

Scalable Sparse Matrix Techniques for Modeling Crack Growth [★]

Padma Raghavan¹, Mark A. James², James C. Newman³, and B. R. Seshadri⁴

¹ The Pennsylvania State University, e-mail `raghavan@cse.psu.edu`.

² National Research Council, e-mail `m.a.james@larc.nasa.gov`.

³ Mississippi State University, e-mail `j.c.newman.jr@ae.msstate.edu`.

⁴ Old Dominion University, e-mail `b.r.Seshadri@larc.nasa.gov`.

1 Introduction

Elastic plastic fracture mechanics relies on the finite element method both as an engineering tool to evaluate residual strength of a damaged structure and to implement research theories that cannot be evaluated analytically. One common formulation assumes that the crack plane is flat and that the structure is symmetric about the crack plane. This situation arises for typical laboratory coupons such as the compact tension and middle crack tension specimens. Crack growth is modeled by monitoring analysis results for a critical fracture criterion, extending the crack, and resuming the analysis. When a symmetry plane corresponds to the crack-plane, an efficient crack growth procedure called *nodal release* can be used. A typical implementation reduces to the simultaneous solution of a set of linear equations, each of which corresponds to a displacement in the body being modeled. The system of equations is symmetric positive definite and sparse. This paper presents scalable sparse matrix factorization and update techniques required for efficient crack extension using nodal release.

At initialization of the nodal release process, all crack nodes on the crack-plane are constrained with symmetry conditions, except those nodes representing the crack. Of the three degrees of freedom at each node, only the crack-plane normal degree of freedom is constrained to zero leaving the other two unconstrained. If a solution variable of a system of equations $Ax = b$ is forced to zero, then that row and column of the system can be eliminated. Another strategy is to simply add a large number to the diagonal element of the stiffness matrix A , thus numerically deemphasizing the other terms of the equation and forcing the solution to zero. This approach is attractive because in certain circumstances the modification to the diagonal element can subsequently be removed from the system, thus releasing the constraint on the equation to grow the crack numerically. This process has been used quite effectively for crack growth for a number of years [12, 19].

In terms of the underlying sparse matrix computations, the nodal release method for crack extension requires (i) factoring the sparse symmetric positive definite matrix $A = LL^T$ (Cholesky) and solving $Ax = b$ at the initial condition, (ii) repeatedly updating the factor to model releasing constraints on certain equations to grow the crack numerically, and, (iii) between two successive updates to the factor, solving for a sequence of right hand side vectors to iterate to equilibrium as the load is incrementally increased. We present a framework in which these three major sparse matrix operations can be implemented efficiently. Furthermore, we show that our methods will allow latency tolerant scalable parallel implementations for modeling crack growth using multiprocessors or networks of workstations.

Section 2 describes the nodal release method for crack propagation and contains background material on sparse matrix factorization. Section 3 contains our new method for computing the sparse factorization once and efficiently updating the factor to model crack growth. This section also contains an analysis of the computational costs of the update algorithm, its memory requirements, and the communication costs and scalability of a multiprocessor implementation. Section 4 provides preliminary results from a serial implementation and Section 5 contains some concluding remarks.

[★] This work has been funded in part by the National Aeronautics and Space Agency through grant NAG-1-01085 and by the National Science Foundation through grants NSF CCR-981334 and NSF ACI-0102537 and through the Maria Goeppert Mayer Award from the Department of Energy and the Argonne National Labs. This work was performed by Dr. James when he was an NRC Research Associate at NASA Langley Research Center, Hampton, VA 23681.

2 Background

In this section we provide brief overview of the state-of-the art algorithms and implementations for sparse Cholesky factorization and the nodal release method for crack propagation.

2.1 Sparse Direct Solvers

Sparse direct solvers are based on a Cholesky factorization of the coefficient matrix when it is symmetric and positive definite. The first step in sparse Cholesky factorization concerns managing *fill-in*, i.e., zeroes in the original matrix that become nonzeros in the factors. A typical solution process has four steps (1) ordering to compute a fill-reducing numbering, (2) symbolic factorization to determine the nonzero structure of the factor, (3) numeric factorization, and, (4) triangular solution [5, 6]. As shown in the next section, the last two numeric steps are of key significance for our crack growth application. Recent research (see survey in [4]) has resulted in algorithms for a scalable, parallel implementation of sparse Cholesky factorization. We next describe the basic data structures and algorithms used for the numeric steps.

Henceforth, let L be the structure of the Cholesky factor of A after the application of a fill-reducing permutation. Modern numeric factorization schemes are based on exploiting *effectively dense* groups of columns within L . The columns of L can be grouped into *supernodes* where a supernode is a set of consecutive columns that have nested sparsity structure. The lower triangular matrix induced by the columns in a supernode is essentially dense in the subscripts of rows containing nonzeros in the lowest numbered column. The computation can then be organized as a set of dense matrix-matrix and matrix-vector operations implemented using cache-efficient kernels available through the Basic Linear Algebra Subroutines (BLAS)[3]. As a consequence of sparsity, columns in a supernode need not be updated by columns in all preceding supernodes. The data-dependence between supernodes is given by a *supernodal tree*; a supernode j can be updated only by columns in supernodes within the subtree rooted at j . The two leading schemes for sparse numeric factorization are a column-block approach and a multifrontal method [5, 14, 18]. The two schemes differ in the data-movement and in the amount of temporary storage during factorization.

