WAPD-T-3094
CONF-960249--1

PRECONDITIONED GRADIENT METHODS FOR SPARSE LINEAR SYSTEMS FOR 'VERY LARGE' STRUCTURAL PROBLEMS.

I. K. Abu-Shumays, D. N. Hutula, J. J. Haan and G. T. Myers

December 1995

DE-AC11-93PN38195

BETTIS ATOMIC POWER LABORATORY WEST MIFFLIN, PENNSYLVANIA 15122

Operated for the U. S. Department of Energy
by WESTINGHOUSE ELECTRIC CORPORATION

WAPD-T-3094

Preconditioned Gradient Methods for Sparse Linear Systems for 'Very Large' Structural Problems

I. K. Abu-Shumays, D. N. Hutula, J. J. Haan, and G. T. Myers

ABSTRACT

This paper deals with background and practical experience with preconditioned gradient methods for sparse linear systems for 'very large' structural problems.

The competitive advantage of applying preconditioned conjugate gradient methods to the solution of very large sparse linear systems which arise in structural problems is demonstrated. Such systems, while being symmetric positive definite, are typically not diagonally dominant and are based on applications of the finite element method over irregular (unstructured) mesh subdivisions. **In addition, such very large systems are often ill-conditioned.** As a consequence, attempts prior to 1990 to apply iterative gradient methods to structural problems were marginally successful and these methods were viewed in the non-academic world as inferior to direct solution methods. This is due in part to the advanced state-of-the-art of direct solution methods. Iterative methods were previously applied to much smaller structural problems and the speed advantage as compared to direct methods was not consistently realized.

The conjugate gradient method with diagonal preconditioning (CG/D) is demonstrated to substantially increase the size of structural problems that can be analyzed, significantly reduce computer storage requirements, and cut computing cost; thus allowing for much more detailed modeling and increased engineering efficiency. For one case for a structural system with 396,087 unknowns, the conjugate gradient method with diagonal preconditioning is demonstrated to be a factor of sixty faster than the direct method. For certain problems, however, the number of iterations required by the CG/D method is excessive and improved methods are needed.

A stand-alone iterative solver research computer program was developed to evaluate the merits of various matrix preconditioners. A matrix preconditioner based on a shifted incomplete Cholesky factorization algorithm was demonstrated to be superior to other choices. The stand-alone program incorporates an effective data management strategy which utilizes disk and solid state auxiliary computer storage devices to make it possible to efficiently solve excessively large structural problems on state-of-the-art vector and parallel computers.

The background of gradient methods, algorithms for their implementation, and practical experience in their applications to structural problems are presented.

1. Introduction

Application of the finite element method to structural problems of general interest typically yields linear systems of equations with between 100,000 and 6 million unknowns or degrees of freedom. The limitation on the size is dictated by the computer resources available. The coefficient matrix **A** typically has up to 243 non-zero elements in each row and each column for 20 node brick elements. One undesirable feature of the system of equations of interest is the fact that the elements of **A** can vary widely in magnitude and the matrix **A** is often mathematically ill-conditioned. Thus care should be exercised in the selection of solution methods.

Until recently, the structural community employed highly efficient direct methods for solving sparse linear systems corresponding to large three-dimensional structural problems. The direct method developed by Dr. D. N. Hutula, while requiring extensive computer storage is extremely efficient and runs essentially at optimum speed of the Cray Y-MP and C-90 supercomputers.

A team was formed at Bettis in the Fall of 1990 to investigate and implement iterative methods for the solution of structural problems which are one or two orders of magnitude larger than is possible by direct methods.

There is extensive experience in applying iterative methods to coupled systems of diffusion type elliptic partial differential equations and to fluid flow problems. The elliptic diffusion systems of interest are typically modeled over a mesh structure which is rectangular in logical space and the corresponding systems of linear equations are symmetric positive definite (i.e., possess real and positive eigenvalues) and are in addition diagonally dominant. Preconditioned conjugate gradient iterative methods [1,7] have been demonstrated to be very effective for such problems. The main idea of a preconditioning method is first to obtain a matrix which is relatively easy to invert and is a good approximation to the coefficient matrix of the system under consideration, and second, to apply this as a splitting matrix or preconditioner to construct fast converging conjugate gradient polynomial iterative solution methods.

This paper summarizes our success in adapting preconditioned conjugate gradient methods to structural problems. Success with iterative methods for structural problems has been achieved in spite of the fact that the corresponding systems of equations lack nice mathematical properties.

To date, work on iterative solvers at Bettis has involved four phases. In the first phase, the Bettis team developed and tested the effectiveness of several new and old variations of preconditioned conjugate gradient and conjugate residual iterative methods in stand-alone computer programs. The initial test computer programs were for problems that fit in the central memory of the Cray Y-MP and Cray C-90 supercomputers. The diagonally scaled conjugate gradient (CG/D), and conjugate residual (CR/D) methods in particular, were identified as the methods of choice for exceedingly large problems. The preconditioned conjugate gradient method with a preconditioner based on shifted Cholesky incomplete factorization (PCG/IF) has been identified as the method of choice for moderate and very large size structural problems. This method was first conceived by Manteuffel [9] and variations on it were later introduced by the present authors and others. A generalization of this method is discussed below.

The diagonally scaled conjugate gradient (CG/D) method was implemented in a special stand-alone computer program which utilizes disks and solid state auxiliary storage devices to handle problem sizes too large to fit in the central memory of Cray computers, and was demonstrated to cut computer storage requirements by an order of magnitude as compared to the direct method previously in use. The initial implementation was not optimal, but was intended to demonstrate the effectiveness of the method when applied to practical models of structural problems. For a section of a multiple-connected structure with 396,087 degrees of freedom (unknowns), the simplest diagonally scaled conjugate gradient iterative method reduced the storage required by a factor of 48 and the solution time of 22.5 hours to 2.5 hours on the Cray Y-MP vector supercomputer. For other test problem configurations, solution time and storage space reductions were not as large, but in all cases storage space reduction varied between one and three orders of magnitude, thus allowing much larger models to be developed. Allowing models containing greater detail also improves the user understanding of the results.

In the second phase, D. N. Hutula implemented an element-by-element (EBE) scheme to the diagonally scaled conjugate gradient and conjugate residual methods. The main idea is to avoid explicitly constructing the coefficient matrices of the systems to be solved and instead perform the needed computation by repeatedly assembling the contributions of individual structural elements. Since individual elements are independent of each other, the computations were highly vectorized, increasing the speed of the calculations. As a consequence, the current EBE implementation of the iterative solver requires a factor of 20 less computer storage and runs four times faster than the stand-alone solver. The EBE iterative solver was applied to the 396,087 degree of freedom problem which required 22.5 hours to run on the Cray Y-MP computer using the direct method. The storage requirements for this problem were reduced to under three million locations, a factor of over 1000, and the computing time was reduced to less than one-half hour, a factor of over 60.

The Cray Y-MP being used has eight central processing units (8 CPUs), 64 Megawords (Mw) of central memory and a clock cycle time of approximately 6 nanoseconds (ns). The Cray C-90 has 16 CPUs, 256 Mw of central memory and a clock cycle time of approximately 4 ns. Each Cray has a 512 Megawords solid state storage device (SSD) and a number of disks for auxiliary storage. The data transfer rates are approximately (a) 1,000 Megabytes (MB) per second between SSD and central memory on the Cray Y-MP, (b) 1900 MB per second between SSD and central memory on the Cray C-90, (c) 12 MB per second between disks and central memory on the Cray Y-MP and (d) 20 MB per second between disks and central memory on the Cray C-90. It is very essential for efficiency consideration to be able to overlap data transfer with computations. The programs implemented in this work are designed for a single CPU and allow for utilizing SSD and up to six distinct disks. Opening files on distinct disks and transferring data to and from SSD and from these disks simultaneously makes it possible to substantially increase the effective data transfer rate. As a consequence, the program implemented for both the direct method and the EBE iterative method run for large problems at the rate of between 800 and 900 MFLOP (Million Floating Point Operations Per Second) for large problems on the Cray C-90. This compares very well with the theoretical speed of a little over 1,000 MFLOPS.

In the third phase, D. N. Hutula and B. E. Wiancko developed an elegant partitioning scheme and constructed a parallel version of the element-by-element iterative solver suitable for the Intel iPSC/860 massively parallel computer with 64 processors each having 64 megabytes of memory. Substantial reductions in computer storage requirements and in running time have been achieved for the parallel version over the iterative serial version of the program. Problems with up to three million finite elements and 36 million degrees of freedom are possible on the Intel computer and speeds comparable to those of a Cray-C90 have been achieved with 64 processors on the iPSC/860 machine. Their program was also adapted to run on an Intel Paragon, though the Paragon had less memory than the iPSC/860.

In spite of the successes with the diagonally preconditioned conjugate gradient solvers, the underlying method requires too many iterative steps for solution convergence for certain problems, such as continuum element representation of shell problems. Consequently, in the fourth phase, the authors searched for improved iterative solvers suitable for difficult problems which arise in several structural applications.

