An efficient algorithm for simulating fracture using large fuse networks

2003 J. Phys. A: Math. Gen. 36 11403

(http://iopscience.iop.org/0305-4470/36/45/004)

INSTITUTE OF PHYSICS PUBLISHING JOURNAL OF PHYSICS A: MATHEMATICAL AND GENERAL

J. Phys. A: Math. Gen. **36** (2003) 11403–11412 PII: S0305-4470(03)67227-6

An efficient algorithm for simulating fracture using large fuse networks*

Phani Kumar V V Nukala and Srdan Simunovic

Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6359, USA

E-mail: nukalapk@ornl.gov

Received 6 August 2003
Published 29 October 2003
Online at stacks.iop.org/JPhysA/36/11403

Abstract

The high computational cost involved in modelling of the progressive fracture simulations using large discrete lattice networks stems from the requirement to solve *a new large set of linear equations* every time a new lattice bond is broken. To address this problem, we propose an algorithm that combines the multiple-rank sparse Cholesky downdating algorithm with the rank-p inverse updating algorithm based on the Sherman–Morrison–Woodbury formula for the simulation of progressive fracture in disordered quasi-brittle materials using discrete lattice networks. Using the present algorithm, the computational complexity of solving the new set of linear equations after breaking a bond reduces to the same order as that of a simple *backsolve* (forward elimination and backward substitution) *using the already LU factored matrix*. That is, the computational cost is $O(nnz(\mathbf{L}))$, where $nnz(\mathbf{L})$ denotes the number of non-zeros of the Cholesky factorization $\mathbf{L}$ of the stiffness matrix $\mathbf{A}$. This algorithm using the direct sparse solver is faster than the Fourier accelerated preconditioned conjugate gradient (PCG) iterative solvers, and eliminates the *critical slowing down* associated with the iterative solvers that is especially severe close to the critical points. Numerical results using random resistor networks substantiate the efficiency of the present algorithm.

PACS numbers: 62.20.Mk, 46.50.+a

1. Introduction

Progressive damage evolution leading to failure of disordered quasi-brittle materials has been studied extensively using various types of discrete lattice models [1–8]. Large-scale numerical

simulation of these lattice networks in which the damage is accumulated progressively by breaking one bond at a time until the lattice system falls apart has often been hampered due to the fact that a new large set of linear equations has to be solved everytime a lattice bond is broken. Since the number of broken bonds at failure, n_f, increases with increasing lattice system sizes, L, i.e. $n_f \sim O(L^{1.7})$, numerical simulation of large lattice systems becomes prohibitively expensive. Furthermore, in fracture simulations using discrete lattice networks, ensemble averaging of numerical results is necessary to obtain a realistic representation of the lattice system response. This further increases the computational cost associated with modelling fracture simulations in disordered quasi-brittle materials using large discrete lattice networks.

Fourier accelerated PCG iterative solvers [9–11] have been used in the past for simulating the material breakdown using large lattices. However, these methods do not completely eliminate the *critical slowing down* associated with the iterative solvers close to the critical point. As the lattice system gets closer to macroscopic fracture, the condition number of the system of linear equations increases, thereby increasing the number of iterations required to attain a fixed accuracy. This becomes particularly significant for large lattices. Furthermore, the Fourier acceleration technique is not effective when fracture simulation is performed using central-force and bond-bending lattice models [10].