Parallel sparse Cholesky with P processors utilizes both task and data parallelism. For ease of explanation, consider the following *compute tree* derived from the supernodal tree. Assume without loss of generality that the latter is complete binary for at least $\log_2 P$ levels from the root. At these levels, each compute tree node maps directly to a supernodal tree node. However, the last $\log_2 P$ leaf nodes in the former now represent subtrees rooted at corresponding supernodal tree vertices. These subtrees correspond to task parallel *local phase* computations. All associated columns are assigned to a processor and computations associated with factorization and triangular solution proceed independently. At an interior node j of the compute tree, all processors assigned to leaves of the subtree at j participate in *distributed phase* computations which are essentially data-parallel dense distributed matrix operations. Observe that all processors participate in computations associated with the dense matrix at the root.

2.2 Finite Element Analysis of Crack Growth

We now provide a brief description of a finite element analysis program incorporating crack growth with particular emphasis on equilibrium and crack growth steps. The derivation of the governing equations follow from [1, 11] using differential equations that describe the problem at equilibrium and use the principle of virtual displacements. These equations are then applied to model crack growth using the nodal release technique. The equations are valid for general equilibrium as long as (i) a statically admissible stress distribution is defined, and (ii) a kinematically admissible displacement solution is defined. The current work is limited to small strain, small rotation, but materially non-linear static equilibrium.

The finite element method discretizes the body of interest into regions each individually interpolated by linear or quadratic functions. First, isoparametric interpolation functions for the displacements are defined as: $u^{(m)} = N^{(m)}U$ where $N^{(m)}$ is the displacement interpolation matrix and U is the element nodal displacements for element m . The strains are interpolated using $\varepsilon^{(m)} = B^{(m)}U$ where $B^{(m)}$ is the strain-displacement matrix obtained by differentiating the displacement interpolation matrix $N^{(m)}$. The last equation can be written in a discretized form for a group of elements, where the contribution of each element is added to the global system of equations as follows:

$$\sum_{m=1}^n \left\{ \int_{V^{(m)}} \delta \varepsilon^{(m)T} \sigma^{(m)} dV^{(m)} \right\} = \sum_{m=1}^n \left\{ \int_{S^{(m)}} \delta u^{S^{(m)T}} f^{S^{(m)}} dS^{(m)} \right\}. \quad (1)$$

Upon substituting for the virtual strains and displacements,

$$\delta U^T \sum_m \left\{ \int_{V^{(m)}} B^{(m)T} \sigma^{(m)} dV^{(m)} \right\} = \delta U^T \sum_m \left\{ \int_{S^{(m)}} N^{(m)} f^{S^{(m)}} dS^{(m)} \right\}. \quad (2)$$

Now δU is the variation of the displacement vector for the total domain. Since δU is arbitrary, by dropping the superscript for the element, and considering only one element, $\int_V B^T \sigma dV = \int_S N f^S dS$. The right hand side of the last equation is zero except at equations where external loads are applied. For elastic-plastic applications the traditional linear Hooke's Law stress-strain relationship only holds in the linear region of the material response. In the nonlinear region the relationship is nonlinear and can be expressed as $\sigma = \int_0^\varepsilon \sigma(\varepsilon) d\varepsilon$. The von Mises yield model and an incremental stress-strain relationship is employed with a multi-linear yield surface hardening function [1]. The problems of interest here are commonly called quasi-static; that is, there are no inertial effects, but when nonlinear effects such as plasticity are considered, the solution to the problem may be loading path dependent, and so, dependent on a pseudo-time. Equations 1 and 2 are solved using the modified Newton-Raphson scheme, where only the initial stiffness matrix is used for iterations to equilibrium. This modified Newton-Raphson scheme provides the motivation for the algorithms described in this paper because it allows a more efficient implementation of an existing crack growth algorithm.

The application of primary interest is crack growth. Because crack growth involves creating new surface area over time, the boundary conditions are continually changing as the analysis progresses. Figure 1 shows a typical laboratory specimen, a middle crack tension, or $M(T)$, specimen. In this case three planes of symmetry ($x = 0, y = 0, z = 0$) are used and the $y = 0$ plane is coincident with the crack plane. Initially, all crack nodes on the crack-plane are constrained with symmetry conditions, except those nodes representing the crack. At each node there are three degrees of freedom, and only the crack-plane normal degree of freedom (the UY displacement) is constrained to zero. Figure 2 shows this process schematically in two dimensions. Stiff springs ahead of the crack tip provide the symmetry boundary condition required before extension. The stiff springs at the crack tip are released and replaced by an equivalent force, which can be released over several steps to simulate growth. When symmetry conditions such as these change, the traditional approach is to reform and re-factor the stiffness matrix with the new boundary conditions imposed. However, a significantly more efficient approach is to update the factored form of the stiffness matrix ($A = LL^T$) to reflect the changes in the boundary conditions.