The effectiveness of the preconditioned conjugate gradient (PCG) method with shifted Cholesky incomplete factorization (IF) preconditioners was first demonstrated for solving difficult problems which are small enough to fit in the central memory of the current generation of Cray supercomputers. In the Fall of 1992, an extended version of this PCG/IF method was then implemented in the stand-alone computer program which utilizes auxiliary storage. Numerical tests summarized below demonstrate that the PCG/IF method can result in a factor of 2.5 or more reduction in computing cost over the diagonally preconditioned conjugate gradient CG/D method. The PCG/IF method requires 50% more computer storage than the CG/D method, and its initial implementation in the stand-alone program assumes that the coefficient matrix for the system of equations for a structural model is assembled in advance. An element-by-element analogue of the PCG/IF method is needed for practical application.

2. <u>Preconditioned Conjugate Gradient Methods</u>

Consider the linear system

$$Ax = b, \tag{2.1}$$

where the coefficient matrix A is symmetric positive definite (SPD). It can be easily shown that solving the system (2.1) when A is SPD is equivalent to minimizing

$$f(x) = \frac{1}{2}x^TAx - x^Tb, \qquad E(x) = (x-\bar{x})^T A(x-\bar{x}), \tag{2.2}$$

where f(x) is a quadratic functional and E(x) is an error function. Here $\bar{x}$ denotes the exact solution of Eq. (2.1) and the superscript T denotes transpose. Note that the negative of the gradient of the functional f(x) is nothing but the residual **r**, and is given by

$$-\nabla f(x) \doteq b - Ax \equiv r. \tag{2.3}$$

Any iterative method in which each successive approximation x_{k+1} is computed such that the increment $x_{k+1} - x_k$ is a linear combination of the gradient vectors (the residuals) taken at previous approximations is called a gradient method [7,8].

The objective of preconditioning is to consider, in place of the system Eq. (2.1), the modified system

$$Q^{-1}Ax = Q^{-1}b, \tag{2.4}$$

where $\mathbf{Q^{-1}A}$ has a better condition number than the original matrix **A**. The closer **Q** is to approximating **A**, the more clustered the eigenvalues of $\mathbf{Q^{-1}A}$ are around unity, which leads to improved convergence of gradient methods. Thus the main idea of preconditioning is first to obtain a matrix **Q** which is relatively easy to invert and is a good approximation to the coefficient matrix of the system under consideration, and second to apply this as a splitting matrix or preconditioner to construct faster converging conjugate gradient methods. The conjugate gradient algorithm for solving the linear systems (2.1) and (2.4) where **A** and **Q** are SPD matrices, is as follows:

__Algorithm 1__:

(a) Select initial guess: $\mathbf{x}_o$,
Construct residual: $\mathbf{r}_o = \mathbf{b} - \mathbf{A}\mathbf{x}_o$
Select a preconditioner: **Q**

(b) Iterate: k = 0, 1, 2, ..., until convergence
Construct pseudoresidual $\boldsymbol{\delta}_k$, A-conjugate direction vectors $\mathbf{p}_k$, and update the solution and the corresponding residual as follows:

$$\boldsymbol{\delta}_k = \mathbf{Q}^{-1}\mathbf{r}_k,$$

$$\beta_0 = 0, \qquad \beta_k = (\boldsymbol{\delta}_k, \mathbf{r}_k)/(\boldsymbol{\delta}_{k-1}, \mathbf{r}_{k-1}), \quad k \geq 1,$$

$$\mathbf{p}_0 = \boldsymbol{\delta}_0, \qquad \mathbf{p}_k = \boldsymbol{\delta}_k + \beta_k \mathbf{p}_{k-1}, \quad k \geq 1,$$

$$\alpha_k = (\boldsymbol{\delta}_k, \mathbf{r}_k)/(\mathbf{p}_k, \mathbf{A}\mathbf{p}_k),$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k,$$

$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A}\mathbf{p}_k.$$

The iterative steps are stopped when a convergence test such as

$$|\mathbf{r}_{k+1}|_2 \,/\, |\mathbf{b}|_2 \leq \text{EPS}, \qquad \textbf{(2.5)}$$

is satisfied.

In Algorithm 1, the expression (u, v) stands for the vector inner product (u, v) = $u^T v$. A similar algorithm, omitted for brevity, holds for the conjugate residual method.

The various steps in the conjugate gradient and residual algorithms involve (i) matrix-vector multiplications, (ii) solution of the pseudoresidual equation based on the preconditioning matrix Q and the residual right-hand side, and (iii) inner products and linear combinations of vectors. Item (iii) is straightforward and can be highly vectorized (parallelized based on strip-mining techniques). The other items will be discussed below.

2a. __Error Measures__

The simplest and most widely used error measure and stopping criterion is the ratio of the spectral norm (2-norm) in Eq. (2.5) which can also be written as

$$\text{RES} = \text{RES}(n) \equiv \|\mathbf{r}_n\|_2/\|\mathbf{b}\|_2 \leq \text{EPS}. \tag{2.6}$$

Note that the preferred initial guess for the solution in Algorithm 1 is $\mathbf{x}_0 = \mathbf{0}$ and the corresponding residual is given by $\mathbf{r}_0 = \mathbf{Ax}_0 - \mathbf{b} = \mathbf{b}$, the right-hand side of Eq. (2.1) under consideration. The iteration counters n for RES in (2.6) and for other error measures may be omitted for brevity. This measure can be rendered more conservative as indicated below.

For basic iterative methods (the conjugate gradient method being an acceleration technique of such a basic method [1,7])

$$\mathbf{x}_{n+1} = \mathbf{Gx}_n + \mathbf{k}, \tag{2.7}$$

a distinction is made between the difference between two successive solutions

$$\boldsymbol{\delta}_n = \mathbf{x}_{n+1} - \mathbf{x}_n, \tag{2.8}$$

and the exact error at step n

$$\boldsymbol{\epsilon}_n = \mathbf{x} - \mathbf{x}_n, \tag{2.9}$$

where $\mathbf{x}$ is the exact solution. Note from (2.7) - (2.9) that subtracting Eq. (2.7) from the corresponding expression $\mathbf{x} = \mathbf{Gx} + \mathbf{k}$ for the exact solution yields

$$\boldsymbol{\epsilon}_{n+1} = \mathbf{G}\boldsymbol{\epsilon}_n. \tag{2.10}$$

It follows that

$$\begin{aligned} \boldsymbol{\delta}_n &= x_{n+1} - x_n = x_{n+1} - x + x - x_n \\ &= -\epsilon_{n+1} + \epsilon_n = -G\epsilon_n + \epsilon_n = (I-G)\epsilon_n. \end{aligned} \tag{2.11}$$

Thus the exact error at step n is given in terms of the corresponding difference between successive solutions by

$$\boldsymbol{\epsilon}_n = (\mathbf{I}-\mathbf{G})^{-1}\boldsymbol{\delta}_n. \tag{2.12}$$

If $\rho(\mathbf{G})$ is the spectral radius for the iteration matrix **G**, then

$$\|(I - G)^{-1}\| \approx \frac{1}{1 - \rho(G)}, \tag{2.13}$$

where $\|\cdot\|$ is a suitable norm. A good estimate of the norm of the exact error is then

$$\|\epsilon_n\| \approx \frac{\|\delta_n\|}{1 - \rho(G)} \approx \frac{\|\delta_n\|}{1 - \text{Error Reduction}} \approx \frac{\|\delta_n\|}{1 - \|\delta_n\|/\|\delta_{n-1}\|}. \tag{2.14}$$

Equation (2.14) can be used as a basis to establish realistic error measures. The first attempt is to divide the measure in Eq. (2.6) by "one minus the error reduction." Unfortunately, the observed behavior of error reduction estimates is not monotonically decreasing at every step of a conjugate gradient or residual method. Thus local error reduction measures are not very meaningful. A reasonable approach is to estimate the error reduction based on the last k iterations. A conservative approach is to estimate the error reduction based on all the iterations. It is easy to see that this approach is conservative based on the observation that the convergence behavior of gradient methods, while fluctuating, in general improves as the iterations proceed.

Note for example that RES in Eq. (2.6) represents the net change in the spectral norm of the residual achieved in n iterations. The average per iteration

$$\text{Average Change per Iteration} = (\text{RES})^{1/n}, \tag{2.15}$$

can be used as an estimate of the error reduction. Thus a new extrapolated measure of the error based on Eq. (2.6) is

$$\text{ERES} \equiv \text{RES}/(1 - \text{RES}^{1/n}) = (\|r_n\|_2/\|b\|_2)/[1 - (\|r_n\|_2/\|b\|_2)^{1/n}]. \tag{2.16}$$

The authors derived several other error measures in place of the residual in Eq. (2.6) based on the minimization properties in Eq. (2.2). Extrapolated forms of the various measures, as in Eq. (2.16) were also derived. Extensive testing indicated that the behavior of the new error measures is very similar to that for the residual and its extrapolated form in Eqs. (2.6) and (2.16). Furthermore, among these measures, the ERES extrapolated residual error measure of Eq. (2.16) is conservative and appears to be the most appropriate for practical applications. Thus other error measures are omitted for brevity.

2.b. Stopping Criteria for Structural Analysis Accuracy

The residual and extrapolated residuals in the previous subsection are by no means true measures for the exact errors. The system of linear equations (2.1) of primary interest here is derived from finite element modeling of structural problems. The unknowns in the vector x represent displacements at various nodes. Subsequent structural analysis requires the evaluation of stresses which are linear combinations of derivatives of displacement vectors. Thus we are not only interested in measuring the error of the solution but also the error in the derivative of the solution to ensure accuracy for subsequent structural analysis applications.