This study presents an algorithm that combines the multiple-rank sparse Cholesky downdating scheme with the rank-p inverse updating scheme of the stiffness matrix, which effectively reduces the computational bottleneck involved in re-solving the new set of equations after everytime a bond is broken. In this paper, we consider a *random threshold* model problem, where a lattice consists of fuses having the same conductance, but the bond breaking thresholds, i_c, are based on a broad (uniform) probability distribution, which is constant between 0 and 1. This relatively simple model has been extensively used in the literature [1–7] for simulating the fracture and progressive damage evolution in brittle materials, and provides a meaningful benchmark for comparing different algorithms. A broad thresholds distribution represents large disorder and exhibits diffusive damage leading to progressive localization, whereas a very narrow thresholds distribution exhibits brittle failure in which a single crack propagation causes material failure. Periodic boundary conditions are imposed in the horizontal direction to simulate an infinite system, and a constant voltage difference (displacement) is applied between the top and the bottom of lattice system. The simulation is initiated with a triangular lattice of intact fuses of size $L \times L$, in which disorder is introduced through random breaking thresholds. The voltage V across the lattice system is increased until a fuse (bond breaking) burns out. The burning of a fuse occurs whenever the electrical current (stress) in the fuse (bond) exceeds the breaking threshold current (stress) value of the fuse. The current is redistributed instantaneously after a fuse is burnt. The voltage is then gradually increased until a second fuse is burnt, and the process is repeated. Each time a fuse is removed, the electrical current is redistributed and hence it is necessary to re-solve Kirchhoff equations to determine the current flowing in the remaining bonds of the lattice. This step is essential for determining the fuse that is going to burn up under the redistributed currents. Therefore, numerical simulations leading to final breaking of lattice system network are very time consuming especially with increasing lattice system size.

1.1. Summary of the proposed algorithm

The algorithm presented in this paper reduces the computational complexity of obtaining the solution $\mathbf{x}_n$, after the nth bond is broken, to a backsolve using the already existent factorization of the stiffness matrix $\mathbf{A}_m$, and $p = (n - m)$ vector updates. The algorithm is based on the

well-known Shermon–Morrison–Woodbury formula [12] for obtaining the inverse of the new stiffness matrix $\mathbf{A}_{n+1}^{-1}$ (after the $(n+1)$th fuse is burnt) from the old stiffness matrix inverse $\mathbf{A}_n^{-1}$ through a rank-1 update. Infact, the algorithm is such that if the inverse of the lattice stiffness at any stage ($m = 0, 1, 2, \ldots$) of analysis $\mathbf{A}_m^{-1}$ is available, then all subsequent analysis involving ($n = m + 1, m + 2, \ldots$) burnt fuses can be carried out using $p = (n - m)$ vector updates. However, since most often the inverse of the stiffness matrix is rarely ever explicitly calculated, the algorithm additionally requires a backsolve using the already existent factored matrix $\mathbf{A}_m$. The backsolve operation is further simplified by the fact that it is performed on a trivial load vector and hence the solution can be obtained easily.

Based on the above description of the algorithm presented in this paper, given the factorization of the matrix $\mathbf{A}_m$, the computational cost involved in all the subsequent steps ($n = m + 1, m + 2, \ldots$) is a backsolve using the already factored matrix, and $p = (n - m)$ vector updates. The computational complexity of the backsolve is $O(nnz(\mathbf{L}_m))$, where $nnz(\mathbf{L}_m)$ denotes the number of non-zeros of the Cholesky factorization $\mathbf{L}_m$ of $\mathbf{A}_m$. The computational complexity of p vector updates is $O(pn_{\text{dof}})$, where n_{dof} denotes the number of degrees of freedom in the system. As p increases, it is possible that the computational cost associated with the p vector updates exceeds the cost involved in the factorization of the matrix $\mathbf{A}_n$. Under these circumstances, it is advantageous to obtain the factorization $\mathbf{L}_n$ of the new stiffness matrix $\mathbf{A}_n$, and use this $\mathbf{L}_n$ for all the subsequent backsolve analysis steps, until the computational cost associated with the vector updates once again exceeds the stiffness factorization cost. Using the algorithm presented in the paper, it is not necessary to re-factorize the new stiffness matrix $\mathbf{A}_n$. Instead, we adopt the multiple-rank update of the sparse Cholesky factorization algorithm [13, 14] for updating the $\mathbf{L}_m \rightarrow \mathbf{L}_n$. This multiple-rank update of $\mathbf{L}_m$ to obtain the new factorization $\mathbf{L}_n$ is computationally cheaper compared with the direct factorization $\mathbf{L}_n$ of the new stiffness matrix $\mathbf{A}_n$ [13, 14].

2. Proposed algorithm

In the following, we describe the updating scheme for the inverse of the stiffness matrix in the case of scalar random fuse model after a fuse has been burnt. A similar procedure can be applied for central-force and beam models [15].