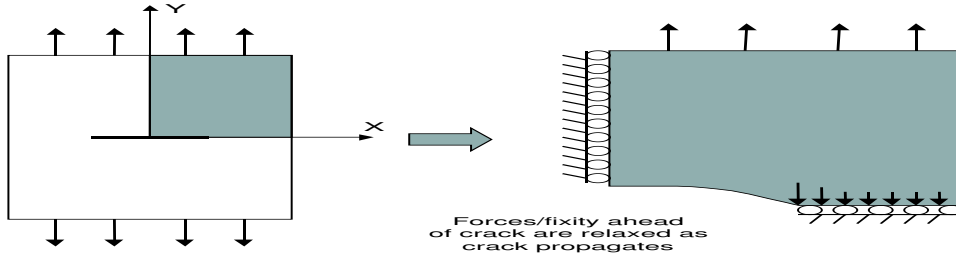


Fig. 1. Schematic showing lines of symmetry for a middle crack tension specimen. During elastic-plastic fracture, symmetry conditions are imposed along the vertical edge, and special stiff springs maintain symmetric along the fracture surface.

2.3 The Role of Cholesky Factors in Crack Simulation.

Figure 3 shows the main steps of the crack growth simulation code. In terms of the underlying sparse matrix computations, the application has the following form.

1. Factorization: Compute $A = LL^T$ for the original stiffness matrix and boundary conditions. This occurs in the initialization step after reading data from a file, generating internal representations for the boundary conditions, and building the stiffness matrix A .

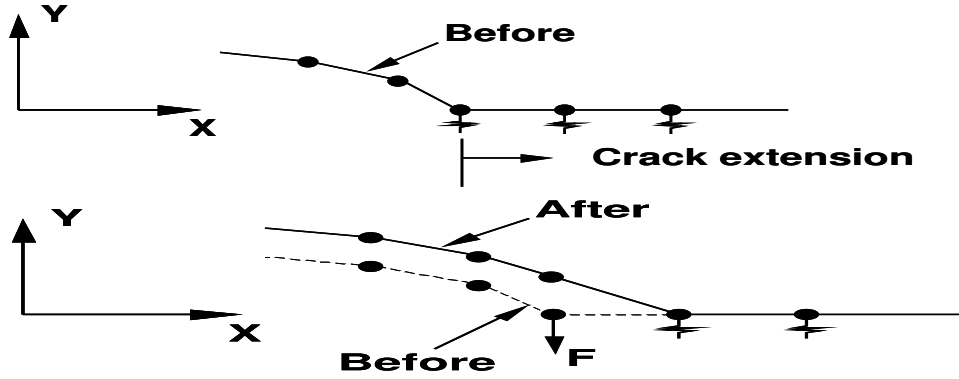


Fig. 2. The top figure shows the crack opening profile before crack extension; stiff springs ahead of the crack tip provide the symmetric boundary condition required. The second figure shows the crack opening profile after crack extension when stiff springs at the crack tip are released and replaced by an equivalent force, which can be released over several steps to simulate growth.

2. Triangular Solution: Use L to solve for a sequence of right-hand-side vectors as the load is incrementally increased while iterating to equilibrium.
3. Factor Update. Update L to reflect a change in the boundary conditions as the crack is grown.
4. Repeat steps 2 and 3 until the crack is resolved.

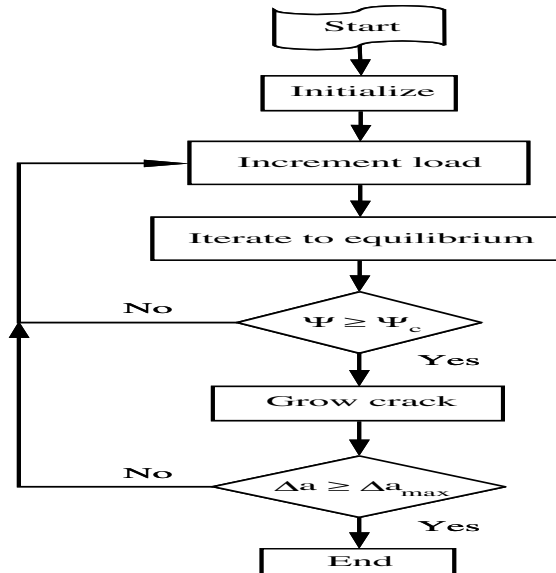


Fig. 3. Crack Growth Simulation Process; Ψ is a fracture parameter and Ψ_c is its critical value and Δa is the crack extension.