The approach adopted here is to select a convenient error measure for displacements, such as the residual measure RES, and implement the following algorithm:

(i) Iterate until RES $\leq 10^{-3}$.
Construct derivatives of the solution for displacements to evaluate corresponding predictions of stress components.

(ii) Continue iterations until RES is reduced by an additional order of magnitude (factor of 10).

(iii) Re-evaluate stress components.
Update maximum stress vector.
Evaluate maximum change in stress over all stress vectors which are larger than 10^{-4} times maximum stress vector.

(iv) Check convergence. Return to step (ii) if:

(a) The maximum change in stress is larger than a user specified tolerance, and
(b) The total number of iterations is less than a prespecified value, and
(c) The remaining computing time is sufficient.

(v) Save the latest solution and additional information to allow restart and continuation.

The nice feature about iterative methods is the fact that a user can specify a given solution accuracy, obtain intermediate results, and if desirable, change the accuracy requirement and, continue by restarting the iterative process from where it stopped.

3. Assembled Matrix Storage and Matrix-Vector Multiplications

The system of equations corresponding to the structural problems of interest are large, sparse, symmetric, and do not have any regular pattern. Several possible common data structures for storing the corresponding coefficient matrices are summarized in [6,8] and are used in a variety of available sparse system solution packages. Of these, (a) the symmetric form of compressed sparse row/column storage, and (b) a modified form of the "Jagged Diagonal" storage format [6] were adopted and tested. In the compressed sparse row format, the non-zero elements in the upper part of the coefficient matrix **A** are ordered by rows and stored consecutively in a single array and their corresponding column locations are also stored in a single array

$$A(k), \quad JA(k), \quad k = 1, \ldots, NELT, \tag{3.1}$$

where NELT is the total number of non-zero elements in the upper triangular part (including the diagonal) of A. An array

$$IA(i), \quad i = 1, \ldots, N, \quad IA(N) = NELT, \tag{3.2}$$

where N is the order of the matrix (number of degrees of freedom) is selected as a pointer to the location of the ith diagonal elements of **A**.

An example of the compressed sparse row storage format is shown for a model 20 by 20 SPD matrix in Figure 1.

The Jagged Diagonal storage format is described in detail by Gregoire [6]. The main idea is to perform a permutation

$$\hat{A} = PAP^T, \tag{3.3}$$

where P is an orthogonal permutation matrix chosen so that the rows of $\hat{A}$ are ordered by a decreasing number of non-zero elements per row. The resulting Jagged Diagonal storage of **A** then makes it possible to achieve a very high degree of vectorization of matrix-vector multiplications. The vector length for the matrix-vector multiplications is initially N and decreases monotonically.

For the structural problems of interest to us, each row or column of a coefficient matrix can typically have up to 243 non-zero elements, and each row in the top triangular part of the coefficient matrix **A** typically has an average of between 50 and 100 non-zero elements. As a consequence, matrix-vector multiplication based on the compressed sparse row storage is predominantly vectorizable with a variable vector length. Since the average vector length is typically over 50, this is viewed as a decent vector length for supercomputers such as the Cray Y-MP and C-90.

For problems which fit in the central memory of a supercomputer, a variant of the Jagged Diagonal storage which takes advantage of matrix symmetry is the preferred choice [6,8]. For the large size of structural problems of interest to us (300,000 to several million unknowns), numerical tests carried out indicated that the cost of the permutation in (3.3) needed to construct the Jagged Diagonals is too high and is not covered by the slight gain in the speed of the matrix-vector multiplications. As a consequence, the compressed sparse row storage was adopted for our initial numerical studies.

One other matrix storage strategy was later tested. We introduced and successfully applied a "Block Jagged Diagonal" storage algorithm which takes advantage of matrix symmetry to cut in half the storage and data transfer requirements. In place of a single permutation, as in Eq. (3.3), it is desirable to carry out separate permutations

$$\begin{aligned} A_{l+1} &= P_l A_l P_l^T \\ &= P_l P_{l-1}, \ldots, P_0 A P_0^T, \ldots, P_{l-1}^T P_l^T, \end{aligned} \tag{3.4}$$

$$\underline{\underline{A}} = \begin{array}{r|cccccccccccccccccccc|}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\
\hline
1 & 8 & -1 & & & & & & & & -1 & & & & & & & & & & -1 \\
2 & -1 & 8 & -1 & & & -1 & & & & -1 & & & -1 & & & & & & & \\
3 & & -1 & 8 & -1 & & & & & & -1 & & & & & & & & & & \\
4 & & & -1 & 8 & -1 & & & & & & & & & & & & & & & \\
5 & & & & -1 & 8 & -1 & & & & & & & & & & & & & & \\
6 & & -1 & & & -1 & 8 & -1 & & & & & & & & & & & & & \\
7 & & & & & & -1 & 8 & -1 & & & & & & & & & & & & \\
8 & & & & & & & -1 & 8 & -1 & & & & & & & & & & & \\
9 & & & & & & & & -1 & 8 & -1 & & & & & & & & & & \\
10 & -1 & -1 & -1 & & & & & & -1 & 8 & -1 & & & & & & & & & \\
11 & & & & & & & & & & -1 & 8 & -1 & & & & & & & & \\
12 & & & & & & & & & & & -1 & 8 & -1 & & & & & & & \\
13 & & -1 & & & & & & & & & & -1 & 8 & -1 & & & & & & \\
14 & & & & & & & & & & & & & -1 & 8 & -1 & & & & & \\
15 & & & & & & & & & & & & & & -1 & 8 & -1 & & & & \\
16 & & & & & & & & & & & & & & & -1 & 8 & -1 & & & \\
17 & & & & & & & & & & & & & & & & -1 & 8 & -1 & & \\
18 & & & & & & & & & & & & & & & & & -1 & 8 & -1 & \\
19 & & & & & & & & & & & & & & & & & & -1 & 8 & -1 \\
20 & -1 & & & & & & & & & & & & & & & & & & -1 & 8 \\
\hline
\end{array}$$

Coefficient Array

A(k)= [8,-1,-1,-1,8,-1,-1,-1,-1,8,-1,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8,-1,8]

Column Array

J(k)= [1,2,10,20,2,3,6,10,13,3,4,10,4,5,5,6,6,7,7,8,8,9,9,10,10,11,11,12,12,13,13,14,14,15,15,16,16,17,17,18,18,19,19,20,20]

Row Ending Index Array

I(k)= [4,9,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,45]

Figure 1. Sample 20 by 20 Sparse Symmetric Matrix and the Three Arrays Used for its Condensed Row Storage

where each permutation I only reorders the matrix elements corresponding to a preselected set (a window or a partition of diagonal elements of appropriate size) of unknowns. With this approach, the need to cycle through the matrix numerous times to construct the required Jagged Diagonals (which can be very costly for very large problems) is eliminated. Here only segmental Jagged Diagonals are constructed in such a way as to achieve reasonable vector length appropriate for the computer being utilized. Such an approach would also be appropriate for parallel processing where each processor deals with a distinct segment of the matrix under consideration:

4. Choice of Preconditioners

The convergence properties and computational efficiency of the conjugate gradient and residual methods depend critically on the choice of a matrix preconditioner **Q**. The matrix **Q** should have the same order as **A** but must be easier to invert than **A**. It is preferable to select **Q** to be a good approximation to **A**. The following choices or preconditioned conjugate gradient methods were tested:

- **Q** = **I**, where **I** is the identity matrix,
- **Q** = **D**, where **D** is the diagonal part of **A**,
- **Q** = the tridiagonal part of **A**,
- **Q** = a block diagonal part of **A** (For initial testing, only diagonal blocks of a preselected size were permitted),
- **Q** = $(\mathbf{U}^T + \mathbf{d})\,\mathbf{d}^{-1}\,(\mathbf{d} + \mathbf{U})$, where $\mathbf{A} = \mathbf{U}^T + \mathbf{U} + \mathbf{D}$, **D** is a diagonal matrix and U is a strictly upper diagonal matrix, and d is selected such that **D** = Diagonal $(\mathbf{d} + \mathbf{U}^T\mathbf{d}^{-1}\mathbf{U})$,
- **Q** = $(\mathbf{U}^T + \mathbf{D})\,\mathbf{D}^{-1}\,(\mathbf{D} + \mathbf{U})$, the SSOR symmetric successive overrelaxation iteration matrix with the overrelaxation parameter $\omega = 1$. Here again, $\mathbf{A} = \mathbf{U}^T + \mathbf{U} + \mathbf{D}$, where **D** is a diagonal matrix and **U** is a strictly upper diagonal matrix,
- **Q** based on incomplete Cholesky factorizations with generation dependent, level dependent or magnitude dependent zero fill-in (typically referred to as ICCG(i), etc.)

Additional comments on some of the methods outlined above and considered in the present study are now in order.

(i) The implementation of the CG/D and CR/D methods with preconditioner Q = D, the diagonal matrix equal to the diagonal part of **A**, is simplified by multiplying the system (2.1) by $\mathbf{D}^{-1/2}$ and changing variables to transform it as follows

$$D^{-1/2} A D^{-1/2} D^{1/2} x = D^{-1/2} b, \tag{4.1}$$

$$\hat{A}\hat{x} = k, \quad \hat{A} = D^{-1/2} A D^{-1/2}, \quad \hat{x} = D^{1/2} x, \quad k = D^{-1/2} b. \tag{4.2}$$

Note that each diagonal element of $\hat{\mathbf{A}}$ is unity and the corresponding preconditioner for the CG/D or CR/D method applied to the system $\hat{\mathbf{A}}\hat{\mathbf{x}} = \mathbf{k}$ of Eq. (4.2) is the identity matrix **Q** = **I**. As a consequence, operations by the preconditioners as in Algorithm 1 are eliminated.