Let $\mathbf{A}_n$ represent the stiffness matrix of the random fuse network system in which n number of fuses are either missing (random dilution) or have been burnt during the analysis. Let us also assume that a fuse ij (the $(n+1)$th fuse) is burnt when the externally applied voltage is increased gradually. In the above description, i and j refer to the global degrees of freedom connected by the fuse before it is broken. For the scalar random fuse model, the degrees of freedom i and j are also equivalent to the node i and node j connected by the fuse before it is broken. The new stiffness matrix $\mathbf{A}_{n+1}$ of the lattice system after the fuse ij is burnt is given by

$$\mathbf{A}_{n+1} = \mathbf{A}_n - k_{ij}\mathbf{v}\mathbf{v}^t \tag{1}$$

where

$$\mathbf{v}^t = \{0 \quad \cdots \quad 1 \quad \cdots \quad -1 \quad \cdots \quad 0\} \tag{2}$$

and k_{ij} is the conductance of the fuse ij before it is broken. After breaking the fuse ij, the electrical current in the network is redistributed instantaneously. The redistributed current values in the network are calculated by re-solving the Kirchhoff equations, i.e. by solving the new set of equations formed by the matrix $\mathbf{A}_{n+1}$. This procedure is very time consuming since a new set of equations (inverse of $\mathbf{A}_{n+1}$ for $n = 0, 1, 2, \ldots$) need to be solved everytime after

breaking the $(n+1)$th fuse. However, significant computational advantages can be gained if the inverse of $\mathbf{A}_{n+1}$ is obtained simply by updating the inverse of $\mathbf{A}_n$. This is achieved by using the well-known Shermon–Morrison–Woodbury formula for inverting the rank-p update of a matrix. Thus, the inverse $\mathbf{A}_{n+1}^{-1}$ of equation (1) can be expressed as

$$\mathbf{A}_{n+1}^{-1} = \left[\mathbf{A}_n^{-1} + k_{ij}\frac{\mathbf{u}\mathbf{u}^t}{(1 - k_{ij}\mathbf{v}^t\mathbf{u})}\right] \tag{3}$$

where

$$\mathbf{u} = \mathbf{A}_n^{-1}\mathbf{v} = \mathbf{A}_{n\ (i-j)}^{-1} = \left[(i\text{th} - j\text{th}) \text{ columns of } \mathbf{A}_n^{-1}\right]. \tag{4}$$

Hence, the inverse of the stiffness matrix of the lattice system after breaking the $(n+1)$th fuse ij is obtained simply by a rank-1 update of the inverse of the stiffness matrix before the fuse is broken. Further, if the inverse of the matrix $\mathbf{A}_n$ is available explicitly, then the vector $\mathbf{u}$ can be obtained trivially from the ith and jth columns of $\mathbf{A}_n^{-1}$. In particular, this implies that if the inverse of the matrix $\mathbf{A}_n$ is available explicitly at any stage $n = 0, 1, 2, \ldots$ of analysis, then the redistributed currents in all subsequent stages of analysis involving $m = n+1, n+2, \ldots$ burnt fuses can be obtained in a trivial fashion from the column vectors of $\mathbf{A}_n^{-1}$ and the vectors $\mathbf{u}_p$, where $p = 1, 2, \ldots, (m-n)$. However, since the inverse of the stiffness matrix $\mathbf{A}_n$ is not usually calculated explicitly, the vector $\mathbf{u}$ is obtained using the already factorized $\mathbf{A}_n$ matrix through a backsolve operation (forward reduction and backward substitution) on the vector $\mathbf{v}$ (equation (4)).

Remark 1. Without loss of generality, when the fuse that is broken is attached to a *constrained/prescribed* degree of freedom j, the vector $\mathbf{v}$ is given by

$$\mathbf{v}^t = \{0 \quad \cdots \quad 1 \quad \cdots \quad 0\} \tag{5}$$

and

$$\mathbf{u} = \mathbf{A}_{n\ (i)}^{-1} = \left[i\text{th columns of } \mathbf{A}_n^{-1}\right]. \tag{6}$$

In the case of periodic boundary conditions, consider the case of a broken fuse jk that is attached to a slave degree of freedom k whose master degree of freedom is i. Under these circumstances, the methodology presented earlier is applicable in a straightforward manner if it is understood that breaking the fuse jk is equivalent to breaking the fuse ij.