The crack growth step requires changing the stiffness matrix appropriately. In the application, a large number is added to the diagonal element of the stiffness matrix, thus numerically deemphasizing the other terms of the equation and forcing the solution to zero. To grow the crack numerically, the large diagonal element is subsequently removed from the system, thus releasing the constraint on the equation. Given a sparse symmetric positive matrix A , the factorization $A = LL^T$ can be first computed and then updated to model nodal release when the change to the matrix is a symmetric rank-1 modification. When the LL^T factors are *dense*, method of Gill et. al [7] can be used but adapting this dense update scheme for sparse factors is complicated by the fact that it involves both symbolic and numeric operations. One potential

approach is to use the sparse update scheme outlined in the paper by Davis and Hager [2]. The update process then involves traversing the elimination tree (used in the symbolic factorization step) and selectively applying a symbolic update step as dictated by column dependencies arising from the sparsity structure. This determines the changed zero-nonzero structure of L after which a numeric update can be applied.

The arithmetic cost per update of the sparse rank-1 method will be low but the implementation will suffer from cache-inefficiency because only level-1 BLAS can be used. Furthermore, the scope and potential for utilizing parallel processing is not significant. Another drawback is that the numerical update algorithm, can become unstable when defined in terms of LL^T [7]. More numerically stable versions presented in [7] are not directly applicable to the sparse case because they rely on certain orthogonal transformations. In the original implementation of the crack-modeling simulation using a skyline solver and the method of Gill et. al for updates, some sensitivity has been observed. The solution is sensitive to the value of the constraint term, requiring a term large enough to constrain the equation, but not so large as to truncated all of the significant digits of the constrained term. An implementation that does not attempt to restore the original stiffness matrix by subtracting out the constraint equation term is therefore more desirable.

3 Scalable Sparse Matrix Algorithms for Crack Propagation.

Our sparse computation framework takes into account three key application features. First, the unknowns (columns) of the linear system that are affected by crack growth \mathcal{U} is known in the initialization step. Second, modeling crack growth as a series of rank-1 updates to the original factor can become unstable. Third, several thousand triangular solutions are required during the simulation; the right-hand-side vectors are generated in successive iterations to equilibrium (as the load is incrementally increased).

Ordering. We first utilize the fact that the set of columns (\mathcal{U}) that can be updated during the simulation is known in advance. If N is the dimension of A , then the columns in the update column set are numbered $N - |\mathcal{U}| + \infty, \dots, N - \infty$, i.e., assigned the highest $|\mathcal{U}|$ numbers. Such an ordering places the columns in the root supernode of the supernodal tree. Such an ordering step will allow us to develop an efficient update process to the Cholesky factor L . Additionally, it can potentially speed-up the computation within the application. For example, the application level costs for crack-growth will be decreased because associated degrees of freedom get grouped together allowing data reuse and eliminating overheads from repeatedly traversing complex data-structures.

Computing and Updating Cholesky Factors. The crack growth simulation naturally produces several column modifications to A at the same time. These can be combined and applied in a single update step. With a careful choice of data-structures and factorization related information from lower numbered columns, the update process can be incorporated very efficiently within a *multifrontal* [5] scheme. Very few symbolic operations are needed and the multi-rank update can be applied using higher levels of BLAS [3] for improved efficiency. Our method is also inherently data-parallel.

Our update algorithm can be viewed as an extension to any efficient numeric factorization scheme such as the column methods of Ng and Peyton [14] or the multifrontal scheme of Duff and Reid [5]. We explain our method in terms of the latter because it is more closely related to the analysis in the next section. Multifrontal sparse Cholesky, treats sparse matrix factorization as a sequence of dense matrix operations which depend on the supernodal tree or compute tree described earlier. At a vertex j in the tree, numeric factorization involves factoring a leading block of columns of the associated dense matrix. The factored portion is used to accumulate contributions to columns in ancestor supernode. At vertex j

during factorization a dense triangular matrix $M_j = \begin{pmatrix} L_j & 0 \\ \tilde{L}_j & Q_j \end{pmatrix}$ is assembled and partially factored. The matrix M_j is assembled using elements in A corresponding to columns in L_j and contributions to L_j, \tilde{L}_j and Q_j from the children vertices of j . Only the first L_j columns of M_j are fully factored. The contributions from these columns to higher numbered columns is accumulated into Q_j . The latter is the portion of the matrix which will be assembled into the matrix M_i where i is the parent of j .