(ii) Various forms of the incomplete Cholesky factorization are obtained as follows. The Cholesky factorization of the SPD matrix **A** of (2.1) is of the form

$$A = U^T D U, \tag{4.3}$$

where **D** is a diagonal matrix, **U** is a unit upper triangular matrix, and $\mathbf{U}^T$ is the transpose of **U** and is a lower triangular matrix. The elements of **D** and **U** are given in terms of the elements of **A** by the expressions

$$d_i (= D_{ii}) = a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2 d_k, \qquad i = 1, ..., N, \tag{4.4}$$

$$u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj} d_k)/d_i, \qquad j = i+1, ..., N. \tag{4.5}$$

Direct solution methods are often based on algorithms suitable for the factorization in Eqs. (4.3)-(4.5). For sparse matrices, typically the matrix **U** is much more dense than the upper triangular part of **A** (numerous non-zero elements of **U** correspond to zero elements of **A** — referred to as zero fill-in) and storage may become a problem for very large systems. Incomplete factorization is based on the assumption, valid for certain idealized situations, that the dominant part of **U** is that part which corresponds to the non-zero part of **A**. Thus reasonable approximations to **A** can be obtained from (4.3) - (4.5) by, for example, imposing a sparsity pattern on **U** and not computing or storing elements beyond a desired sparsity structure. The sparsity structure selected for **U** can be identical to that of the upper part of **A** or can be a little larger in order to improve the approximation. The sparsity structure may also be based on the magnitudes of elements of **U**, thus certain elements in (4.5) smaller in magnitude than a prescribed value would not be stored nor used in subsequent computations. The use of incomplete Cholesky factorization with varying degrees of zero fill-in to construct preconditioners for iterative methods was demonstrated to be very effective in accelerating convergence and significantly reducing computational cost.

(iii) The special case of incomplete Cholesky factorization with no zero fill-in is of considerable interest. Here, the expressions in Eqs. (4.4) and (4.5) for elements of **U** corresponding to the zero elements of **A** are ignored altogether and not computed or stored. The preconditioner

$$\mathbf{Q} = \mathbf{U}^T\mathbf{D}\mathbf{U} \approx \mathbf{A}, \tag{4.6}$$

in this case can be computed based on Eqs. (4.4) and (4.5) as described above. Since **U** is restricted to have a structure identical to the non-zero structure of the upper triangular part of **A**, only an array

$$U(k), \quad k = 1, ..., NELT, \tag{4.7}$$

is needed for its compressed sparse row storage. The arrays JA(k), IA(i) in Eqs. (3.1) and (3.2) used to identify row and column locations of elements A(k) of **A** can also be used together with Eq. (4.7) to completely describe **U**.

The construction of **U** can be achieved one row at a time as follows: (a) non-zero elements a_{ij}, $j \geq i$ in Eqs. (4.4) and (4.5) for a fixed i are mapped into a work array; (b) the contributions from previous rows of **U** are subtracted from the work array in accordance with Eqs. (4.4) and (4.5); and (c) the work array elements of **U** corresponding to non-zero a_{ij} elements of **A** are extracted.

A problem with the special case with no fill-in is the fact that some diagonal elements of the matrix **D** in the incomplete factorization may turn out to be negative, which in general would cause the preconditioned conjugate gradient method with the preconditioner $\mathbf{Q} = \mathbf{U}^T\mathbf{D}\mathbf{U}$ to become very slow or to diverge. Several approaches are reported in the open literature to remedy the situation, including replacing each diagonal element by its absolute value(see for example [1,7,8]). An approach to introduce a diagonal shift discussed in [9] was rediscovered by the present authors and by others, was implemented in the present work and was found to very effective. The idea is to increase each diagonal element of the initial matrix, for the purpose of the incomplete factorization only, by between 1% and 7%. Increasing the diagonal elements increases the numerical stability of the incomplete factorization algorithm and the chance to obtain a SPD preconditioner. A modification of the approach to dynamically increase diagonal elements within windows of the matrix and to do so only when negative diagonal elements are encountered was also developed and implemented. Here, if a diagonal element of the factorization is found to be zero or negative, all rows of the incomplete factor **U** contributing to it are recomputed with the diagonal parts of **A** multiplied by γ where $\gamma = 1.02, 1.04, \ldots$, or is an input value to be increased incrementally to ensure a positive definite factorization. However, increasing the diagonal elements leads to a preconditioner which is a poorer approximation to the initial matrix.

The new modified incomplete Cholesky factorization with no zero fill-in involves replacing a_{ii} in Eq. (4.4) for d_i by the product $\gamma_m a_{ii}$ to get

$$d_i \left(= D_{ii}\right) = \gamma_m \, a_{ii} - \sum_{k=1}^{i-1} u_{ki}^2 \, d_k, \quad i = 1, \ldots, N. \tag{4.8}$$

The parameters γ_m are required to be greater than or equal to one ($\gamma_m \geq 1$), and be monotone increasing as a function of m ($\gamma_{m+1} \geq \gamma_m$) and of i. The first value γ_1 is selected in a narrow range around 1, say $1 \leq \gamma_1 \leq 1.05$. The values γ_m are to be determined dynamically during the incomplete factorization procedure. The idea is to take advantage of the fact that increasing the diagonal elements of the original SPD matrix (increasing diagonal dominance) improves the condition of the matrix [decreases the ratio of the largest eigenvalue divided by the smallest eigenvalue] and adds to the numerical stability of Cholesky factorizations. However, increasing the diagonal elements leads to a preconditioner which is a poorer approximation to the initial matrix.

The preferred algorithm for modifying the incomplete Cholesky factorization with no zero fill-in is as follows:

<u>**Algorithm 2**</u>:

(a) Set m = 1 and select γ_1 and an increment $\Delta\gamma$ for modifying the γ_m parameter. The default value preferred here is $\gamma_1 = 1$, unless the analyst suspects in advance that the standard incomplete factorization is inappropriate. The choice $\Delta\gamma = 0.05$ is a reasonable choice, but larger or smaller values may be selected. The incomplete Cholesky factorization is started in accordance with any suitable standard procedure based on Eqs. (4.4) and (4.5) for u_{ij} and d_i given above.

(b) Once a diagonal element d_i is computed, its sign is checked. If d_i is found to be positive, the incomplete Cholesky factorization procedure continues. If d_i is found to be zero or negative, then the following actions are taken: (i) the value of m is increased by one and γ_m is set to $\gamma_m = \gamma_{m-1} + \Delta\gamma$, (ii) the first row ℓ (the row with the smallest row index ℓ) which makes a direct non-zero contribution to d_i as given in Eq. (4.4) is identified, (iii) the modified incomplete Cholesky factorization is restarted at row ℓ (i is reset equal to ℓ) using the last modified parameter γ_m.

(c) One should guard against unreasonable cost of the factorization but check to ensure that m remains smaller or equal to some preselected upper limit M (say M = 10 for illustration). If in step (b), a value of m larger than M is detected, the modified incomplete Cholesky factorization should be restarted with a choice of a larger initial value γ, or should be abandoned and the user should be advised to apply the preconditioned conjugate gradient method with diagonal scaling.

The above algorithm was implemented and tested for small size systems and was found to be very effective. In fact, the standard incomplete Cholesky factorization with no zero fill-in failed to produce an SPD preconditioner for several structural problems and the modified algorithm proved to be very successful. For simplicity, the initial implementation in a stand-alone iterative solver research computer program designed for large problems which require auxiliary storage was restricted to M = m = 1, a single shift as in [9].

5. Initial Numerical Results for Problems which fit in computer central memory

The initial implementation of the conjugate gradient and residual methods was done for problems which fit in the central memory of the Cray Y-MP and Cray C-90 supercomputers and was based on storage of the upper triangular part of the matrix **A** by the compressed sparse row storage scheme presented in Section 3 above. The stored part of **A** includes the diagonal part **D** and the strictly upper triangular part **U** of **A** = **UT** + **D** + **U** required for the preconditioners

$$Q = (U^T + d)\, d^{-1}\, (d + U), \tag{5.1}$$

$$Q = \frac{\omega}{2-\omega}\, (U^T + \frac{1}{\omega}D)\, D^{-1}\, (\frac{1}{\omega}D + U). \tag{5.2}$$

The diagonal matrix **d** and its inverse $\mathbf{d}^{-1}$ can be constructed by a simple recurrence relation by requiring the diagonal elements of the preconditioner Q of Eq. (5.1) to coincide with the corresponding diagonal elements of **A**. Note that **Q** in (5.2) is the standard SSOR preconditioner.

In addition to the storage of **A** and of $\mathbf{D}^{-1}$ or $\mathbf{d}^{-1}$, the first set of preconditioned conjugate gradient and residual methods considered requires storage of between 5 and 7 vectors, each of length N, needed in order to carry out the various steps in Algorithm 1 and in an analogous algorithm for the conjugate residual method.