Remark 2. The load vector $\mathbf{b}_{n+1}$ will differ from the load vector $\mathbf{b}_n$ only if the $(n+1)$th broken fuse ij is attached to a prescribed degree of freedom, where a constant voltage difference is imposed. Once again, for presentation purposes, let us assume that j is such a prescribed degree of freedom. Then the load vector $\mathbf{b}_{n+1}$ is given by

$$\mathbf{b}_{n+1} = \mathbf{b}_n + \mathbf{w} \tag{7}$$

where

$$\mathbf{w}^t = k_{ij}\{0 \quad 0 \quad \cdots \quad -1 \quad \cdots \quad 0 \quad 0\}. \tag{8}$$

If neither i nor j is a prescribed degree of freedom, then $\mathbf{w} = \mathbf{0}$.

Before breaking the $(n+1)$th fuse, the solution vector $\mathbf{x}_n$ is obtained by solving the Kirchhoff equations

$$\mathbf{A}_n\mathbf{x}_n = \mathbf{b}_n. \tag{9}$$

After breaking the $(n+1)$th fuse that connects the ith and jth degrees of freedom, the updated solution vector $\mathbf{x}_{n+1}$ is obtained by solving the new set of Kirchhoff equations

$$\mathbf{A}_{n+1}\mathbf{x}_{n+1} = \mathbf{b}_{n+1}. \tag{10}$$

Substituting equations (3), (7) and (9) into the solution of equation (10) and simplifying the result, we have

$$\begin{aligned}\mathbf{x}_{n+1} &= \mathbf{A}_{n+1}^{-1}\mathbf{b}_{n+1} \\ &= \left[\mathbf{A}_n^{-1} + k_{ij}\frac{\mathbf{u}\mathbf{u}^t}{(1 - k_{ij}\mathbf{v}^t\mathbf{u})}\right](\mathbf{b}_n + \mathbf{w}) \\ &= \mathbf{x}_n + \alpha(\mathbf{u}^t\mathbf{b}_{n+1})\mathbf{u} \end{aligned} \tag{11}$$

where

$$\begin{aligned}\alpha &= \frac{k_{ij}}{(1 - k_{ij}\mathbf{v}^t\mathbf{u})} - k_{ij} \qquad \text{if } i \text{ or } j \text{ is prescribed} \\ &= \frac{k_{ij}}{(1 - k_{ij}\mathbf{v}^t\mathbf{u})} \qquad \text{otherwise.}\end{aligned} \tag{12}$$

The only unknown in equation (11) is the column vector $\mathbf{u}$, which can be obtained through a backsolve operation using either equation (4) or equation (6). Furthermore, it is not necessary to explicitly assemble the matrix $\mathbf{A}_n$ and perform factorization to do the backsolve operation. Instead, we can use the already factorized matrix $\mathbf{A}_m$ to obtain the vector $\mathbf{u}$. In the above description, $m < n$ and denotes the latest broken bond at which the factorization $\mathbf{L}_m$ of $\mathbf{A}_m$ is available. To see this clearly, let us first decompose the matrix $\mathbf{A}_n^{-1}$ into $\mathbf{A}_m^{-1}$ and a matrix $\mathbf{C}$ such that

$$\mathbf{A}_n^{-1} = \mathbf{A}_m^{-1} + \mathbf{C} \tag{13}$$

where

$$\mathbf{C} = \sum_{l=1}^{p=(n-m)} k_l \frac{\mathbf{u}_l\mathbf{u}_l^t}{\left(1 - k_l\mathbf{v}_l^t\mathbf{u}_l\right)}. \tag{14}$$