Factorization is applied using the standard multifrontal scheme such as the one in DSCPACK [15]. To allow future modifications to the computed L , we augment the method as follows. Let r be the root of the compute tree and let M_r be the associated matrix. Let $M_r = \begin{pmatrix} L_r & 0 \\ \tilde{L}_r & L_u \end{pmatrix}$ be partitioned such that columns in L_r are columns at the root supernode numbered lower than columns in \mathcal{U} (if any) while L_u corresponds to columns in \mathcal{U} . After the factorization step at r , let $L_u = Q_c + Q_r + A_u$ where Q_c is the dense

triangular matrix of dimension $|\mathcal{U}|$ containing assembled contributions from the children of r , Q_r contains contributions from the factor columns in L_r and A_u contains nonzeros in A corresponding to columns in \mathcal{U} . A matrix $R_u = Q_c + Q_r$ is stored and used to update L when the columns in \mathcal{U} are modified. Using suitable values from R_u and the modified nonzeros in A_u (\tilde{A}_u), a modified L_u (\tilde{L}_u) can be easily computed. This partial re-factorization will require no more arithmetic operations than to perform a dense Cholesky factorization of a matrix of dimension $|\mathcal{U}|$. Incidentally, the latter is no more expensive than applying the method of Gill et. al to the same dense matrix.

Triangular Solution. Recall that repeated triangular solutions are required to iterate to equilibrium; these can cause severe degradation in performance on parallel multiprocessors [8]. This problem of scalability stems from the fact that traditional distributed substitution schemes are communication latency bound. However, this problem of scalability has been recently addressed using the *selective inversion* (SI) method [16]. At each supernode dense matrix in the distributed phase, SI inverts the *dense diagonal* block which is then used to replace substitution by distributed matrix-vector multiplication. As shown in [16], the scheme leads to ideal scalability and efficiency at a slight overhead of computing the inversion. We propose to apply SI to all dense distributed triangular solves at supernodes except at the the root r . At r we will apply SI to the submatrix corresponding to L_r . For the portion corresponding to L_u (or \tilde{L}_u) we continue to use distributed substitution. The latter represents a small fraction of the computation and the communication overheads are potentially not substantial enough to warrant the costs of inversion after each modification step.

3.1 Analysis of Computational Costs and Scalability.

We now provide analytical results to show that our algorithms are scalable for a class of representative model sparse problems. Consider crack propagation when the stiffness matrix has the structure of the model sparse matrix of dimension $N = K^3$ associated with the three-dimensional $K \times K \times K$ five-point finite-difference grid. The crack growth is expanded along a neighborhood of at most c degrees of freedom (columns or equations) along one dimension. Thus $u = |\mathcal{U}| = \lfloor cK \rfloor$ where c is typically no more than 12 and is often smaller. To simplify the analysis, we assume that a regular nested dissection ordering [6] is used so that the compute-tree is a regular binary tree. Each compute tree supernode corresponds to a *separator* in the grid and we henceforth indicate a node by the size of the associated separator [6].

Understanding the scalability of the triangular solution step is relatively easy. Recall that we propose to use the selective inversion scheme; as shown in [16], the scheme leads to ideal scalability and efficiency at a slight overhead of computing the inversion. The inversion overhead is limited to under 6% of the cost of factorization for the $K \times K \times K$ grid. Additionally, the number of messages per triangular solution is greatly reduced [16, 17]. Traditional substitution requires $O(\frac{N}{P}^{\frac{2}{3}})$ messages while selective inversion requires $O(\{\log_2 P\}^2)$ per triangular solution [17]. Observe that the latter is independent of the matrix dimension and is a slow growing function of the number of processors. In our modified method, SI is used in all cases except the distributed triangular solution using the last $|\mathcal{U}|$ columns. This increases the number of messages by at most $\frac{cK}{P}$ leaving it still considerably smaller than for the traditional substitution method which requires $O(\frac{K^2}{P})$, $N = K^3$ messages. The rest of the section concerns the scalability of method to compute and update the factors.

The following two results are used in our analysis. First, consider the Cholesky factorization of an $m \times m$ dense matrix. The total number of floating point operations required, $D(m)$, is given by the sum $\sum_{i=1}^m i^2$ and is thus $\frac{m^3}{3} + O(m^2)$. Next, consider the arithmetic cost of factoring an $m \times m$ dense matrix using p processors. Using a block-cyclic mapping of the matrix to processors, the arithmetic cost at a processor is $O(\frac{m^3}{p})$ and the associated interprocessor communication overhead is no more than $O(\frac{m^2}{\sqrt{p}})$. The total communication overhead over all processors is $O(m^2 \sqrt{p})$. These results are derived in the book by Gupta et. al [10].

Lemma 1. *Consider the model $K \times K \times K$ problem ordered using regular nested dissection. The total arithmetic cost of factorization is no more than $\frac{1088357}{205065}K^6 + O(K^4)$ and thus $O(N^2)$ where $N = K^3$. The total number of nonzeros in the factor L is no more than $\frac{360}{30}K^4 + O(K^3)$.*