Nine structural problems were considered in this study. Representative samples are given in Figure 2 and Table I presented at the end of this paper. Extensive numerical testing, results omitted for brevity, indicate the following:

(i) Consistent with experience at Bettis and elsewhere [1,7,8], the diagonally preconditioned conjugate gradient and residual methods CG/D and CR/D are superior to the standard conjugate gradient and residual methods.

(ii) The diagonally scaled conjugate residual CR/D method is slightly, but not significantly, better than the diagonally scaled conjugate gradient CG/D method. These methods are much better than direct method for a large class of problems and are the methods of choice for extremely large systems. This is because they require the least computer storage and data transfer and are robust methods. The CG/D method is preferred in this work to the CR/D method because of difficulties we have had in adapting SSOR preconditioners to the conjugate residual method.

(iii) The conjugate gradient method with a preconditioner based on a block diagonal rather than strictly diagonal part of **A** was found not to be competitive for structural problems of interest and is not treated further here. The same was observed for the conjugate gradient method with a preconditioner based on the tridiagonal part of **A**.

(iv) The conjugate gradient method with SSOR preconditioning is competitive with the diagonally scaled method for problems which fit in computer central memory, especially if diagonal scaling is also applied to eliminate computations involving the diagonal elements. For problems which require auxiliary storage, the SSOR preconditioning requires three times as much data transfer as the diagonally scaled CG/D case, which would prove costly for the present generation of supercomputers and consequently reduce the effectiveness of this method.

(v) Several alternative formulations of the incomplete Cholesky conjugate gradient method with various degrees of fill-in, ICCG(i), i=1,2,..., were tested for problems which fit in the central memory of the Cray Y-MP and C-90 computers. Indeed the choice of preconditioners based on incomplete Cholesky factorization is superior to the choice of diagonal scaling or preconditioners based on SSOR relaxation methods. Furthermore, these results support the assertion that the conjugate gradient method with incomplete Cholesky preconditioners is in general competitive with direct solution methods for small and medium size problems and is superior for very large structural problems. The conjugate gradient method with incomplete Cholesky preconditioners requires more computer storage than other methods considered, but much less storage than direct methods. Specifically the incomplete Cholesky with zero fill-in preconditioned conjugate gradient method requires approximately 50% more computer storage than the diagonally scaled CG/D conjugate gradient method.

(vi) The initial application to a generic container head structural problem of both the ICCG(0) methods with no zero fill-in, and the equivalent CG/IF conjugate gradient method with shifted Cholesky incomplete factorization but with zero shift, failed. The suitability of the ICCG(1) method with one generation fill-in to successfully run this problem in computer central memory was demonstrated. This problem required over 48 million words of central memory on the Cray Y-MP. The ICCG(1) method converged in 69 iterations, required 54.7 seconds to solve, and confirmed the effectiveness of the method. The direct method for this problem consumed 526 seconds.

(vii) The ICCG(i) option exhibits the same behavior for structural problems as observed for the diffusion problems. Typically saturation occurs early for very small i = 1 or 2. While the number of iterations decreases as i increases, the cost per iteration increases and significantly reduces any overall benefit. The reduced system conjugate gradient methods with preconditioners based on incomplete odd-even cyclic reduction were demonstrated in [1] to be very effective for diffusion problems. In addition, typically saturation takes hold as the number of incomplete odd-even cyclic reduction steps increases (the degree of zero fill-in increases). Fortunately however, we have recently demonstrated that the application of row sum correction (requiring the preconditioner to produce the solution in one iterative step whenever the exact solution is flat, a constant) is very effective for diffusion type problems as it improves convergence rate and delays the appearance of saturation. On the other hand unfortunately, our initial experience indicates that row-sum correction does not appear to work for structural problems, possibly due to the fact that the property of diagonal dominance is typically lost.

(viii) Application of a shift is very effective for producing reasonably good incomplete Cholesky preconditioners for the conjugate gradient method. The CG/IF method with a shift is of special interest as it allows reasonable storage economy and much improved computational efficiency. In particular the link list technique implemented in Fortran in our CG/IF programs and in Cray Assembly Language (CAL) in the Cray SITRSOL PCG/IC(0) solver option [8] is very effective for incomplete factorization.

(ix) Matrix storage based Jagged Diagonals with or without utilizing matrix symmetry leads to high vector length and is to be preferred for problems which fit in computer central memory. Our initial tests indicate that constructing the Jagged Diagonals can be very costly for very large problems where auxiliary storage is necessary.

The initial implementation of the PCG/IF method is not optimal but is sufficient to demonstrate relative merits of this method for solving structural problems as compared to other methods.

6. Implementation of the CG/D Method Utilizing Auxiliary Storage

The conjugate gradient method with diagonal scaling CG/D has the least storage requirements among the methods considered. Consequently it is the method of choice for excessively large problems where computer storage and data transfer costs are expected to be limiting. The CG/D method was selected for initial implementation (a) in a stand alone program for solving linear systems for structural problems and (b) as an integral part of a production quality structural program. For the stand alone version, a separate program assembled the matrix for a structural problem based on compressed sparse row storage format and stored it together with the corresponding right-hand side in two or more files as needed to accommodate the problem size. The stand alone program reads the files for the matrix and right-hand side, solves the system and writes the results to a file which can be read by structural programs for further finite-element analysis computations.

The stand alone program for the CG/D method was constructed based on the Bettis FTB file management system routines and is designed to store the coefficient matrices in central memory, on the solid state storage device (SSD) of the Cray Y-MP and C-90 computers, or on up to six disks. For small size problems FTB files can also be restricted to reside in central memory.

The overhead due to FTB file storage and manipulation is around 15% for problems small enough to fit in central memory, 23% to 33% for problems which fit in the SSD solid state storage device and can be very excessive (factors of 3 to over 10 increase) for problems where the coefficient matrix is stored on disks.

To improve data transfer, the FTB-based CG/D program was designed to allow for storage of each coefficient matrix in several files, which could reside on user specified combinations of SSD and distinct disks. The main idea, for example, of utilizing six disks, in place of say two disks, is that the distinct files can be opened at the same time and thus in principle cut the net computing overhead time for data transfer by a factor of 6. This factor of 6 is not achieved in practice. This is understandable in a non-dedicated batch system since distinct jobs compete for the same disk and SSD resources. Nonetheless, it is always advisable to split files and store them on distinct devices in order to optimize computer utilization.

7. Element-by-Element (EBE) Implementation of the CG/D Method

The most effective implementation of the conjugate gradient method with diagonal scaling, both in terms of storage requirements and calculational speed, appears to be an implementation where the matrix-vector multiplication is performed element-by-element without explicitly calculating or storing the stiffness matrix [3,5]. The implementation possesses the following advantages:

(i) The stiffness matrix does not have to be calculated and assembled. The important point here is that it does not have to be assembled. For a large structure where the matrix will not fit in central memory, assembling the matrix is a difficult task because random access into the sparse matrix data structure is necessary to perform the assembly process. The structural problems of interest would typically have between a few millions and several hundreds of millions of non-zero elements (N = 50,000 to 6 million, NELT $\approx$ 60N to 120N).

(ii) The stiffness matrix does not have to be stored. It is only necessary to store a much smaller amount of data from which the required calculations can be performed. This provides a storage advantage and also contributes to increased efficiency because of reduced data transfer between central memory and external storage devices.

(iii) The required calculations can be highly vectorized and performed at near optimum speed on the Cray computer. This advantage is partially offset by the need to do more calculations than that required for an assembled matrix scheme. However, since the calculations can be performed faster, the net result is an increase in speed.

(iv) The method is amenable to parallel processing because the individual element contributions can be computed independently; each processor can handle its share of the elements.

(v) Strains and stresses are computed as a by-product of the procedure and are available for use in evaluating stopping criteria as described in Section 2b.

A preferred approach to performing the element-by-element matrix-vector multiplication is based on recognizing that even the element stiffness matrix is actually not needed explicitly; rather, only the vector resulting from the matrix-vector multiplication is the required end product and this result vector can be computed without explicitly forming the element stiffness matrix. The procedure is outlined briefly below.

The element stiffness matrix, **K**, is defined by the following volume integral:

$$K = \int B^T * C * B \, dV. \tag{7.1}$$

B is the spatially varying matrix which transforms the nodal displacement vector, **u**, into the strain vector, **e=B*u**. **C** is the (possibly spatially varying) material stiffness matrix which transforms the strain vector into the stress vector, **s=C*e**. The matrix-vector product, **K*u**, can be computed by the following steps:

$$(1)\ e = B*u, \quad (2)\ s = C*e, \quad (3)\ f = B^T*s, \quad (4)\ K*u = \int f dV. \qquad \textbf{(7.2)}$$

The integral in step 4 is, of course, computed numerically in the usual way. This defines the essence of the method implemented in our program.