Due to the amount of the storage requirement $\left(\sim O\left(n_{\mathrm{dof}}^2\right)\right)$, and the computational cost associated in evaluating the equation (14) $\left(\sim O\left(n_{\mathrm{dof}}^2\right)\right)$, the matrix $\mathbf{C}$ is never explicitly calculated or stored. Instead, the vectors $\mathbf{u}_l$ for $l = 1, 2, \ldots, (n - m)$ are stored, and the $(j\text{th} - i\text{th})$ column of $\mathbf{C}$ is evaluated as

$$\mathbf{C}_{(j-i)} = \sum_{l=1}^{p=(n-m)} k_l \frac{\left(\mathbf{u}_{l_j} - \mathbf{u}_{l_i}\right)}{\left(1 - k_l\mathbf{v}_l^t\mathbf{u}_l\right)}\mathbf{u}_l \tag{15}$$

where $\mathbf{u}_{l_i}$ and $\mathbf{u}_{l_j}$ refer to the ith and jth components of the vector $\mathbf{u}_l$. Equation (15) reduces the storage and computational cost to $(\sim O(pn_{\mathrm{dof}}))$ and $(\sim O(pn_{\mathrm{dof}}))$ operations, respectively. Even with this modification, the storage and computational requirements can become prohibitively expensive as the number of updates, p, increases, and hence it is necessary to limit the maximum number of vector updates between two successive factorizations to a certain *maxupd*. That is, it is necessary to perform or update the factorization of the stiffness matrix $\mathbf{A}$ at regular intervals.

Instead of re-factorizing the stiffness matrix $\mathbf{A}$ after every *maxupd* steps, it is more effective to update the factorization $\mathbf{L}_m$ using the multiple-rank sparse Cholesky factorization update algorithm [13, 14]. This multiple-rank update of $\mathbf{L}_m$ to obtain the new factorization $\mathbf{L}_{n+1}$, after breaking the $(n + 1)$th fuse, is computationally cheaper compared with the direct factorization $\mathbf{L}_{n+1}$ of the new stiffness matrix $\mathbf{A}_{n+1}$ [13, 14]. We use the multiple-rank downdate algorithm presented in [13, 14] to obtain the new Cholesky factorization $\mathbf{L}_{n+1}$ from the existing Cholesky factor $\mathbf{L}_m$. The multiple-rank downdate algorithm [13, 14] is based on the analysis and manipulation of the underlying graph structure of the stiffness matrix $\mathbf{A}$

and on the methodology presented in Gill *et al* [16, 17] for modifying a dense Cholesky factorization. This algorithm incorporates the change in the sparsity pattern of **L** and is optimal in the sense that the computational time required is proportional to the number of changing non-zero entries in **L**. In particular, since the breaking of fuses is equivalent to removing the edges in the underlying graph structure of stiffness matrix **A**, the new sparsity pattern of the modified **L** must be a subset of the sparsity pattern of the original **L**. Denoting the sparsity pattern of **L** by $\mathcal{L}$, we have

$$\mathcal{L}_m \supseteq \mathcal{L}_n \qquad \forall m < n. \tag{16}$$

Therefore, we can even use the modified dense Cholesky factorization update (algorithm 5 in the Davis and Hager [13]) and work only on the non-zero entries in **L**. Furthermore, since the changing non-zero entries in **L** depend on the ith and jth degrees of freedom of the fuse ij that is broken, it is only necessary to modify the non-zero elements of a submatrix of **L**.

The multiple-rank update of the sparse Cholesky factorization is computationally superior to an equivalent series of rank-1 updates since the multiple-rank update makes one pass through **L** in computing the new entries, while a series of rank-1 updates require multiple passes through **L** [14]. The multiple-rank update algorithm updates the Cholesky factorization $\mathbf{L}_m$ of the matrix $\mathbf{A}_m$ to $\mathbf{L}_{n+1}$ of the new matrix $\mathbf{A}_{n+1}$, where $\mathbf{A}_{n+1} = \mathbf{A}_m + \sigma\mathbf{Y}\mathbf{Y}^t$, and $\mathbf{Y}$ represents a $n_{\text{dof}} \times p$ rank-p matrix. The computational cost involved in breaking the $(n+1)$th fuse ij is simply a backsolve operation ($O(nnz(\mathbf{L}_m))$) on a load vector given by equation (2) using the already factored matrix $\mathbf{A}_m$, $(n+2-m)$ vector updates, and one vector inner product.