The results can be derived along the lines of analysis for the model $K \times K$, 2-dimensional problem [6]. Let $A(K, 0)$ be the number of operations for the $K \times K \times K$ grid that is not bordered by separators on any

side. This is equivalent to the arithmetic operations over all supernodes in the tree. Likewise, let $A(K, i)$ represent the number of operations for the grid bordered by separators on i sides, where i is in the range $1, \dots, 6$. The top (root level separator/supernode) of size $K \times K$ (S_1) gives two $K \times K \times \frac{K}{2}$ grids. These grids are bordered on one side and with two separators of size $K \times \frac{K}{2}$ (S_2) and can be partitioned into four grids bordered on two sides. These four grids can be separated into eight subgrids of size $\frac{K}{2}$ bordered on 3 sides by 4 separators of size $\frac{K}{2} \times \frac{K}{2}$ (S_3). Let $4A(S_3)$ represent the arithmetic work at the 4 S_3 separators, $2A(S_2)$ represent the work at the 2 S_2 separators and $A(S_1)$ at the S_1 separator. Now we get the recurrence relation:

$$A(K, 0) = 8A\left(\frac{K}{2}, 3\right) + A(S_1) + 2A(S_2) + 4A(S_3) \quad (3)$$

$$A(K, 0) = 8A\left(\frac{K}{2}, 3\right) + \frac{K^6}{3} + 2A(S_2) + 4A(S_3) \quad (4)$$

Recall that $A(S_1)$ is the dense factorization of a matrix of dimension K^2 and is thus $\frac{K^6}{3}$. The result is obtained by deriving expressions for $A(K, i)$ and simplifying. The number of nonzeros in the factor $NZ(K, 0)$ can be obtained using a recurrence relation of the form $NZ(K, 0) = 8NZ\left(\frac{K}{2}, 3\right) + NZ(S_1) + 2NZ(S_2) + 4NZ(S_3)$. Note that $NZ(S_1)$ is the storage for a dense triangular matrix of dimension K^2 and thus $\frac{K^4}{4} + O(K^2)$. \square

Lemma 2. *Consider the model $K \times K \times K$ problem ordered using regular nested dissection for the update algorithm with $|\mathcal{U}| = |\mathcal{K}|$. The total arithmetic cost of factorization increases by no more than $c^2 K^4 + O(K^3)$; thus the higher order cost of the factorization is unaffected and is no more than $\frac{1088357}{205065} K^6$. The incremental cost of update process at any crack growth step is no more than $\frac{c^3 K^3}{3}$ and hence $O(N)$. Additionally, the extra memory required to process incremental updates is proportional to $\frac{c^2 K^2}{2} + O(K)$.*

Using the argument in Lemma 1, the update process simply expands the top separator S_1 to include columns in \mathcal{U} ; let this expanded separator be \tilde{S}_1 . The total arithmetic cost of factorization is:

$$A(K, 0) \leq 8A\left(\frac{K}{2}, 3\right) + A(\tilde{S}_1) + 2A(S_2) + 4A(S_3).$$

The increase in the factorization cost is thus

$$A(\tilde{S}_1) - A(S_1) = D(K^2 + cK) - D(K^2).$$

Using the expression for D , the equation simplifies to $\frac{1}{3}\{K^6 + 3c^2 K^4 + (c^3 + 3c)K^3 - K^6\}$ and thus to $c^2 K^4 + O(K^3)$. This increase is not significant in the asymptotic sense and should be negligible for large K .

Consider updating the factor. extra storage is used to store updates from all previous columns to the last \mathcal{U} columns in S_1 , then the update can require no more than the cost to factor a dense matrix of size $u = |\mathcal{U}|$. This incremental cost of an update step is no more than $\frac{cK^3}{3}$ and hence $O(N)$. Incidentally, this cost is no more than the cost of applying the rank-1 update method of Gill et. al within the given ordering.

We next consider at the extra memory required to store updates from all previous columns to the \mathcal{U} columns in S_1 . This memory is proportional to the number of nonzeros in \tilde{S}_1 corresponding to the last cK columns. The latter form a dense triangular matrix and hence the memory required is proportional to $\frac{c^2 K^2}{2} + O(K)$ nonzero elements.

Parallel Performance. Consider a parallel multifrontal implementation. From the preceding discussion it is clear that parallel multifrontal factorization can be applied in the traditional sense with the only difference that the top separator (dense matrix) increases in size from K^2 to $K^2 + cK$. Since the increase is of lower order we can easily see that the parallel arithmetic cost and communication overhead will not increase in the order of magnitude sense. Thus the scalability of the extended factorization is no different than the scalability of traditional factorization. We next show that an incremental update to the factor is also scalable.

Lemma 3. *A single update step using P processors requires no more than $O\left(\frac{c^3 K^3}{P}\right)$ arithmetic operations. The communication overhead at a processor is $O\left(\frac{K^2}{\sqrt{P}}\right)$.*

We assume that updates from previous columns are stored locally on each processor corresponding to its share of the block-cyclic distribution of the \mathcal{U} portion of S_1 . The assembling would therefore require no extra communication. Now the update process is no more expensive than computing the Cholesky factorization of a matrix of dimension cK using P processors. The result follows from the costs of dense parallel factorization [10].

We have thus provided an extended factorization and incremental update scheme with costs that are of lower order compared to the cost of factorization. Furthermore, the speedup and scalability of the update process is the same as that of a dense matrix of dimension $N^{1/3}$ and thus insignificant when compared to the cost of factorization ($O(N^{2/3})$). The update cost is in fact less than the cost of a single triangular solution using the factors. The repeated triangular solution is also no longer latency bound because we can use selective inversion.