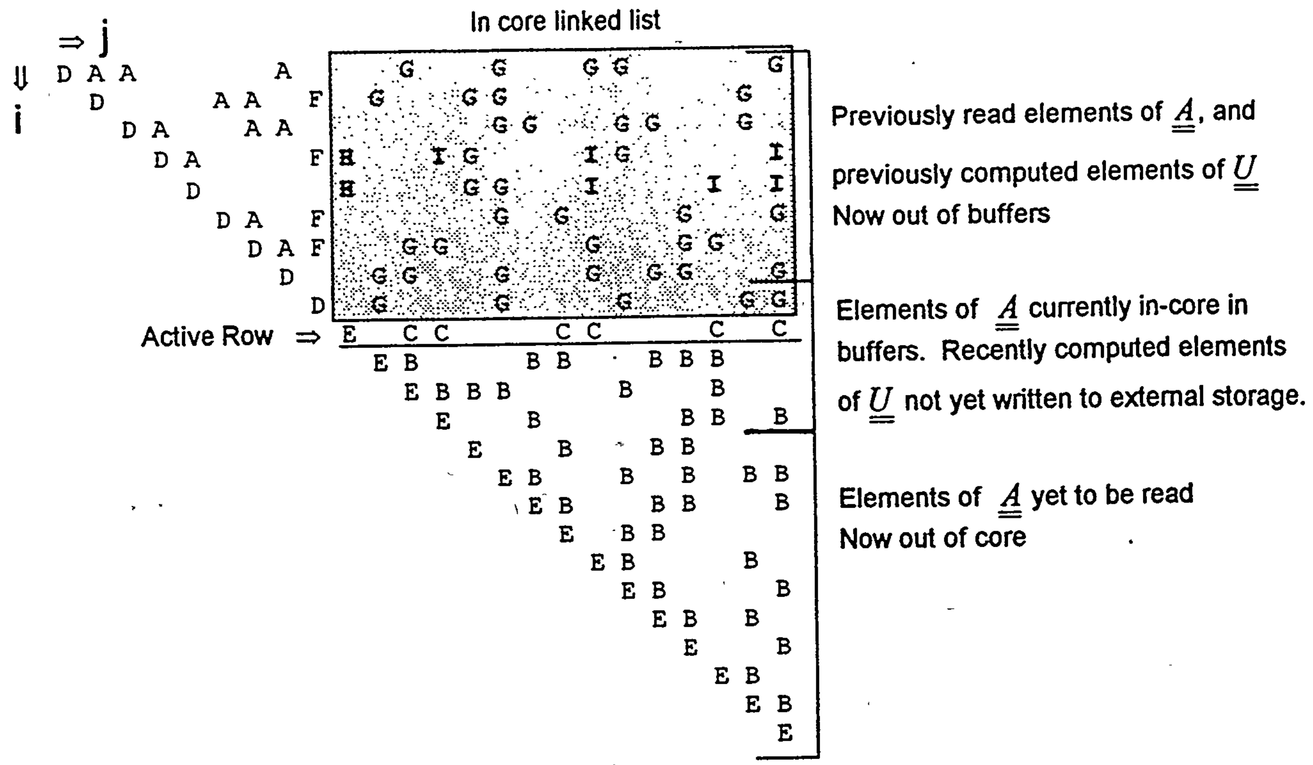
8. Implementation of the CG/IF Method Utilizing Auxiliary Storage

The initial implementation of the shifted Cholesky factorization in the stand-alone iterative solver research computer program which utilizes auxiliary storage was completed in the fall of 1992. The goal at the time was to select algorithms that are simple to program quickly. No attempt was made to vectorize and optimize the construction of the preconditioner.

The Bettis FTB system subroutines are used in the stand-alone computer program to handle data transfer between the central memory of a Cray computer and associated auxiliary SSD solid state and disk storage devices. For efficient data transfer, the storage required for the arrays A(k), and JA(k) is divided into a small number of equal size sets; each set corresponds to non-zero elements in a subset of consecutive rows in the upper triangular part of **A**. Again, central memory, SSD and up to six distinct disks are utilized for efficient data storage.

J. J. Haan developed a modified link list technique suitable for constructing the above shifted Cholesky incomplete factorization matrix preconditioner and implemented it in the stand-alone research computer program which utilizes auxiliary storage. The algorithm implemented enables the construction and storage of the array in Eqs. (4.5)-(4.8) while reading and storing the arrays for the Matrix **A** in Eqs. (3.1),(3.2). The algorithm is illustrated in Figure 3. A summary follows.

A new link-list procedure was developed to overcome potential difficulties with incomplete Cholesky factorization of very large systems. The main idea is to create temporary active storage in the central memory of the computer being used (or on a fast access device), to copy, store and be able to efficiently retrieve so-called active elements of **U**. Precisely, once an element u_{ij} in row i and column j (j>i) is constructed based on the above equations for the incomplete Cholesky factorizations, this element is copied into the temporary active storage space, and indices identifying it are defined. When required for subsequent computation, the element u_{ij} is retrieved not from the separate external storage allocated to **U**, but from the temporary active storage space. Once the element $d_j = D_{jj}$ of the diagonal matrix **D** is computed, all elements u_{ij} for i<j are purged from the temporary active storage space since such elements are not needed for subsequent computations. This purging process of elements no longer needed for the factorization frees space for subsequent use and provides for efficient utilization of computer storage. We recall for clarity that $u_{jj} = 1$ for all j and these diagonal elements need not be computed or stored.


⇒ j
⇓ i
In core linked list
Active Row ⇒
Previously read elements of $\underline{\underline{A}}$, and previously computed elements of $\underline{\underline{U}}$ Now out of buffers
Elements of $\underline{\underline{A}}$ currently in-core in buffers. Recently computed elements of $\underline{\underline{U}}$ not yet written to external storage.
Elements of $\underline{\underline{A}}$ yet to be read Now out of core

A	Off diagonal elements of the factored matrix previously computed and no longer required for subsequent factorization.
B	Off diagonal elements of the unfactored matrix in buffers or yet to be read.
C	Off diagonal elements of the active row, to be added to the linked list storage following factorization of the current active row.
D	Factored diagonal elements. The inverses of these elements are stored in central memory in a separate vector.
E	Diagonal Elements of the un-factored matrix. These elements are available in central memory in a separate vector.
F	Elements released from the linked list storage before factorization of the current active row.

Linked List:

G&H&I	Off diagonal elements previously read and required to factor rows yet to be read. These elements are stored in central memory in a linked list.
H	Non-zero elements above the diagonal element which identify rows that contribute to the factorization of the active row.
I	Off diagonal elements needed to factor the current active row.

Figure 3. Incomplete Cholesky Factorization: Large System Storage Scheme

A reasonable approach for accomplishing the incomplete Cholesky factorization is to construct the elements of **U** one row at a time starting from the first row and proceeding in ascending order to the last row. One arrangement followed here is to link together elements in the active storage space which lie along the same column of **U** (u_{ij} with fixed j and varying i). Since the entries of the elements of the matrix **U** into active storage occur as each element of **U** is calculated, necessarily in this case, the row ordering in the link-list is monotonic. As a consequence, the summations in the above expressions for u_{ij} and d_i can be efficiently calculated by traversing the active element storage list. Elements of non-contributing rows are also ignored in this process, reducing the need to check that **U** element products will have a non-zero contribution to subsequent elements.

Different means exist for creating the needed temporary active storage and implementing the above procedure. The approach implemented here creates an index vector to indicate whether or not a column j of the matrix **U** has active elements in temporary active storage locations. For example, a value of the jth element of the index vector of zero would indicate that there are no active elements stored for row j. A value of the jth element of the index vector different from zero would imply that one or more active elements u_{ij} exist and would indicate (point to) the location of the last (or first) such elements in temporary storage (last element here means the element u_{ij} with the largest row number i). The temporary storage associated with an element u_{ij} also provides the value of the row index i and the location of the next successive element in column j, if such an element exists.

9. Numerical Experiments with the CG/IF Method

The stand-alone research computer program which utilizes auxiliary storage was applied to solve the representative set of five structural problems listed in Table I. This set is a subset of the nine problems we considered. A sample of the corresponding structures is shown in Figure 2.

The linear system corresponding to each of the problems listed in Table I was solved on the Cray Y-MP and the Cray C-90 using both the CG/D and the CG/IF options. Timing results are listed in Tables II and III. Residual error norms $\| r_n \|_2 / \| b \|_2$ are shown in figure 4 both as a function of iteration and as a function of net CPU (Central Processing Unit) compute time on the Cray C-90. Tables II and III and Figure 4 and other results omitted indicate the following:

(i) The cost of the shifted Cholesky incomplete factorization as currently implemented is relatively high. This is due to the fact that the authors did not vectorize the factorization algorithm during the initial implementation. Proper vectorization should cut the cost of the incomplete factorization by a factor of two to ten or more.

(ii) The cost per iteration step for the PCG/IF method is a factor of between 2 to 3 larger than the corresponding cost for the CG/D method. The number of iterations needed for convergence for the PCG/IF method is reduced by a factor of eight or more for the largest problems as compared to the CG/D method. The net results summarized in the last two columns of Table III indicate that the PCG/IF method is a factor of between 1.6 and 4.4 more efficient in terms of iteration time, and a factor of between 1.3 and 3.7 more efficient in terms of total computing cost than the CG/D method.

(iii) The most significant result is the fact that the PCG/IF method is much more efficient than the CG/D method for difficult problems such as shell problems. Table II shows that the CG/D method as implemented in the stand-alone research program is approximately a factor of 2 less efficient for Problem 9 than the direct method. Table III shows that the PCG/IF method is a factor of 3.4 to 4.4 faster for Problem 9 than the CG/D method, and about a factor of two faster than the direct method for this difficult problem.

