mathematics, SIAM, Philadephia, 1987.

[24] J. Meijerink and H. Van der Vorst, *An iterative solution method for linear equations systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[25] B. Szabó, I. Babuška, and B. Chayapathy, *Stress computations for nearly incompressible materials by the p-version of the finite element method*, Int. J. Num. Meth. Engrg, 28 (1989), pp. 2175–2190.

[26] B. A. Szabó, *PROBE Theoretical Manual*, Noetic Technologies, St. Louis, MO, 1985.

[27] B. A. Szabó, *The p- and h-p-version of the finite element method in solid mechanics*, Comput. Methods Appl. Mech. Engrg., 80 (1990), pp. 185–196. International Conference on Spectral and High Order Methods for partial Differential Equations, Como, Italy, June 1989.

[28] D. B. Szyld, *A Two-level Iterative Method for Large Sparse Generalized Eigenvalue Calculations*, PhD thesis, New York University, 1983.

[29] ———, *Criteria for combinig inverse and Rayleigh quotient iteration*, SIAM J. Numer. Anal., 25 (1988), pp. 1369–1375.

[30] A. van der Sluis and H. Van der Vorst, *The rate of convergence of conjugate gradients*, Numer. Math., 48 (1986), pp. 543–560.

[31] M. Vogelius, *An analysis of the p-version of the finite element method for nearly incompressible materials*, Numer. Math., 41 (1983), pp. 39–53.