4 Empirical Results

In this section we present results of our preliminary implementation using the ZIP3D [19] crack simulation package and the sparse solver DSCPACK [15]. We report on observed improvements in performance of a serial implementation; we plan to work on a complete parallel simulation in the near future.

The timing results compare equivalent functionality to obtain consistent comparisons for performance before and after the modifications. Henceforth, we use ‘Old’ to denote the original implementation of the sparse matrix operations using a skyline solver and the method of Gill et. al for applying rank-1 updates. Likewise, we use ‘New’ to denote the new implementation using the algorithms presented in this paper; i.e., multifrontal factorization using the multiple minimum degree ordering, and the update step to modify the factors. The hardware used is an SGI Origin 3200 with 2, 400Mhz, R12000 processors and 2 gigabytes of RAM. Version 7.3.1.3m of the SGI MIPSpro FORTRAN90 and C compilers were used. The virtual size of the largest test problem did not exceed the physical RAM limit and the resident size of even the largest test problem only used only half of available memory. Average processor utilization observed is greater than 99%.

Test Suite. The test suite comprises three crack models with approximately an order of magnitude difference in sizes between the smallest and largest examples. This difference serves to highlight the effects of problem size on the performance of two schemes. We describe the middle tension MT24 model (see Figures 1 and 2) which uses a mesh with refinement along regions of crack growth. Applied uniform displacement boundary conditions along the top edge of the model provide loading with monotonically increasing displacements. The material thickness $B = 6.35\text{mm}$, and the specimen width is $W = 610\text{mm}$. The FEA mesh uses symmetry at $x = 0, z = 0$, and crack plane symmetry at $y = 0$ with an initial crack length of $2a/W = 1/3$. The analysis was set to produce a maximum of 45 mm of crack growth. Table 1 summarizes the properties of the 3 models.

Performance. Table 1 provides the sizes of L as a measure of the memory storage requirements for the old and new methods. It is not surprising that our new implementation using a multiple minimum degree ordering incurred substantially lower fill. Timing results for the three models are shown in Table 2. The time required to factor the system is significantly affected by the algorithm and implementation, and as expected, the new scheme is significantly faster for the larger problems. The new scheme is faster than the old scheme by factors of more than 10 for the larger problems and on average by a factor of 3.73. The performance gain results primarily from using general sparse factorization techniques and an efficient multifrontal implementation. Over all the update steps, the new scheme performs significantly better than the old scheme. The performance improvement is clearly from the superiority of our update algorithm and results in speedups of over 50 for MT24 and on average of 43.4. Finally, over all triangular solution steps our approach is faster than the old scheme on average by a factor of 3.6. It is clear that over all three steps, the new scheme results in a substantial performance improvement. The speedups for the factorization, update and solution steps are shown in Figure 4.

We would like to observe that for both CT2 and CT6, the number of triangular solutions is the exactly the same in both the old and new methods. However, for MT24, the new implementation required two extra solves for the same amount of crack growth (Table 1 shows the count for the old method). Each solve represents an iteration in the Newton scheme, and given the nonlinear nature of the problem, some round off difference is possible. For each model, the results of the old and new methods are identical to six significant digits.

Table 1. Description of crack modeling stiffness matrices (A); each node has 3 degrees of freedom (equations).

Model	Equations	Nonzeroes in A (10^6)	Number of Nonzeroes in L		Updates	Update Columns	Triangular Solutions
			New (10^6)	Old (10^6)			
CT2	14,256	.91	3.15	5.85	104	316 (2.2%)	7,568
MT24	22,986	1.38	4.05	18.4	258	753 (3.2%)	17,149
CT6	101,310	7.41	81.6	197.2	1,210	2,838 (2.8%)	49,270

Table 2. Execution times (in seconds) and speedups.

Model	Execution Time in Seconds											
	Factor Time				Update Time				Solve Time			
Model	Count	New	Old	Speedup	Count	New	Old	Speedup	Count	New	Old	Speedup
CT2	1	4.21	9.37	2.2	104	2.56	100.37	39.2	7,568	895.06	2,043.83	2.3
MT24	1	6.91	95.43	13.8	258	19.91	1,082.43	54.4	17,149	2,936.23	14,853.50	5.1
CT6	1	383.3	4,282.0	11.2	1,210	773.32	28,371.20	36.7	49,270	141,172.21	473,422.87	3.4
Mean				3.73				43.4				3.6