10. Block Jagged Diagonal Data Structure

The block jagged diagonal matrix structure is a generalization of the jagged diagonal structure introduced by J. P. Gregoire [6,8] to speed up matrix-vector multiplications. The original scheme [6,8] constructs jagged diagonals for an entire sparse matrix and is very effective for matrix-vector multiplications only for systems small enough to fit in the central memory of the computer being used. Construction of jagged diagonals for an entire sparse matrix is very costly to implement for systems large enough to require auxiliary storage.

Jagged diagonals can be constructed in a straightforward manner for blocks of a very large matrix. Thus, a preferred approach for carrying out matrix-vector multiplications in situations where the matrix is too large to fit in a central memory of the computer being used, involves modifying the jagged diagonal algorithm of Gregoire and Heroux and others [6,8] as follows. In order to take advantage of symmetry, the upper part of the coefficient matrix **A** is subdivided into sets of rows. The subdivisions are such that the non-zero elements in each set should easily fit in the central memory of the computer being utilized. The non-zero elements in each set are reordered into distinct non-overlapping subsets where no two elements of a subset lie along the same row or column of the matrix. The maximum number of subsets of a given set is given by the maximum number of non-zero elements in any row of the matrix **A** that lies in that set. Pointers to the locations of non-zero elements of **A** are also kept. For a given subset, since the elements lie on distinct rows and distinct columns of the matrix **A**, construction of the contributions of these elements to matrix-vector multiplication are vectorizable with variable vector length less than or equal to the number of elements in the subset. Clearly, the larger the set size, the larger the sizes of the subsets and the larger the average vector length to be expected. The subsets may also be ordered with a monotonically decreasing or increasing number of elements in each of them. It is to be understood that the ordering described above for a set of rows of the upper triangular part of **A**, can equally well be applied to the columns of the lower triangular part of **A**. When **A** is symmetric, only the lower part or upper part needs to be considered. The above ordering scheme can also be applied to matrices which are not symmetric.

The construction and application of block jagged diagonal structure was not actually implemented for the CG/D and G/IF methods for large problems which require auxiliary storage. Its effect however was simulated for matrix-vector multiplications for a large set of contrived matrices. The matrices were chosen to represent varying degrees of sparsity with patterns that are typical for structural systems. The timings indicate that implementation of the block jagged diagonal scheme could cut the cost of the CG/D solver by a factor of two and cut the cost of the CG/IF solver by over 25 percent.

11. Summary and Conclusions

Suitability of preconditioned conjugate gradient and residual methods with alternative choices of preconditioner for solving very large structural problems was demonstrated. For excessively large problems where computer storage and data transfer requirements are limiting, the CG/D diagonally scaled conjugate gradient and residual methods are the methods of choice. The diagonally scaled conjugate gradient method was implemented in a stand-alone program which utilizes the Bettis FTB system routines to manage several distinct disk and solid state storage devices. The CG/D method was also implemented based on an element-by-element scheme for accomplishing matrix-vector multiplications.

The conjugate gradient method with preconditioners based on incomplete Cholesky factorization with zero fill-in (i.e., with a sparsity pattern identical to that of the linear system to be solved) and with a possible shift of origin to ensure numerical stability was shown to be suitable for moderate and large structural problems.

The work described above demonstrated that the PCG/IF preconditioned conjugate.gradient method with preconditioners based on shifted Cholesky incomplete factorization with no zero fill-in is a very competitive method for solving large linear systems which arise in structural applications. This method is especially appropriate for difficult problems such as shell problems for which the CG/D methods require excessively large numbers of iterative steps for convergence.

The current implementation of the PCG/IF method in the stand-alone research program is not optimal. In particular, the scheme for accomplishing the incomplete factorization has not been vectorized. Vectorization should cut the factorization cost by a factor of two to ten or more. The matrix-vector multiplications as implemented for the PCG/IF and CG/D methods are also not optimal. For solutions on vector machines, a preferred approach for carrying out matrix-vector multiplications in situations where the matrix is too large to fit in central memory of the computer being used, involves modifying the jagged diagonal algorithm of Gregoire and Heroux and others [6, 9] as described in Section 10.

The stand-alone research program as described above is useful for application to the solution of very large linear systems where the system is already assembled.

D. N. Hutula convincingly demonstrated that an element-by-element implementation of the CG/D method (a) can reduce computer storage requirements by a factor of 20 as compared to the initial implementation in the stand-alone program, and (b) can improve vectorization of the CG/D algorithm resulting in a factor of four reduction in computing cost. The present work demonstrated that the PCG/IF method is superior to the CG/D method for numerous problems when the full coefficient matrix is given. An element-by-element analog of the PCG/IF method can prove to be very valuable, especially for problems where the CG/D method is costly and can be expected to require numerous iterative steps for solution convergence.

A block jagged diagonal matrix representation was introduced as a generalization to the jagged diagonal structure proposed by J. P. Gregoire [6,8] to speed up sparse matrix-vector multiplications. The resulting speed increase for large problems which require auxiliary storage is roughly two fold when compared to our current condensed sparse matrix-vector conjugate gradient structural solver on the Cray Y-MP and C-90 computers.

Acknowledgements

We are grateful to Dr. M. Natelson, Manager of Reactor Technology, for his insight, and for initiating, motivating, and supporting this work. He understood the effectiveness of iterative methods for the solution of diffusion problems and realized the potential of these methods for applications to structural problems. He challenged us to investigate and implement these methods and provided us with guidance, help, and support.

It is a pleasure to also thank W. R. Cadwell, J.D. Dankosky, D.J. Falcione, L. A. Hageman, C. Jeffrey, R. B. Smith, and E. A. Zanoni for numerous useful discussions and for their support and encouragement during various stages of this work.

REFERENCES

1. I. K. Abu-Shumays, "Incomplete Block Cyclic Reduction as a Preconditioner for Polynomial Iterative Methods," SIAM J. Sci. Stat. Comput., 11, 545-562, May 1990.

2. S. F. Ashby, "Polynomial Preconditioning for Conjugate Gradient Methods," Ph.D. Thesis, Department of Computer Science Report No. UIUCDCS-R-87-1355 (DOE/ER/25026-8) University of Illinois at Urbana-Champaign, 1987.

3. G. F. Carey, E. Barragy, R. McLay, and M. Sharma, "Element by Element Vector and Parallel Computations," Communications in Applied Num. Meth., 4, 299-307, 1988.

4. P. Concus, G. H. Golub, and G. Meurant, "Block Preconditioning for the Conjugate Gradient Method," SIAM J. Sci. Stat. Comput., 6, 220-252, 1985.

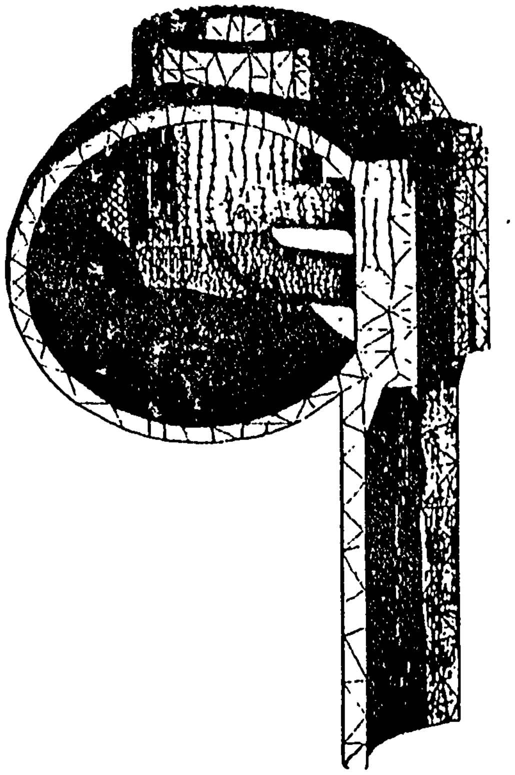
5. J. Erhel, A. Traynard, and M. Vidrascu, "An Element-by-Element Preconditioned Conjugate Gradient Method Implemented on a Vector Computer, Practical Aspects and Experiences," Parallel Computing, 17, 1051-1065, 1991.

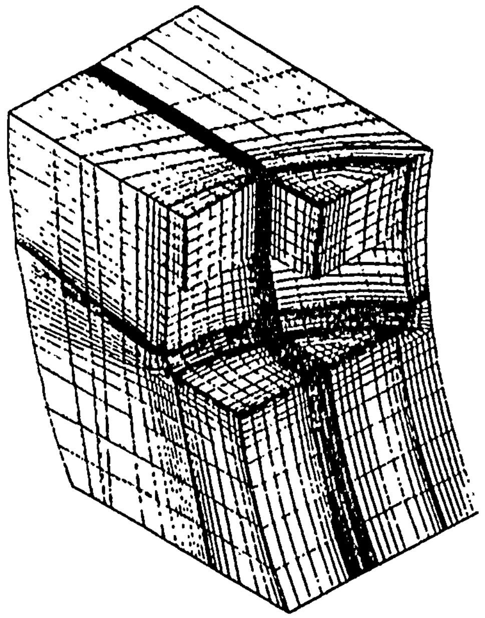
6. J. P. Gregoire, "Efficient Vectorization of the Conjugate Gradient Method," in Science and Engineering on Supercomputers, Computational Mechanics Publications, Southampton Boston, Proceedings of the Fifth International Conference held in London, England, October 22-24, 353-359, 1990.