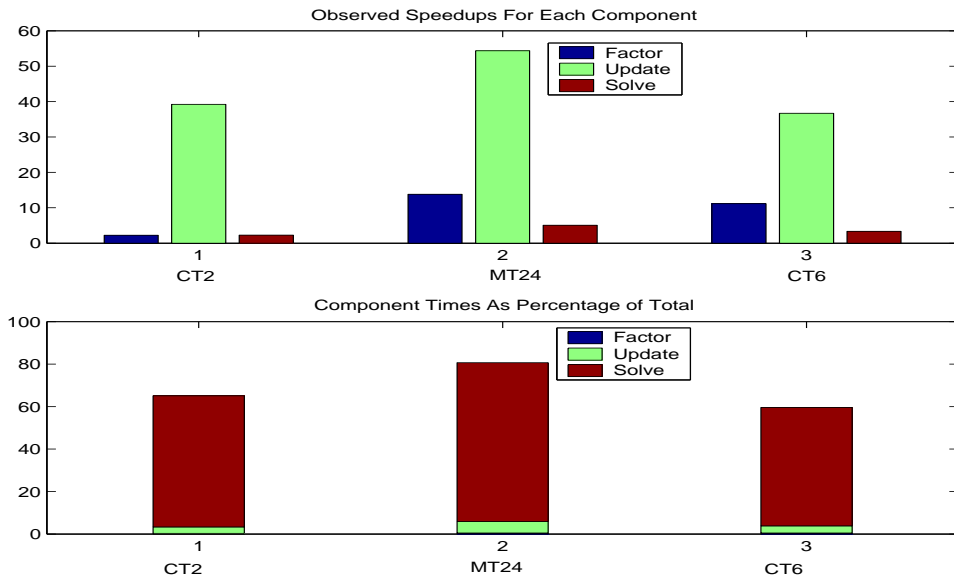


Fig. 4. Speedups of component steps, and component step times as a percentage of total simulation time for the new method.

5 Conclusions

We provide an effective computational scheme for modeling crack growth. Our initial results show that on average, our scheme speeds the Cholesky factor update step by a factor of 43.4. Our analytic results indicate that our method is both efficient and scalable.

References

1. K. J. Bathe, *Finite Element Procedures*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1996.
2. T. A. Davis and W. W. Hager, *Modifying a sparse Cholesky factorization*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 606–627.
3. J. J. Dongarra, J. Croz, S. Hammarling, and I. S. Duff, *An extended set of basic linear algebra subprograms*, ACM Trans. Math. Software, 14 (1988), pp. 1–17.
4. I. S. Duff, *Sparse Numerical Linear Algebra: Direct Methods and Preconditioning*, Technical Report TR-PA-96-22, CERFACS, 42 Avenue G Coriolis, 31057 Toulouse Cedex, France, 1996.
5. I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*, Oxford University Press, Oxford, England, 1987.
6. A. George and J. W-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.
7. P. E. Gill, G.H. Golub, W. Murray and M. A. Saunders, *Methods for Modifying Matrix Factorizations*, Mathematics of Computation, 28 (1974), pp. 505-535.
8. M. T. Heath, E. Ng, and B. W. Peyton, *Parallel algorithms for sparse linear systems*, SIAM Review, 33 (1991), pp. 420–460.
9. A. Gupta, G. Karypis, and V. Kumar, *Highly Scalable Parallel Algorithms for Sparse Matrix Factorization*, IEEE Transactions on Parallel and Distributed Systems Volume 8, Number 5 (1995).
10. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin Cummings, Reading, MA, 1994.
11. L. E. Malvern, *Introduction to the Mechanics of a Continuous Medium*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1969.
12. J. C. Newman Jr., *Finite-Element Analysis of Fatigue Crack Propagation-Including the Effects of Crack Closure*, Ph.D. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1974.
13. J. C. Newman Jr., C. A. Bigelow, and K. N. Shivakumar, *Three Dimensional Elastic-Plastic Finite Element Analyses of Constraint Variations in Cracked Bodies*, Engineering Fracture Mechanics, 46 (1993), pp. 1–13.
14. E. Ng and B. W. Peyton, *Block sparse Cholesky algorithms on advanced uniprocessor computers*, SIAM J. Sci. Comput., 14 (1993), pp. 1034–1056.
15. P. Raghavan. *DSPACK: Domain-Separator Codes for the parallel solution of sparse linear systems*, Technical Report CSE-02-004, Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802-6106, 2002.
16. P. Raghavan, *Efficient Parallel Triangular Solution with Selective Inversion*, Parallel Processing Letters, 8 (1998), pp. 29–40.
17. P. Raghavan, K. Teranishi, and E. Ng, *Towards Scalable Preconditioning Using Incomplete Factorization*, Under Review, Numerical Linear Algebra (2000).
18. E. Rothberg, *Performance of panel and block approaches to sparse Cholesky factorization on the iPSC/860 and Paragon multiprocessors*, Technical Report, Intel Supercomputer Systems Division, 14924 N. W. Greenbrier Parkway, Beaverton, OR 97006, (1993).
19. K. N. Shivakumar and J. C. Newman Jr., *ZIP3d- An Elastic and Elastic-Plastic Finite Element Analysis Program For Cracked Bodies*, NASA TM 102753, (1990), NASA Langley Research Center, Hampton, VA.