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

8. M. A. Heroux, P. Vu, C. W. Yang, "A Parallel Preconditioned Conjugate Gradient Package for Solving Sparse Linear Systems on a Cray Y-MP," Applied Numerical Mathematics, 8, 1991. See also Solvers for Sparse Linear Systems, Volume 3: UNICOS Math and Scientific Library Reference Manual, Cray Research, Inc. Report SR-2081 6.0, 1991.

9. T. A. Manteuffel, "An Incomplete Factorization Technique for Positive Definite Linear Systems," Math. Comp., 34, 473-497, 1980.


(a) Coupled Shell Structure
Problem 2


(b) Finite Element Mesh
Problem 5

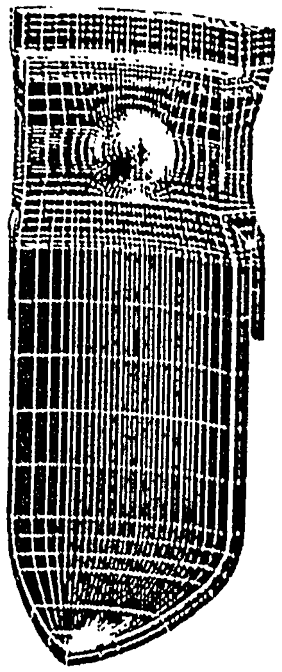

(c) Thick Shell Problem 6

Figure 2. Sample of the Structural Problems

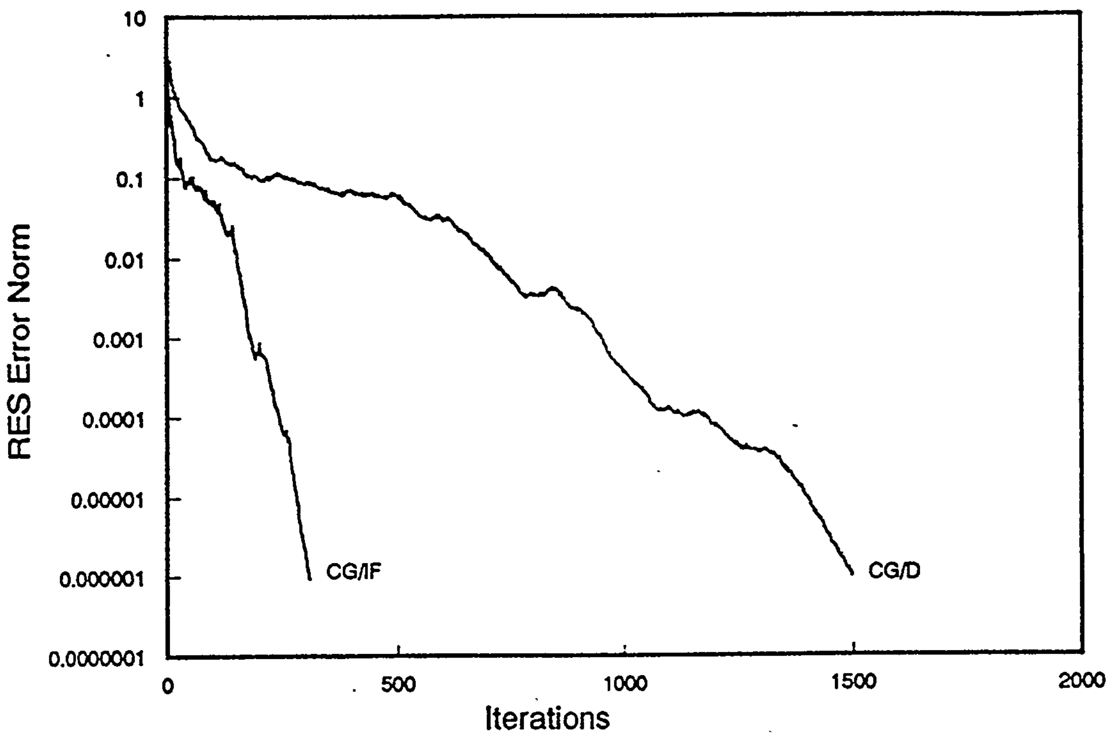


Figure 4a. Residual Error Norms for the CG/D and PCG/IF Methods for Problem 2 as a Function of Iterations

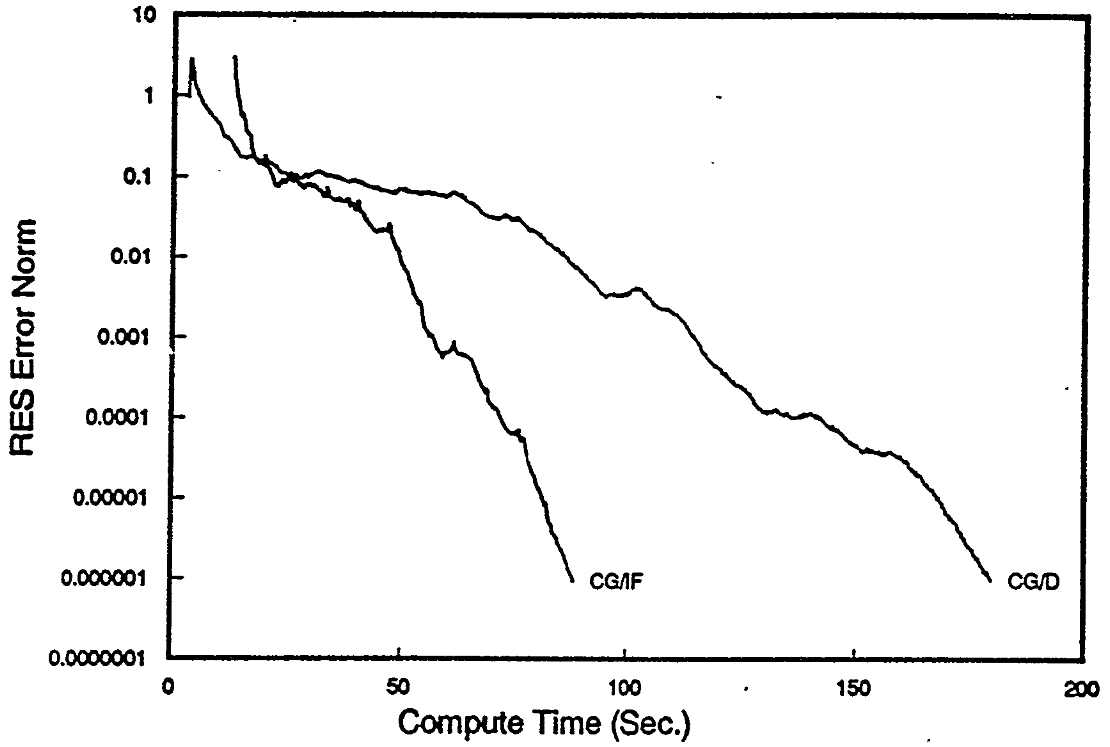


Figure 4b. Residual Error Norms for the CG/D and PCG/IF Methods for Problem 2 as a Function of Net Cray C-90 CPU Compute Time

TABLE I
Sample Structural Problems

Problem No.	Brief Description	N Degrees of Freedom	NELT Non-zero Elements Stored
1.	Generic Container Head	55,576	4,062,279
2.	Coupled Shell Structure	46,737	928,204
3.	Multiple-Connected Structure	360,791	26,958,917
4.	Section of Multiple-Connected Structure	396,087	29,498,463
5.	Solid Structure with FE Mesh	296,635	24,252,026
6.	Thick Shell	267,036	21,174,531

TABLE II

Cray Y-MP Run Time for the CG/D Method
with SSD for Auxiliary Matrix Storage and RES $\leq 10^{-6}$

Problem Number	N Degrees of Freedom	Iterations	Solution Time	
			Iterative	Direct
1	55,576	823	393 sec.	526 sec.
2	46,737	1,500	280 sec.	238 sec.
3	360,791	8,671	7.6 hrs.[a]	6.7 hrs.
4	**396,087**	**1,627**	**1.6 hrs.**	**22.5 hrs.**
5	**296,635**	**1,666**	**1.3 hrs.**	**5.7 hrs.**

[a] Result from an early version of program. About 20% increase in efficiency has been achieved since then.

TABLE III

Cray C-90 Comparison of Conjugate Gradient (CG) Iterative Solvers with Diagonal Scaling (CG/D) and with Shifted Incomplete Cholesky Factorization Preconditioners (PCG/IF)

PROBLEM	CG/D				PCG/IF					CG/D TO PCG/IF RATIO	
	ITER	I/O	SOLUTION	TOTAL	ITER	SHIFT	I/O+Q[a]	SOLUTION	TOTAL	SOLUTION	TOTAL
2	1497	3.3s	177s	180s	307	1.05	12.5s	75s	88s	2.36	2.05
3	8671	26s	3.1 hrs	3.1 hrs	920	1.005	807s	0.7 hrs	0.9 hrs	4.40	3.35
4 (EBE)[b]	1627	27s	37 min	37 min (9.7 min)	189	1.04	16 min	9.1 min	25 min	4.05	1.49
5 (EBE)[b]	1666	21s	30.2 min	31 min (6.6 min)	203	1.05	13 min	7.9 min	21 min	3.82	1.47
6	11000	19s	3.0 hrs	3.0 hrs	1164	1.03	11.0 min	40 min	51 min	4.41	3.46

[a] Data transfer and factorization times combined.
[b] Corresponding TOTAL computing cost for the EBE implementation, shown in ( ), is given here for comparison.