The optimum number of steps between successive factorizations of the matrix **A** is determined by minimizing the computational CPU time required for the entire analysis. Let t_{fac} and t_{upd} denote the average CPU time required for performing/updating the factorization $\mathbf{A}_m$ and the average CPU time required for a single rank-1 update of the solution $\tilde{\mathbf{u}}_{(n+1-m)}$, respectively. Note that the evaluation of $\tilde{\mathbf{u}}_{(n+1-m)}$ requires $(n-m)$ vector updates. Let the estimated number of steps for the lattice system failure be n_{steps}. Then, the total CPU time required for solving the linear system of equations until the lattice system failure is given by

$$\begin{aligned}\Psi &= n_{\text{fac}} t_{\text{fac}} + \sum \frac{(n_{\text{steps}} - n_{\text{fac}})}{n_{\text{fac}}} t_{\text{upd}} \\ &= n_{\text{fac}} t_{\text{fac}} + \frac{1}{2}\frac{(n_{\text{steps}} - n_{\text{fac}})}{n_{\text{fac}}}\frac{n_{\text{steps}}}{n_{\text{fac}}} t_{\text{upd}}\end{aligned} \tag{17}$$

where n_{fac} denotes the number of factorization until lattice system failure. The optimum number of factorizations, $n_{\text{opt-fac}}$, for the entire analysis is obtained by minimizing the function Ψ. The maximum number of vector updates, *maxupd*, between successive factorizations is estimated as

$$maxupd = \frac{(n_{\text{steps}} - n_{\text{opt-fac}})}{n_{\text{opt-fac}}} \tag{18}$$

3. Numerical simulation results

In the following, we consider two alternate forms of the algorithm presented in this paper. These two solver types are

- *Solver type A*. Given the factorization $\mathbf{L}_m$ of $\mathbf{A}_m$, we use rank-1 sparse Cholesky update/downdate [13] to update the factorization $\mathbf{L}_{n+1}$ ($O(nnz(\mathbf{L}_n))$ for all subsequent values of $n = m, m+1, \ldots$. Once the factorization $\mathbf{L}_{n+1}$ of $\mathbf{A}_{n+1}$ is obtained, the solution vector $\mathbf{x}_{n+1}$ is obtained by a backsolve operation ($O(nnz(\mathbf{L}_{n+1}))$.

Table 1. Computational cost associated with solver type A.

Size	CPU (s)	Wall (s)	Simulations
32	0.592	0.687	20 000
64	10.72	11.26	4 000
128	212.2	214.9	800
256	5647	5662	96
512	93779	96515	16

Table 2. Computational cost associated with solver type B.

Size	CPU (s)	Wall (s)	Simulations
32	0.543	0.633	20 000
64	11.15	12.01	4 000
128	211.5	214.1	800
256	6413	6701	96

Table 3. Computational cost associated with *optimal* circulant PCG.

Size	CPU (s)	Wall (s)	Iterations	Simulations
32	11.66	12.26	25 469	20 000
64	173.6	178.8	120 570	1600
128	7473	7725	622 140	128

- *Solver type B*. Given the factorization $\mathbf{L}_m$ of $\mathbf{A}_m$, the algorithm evaluates the new solution vector $\mathbf{x}_{n+1}$, after the $(n+1)$th fuse is burnt, using equation (11) ($O(nnz(\mathbf{L}_m) + (n + 2 - m)$ vector updates). Instead of refactorizing the matrix after *maxupd* steps, we use rank-p sparse Cholesky update/downdate [14] to obtain the factorization $\mathbf{L}_{m+maxupd}$ of the matrix $\mathbf{A}_{m+maxupd}$ ($O(nnz(\mathbf{L}_m))$.

The above two algorithms are benchmarked against the PCG iterative solvers, in which *optimal* [18–21] circulant matrices are used as preconditioners to the Laplacian operator (Kirchhoff equations). The Fourier accelerated PCG presented in [9–11] is not optimal in the sense described in [18–21], and hence it is expected to take more number of CG iterations compared with the *optimal* circulant preconditioners.

In the numerical simulations using solver types A and B, the supernodal Cholesky factorization is performed using the TAUCS solver library (http://www.tau.ac.il/ stoledo/taucs). In these simulations, the maximum number of vector updates, *maxupd*, is chosen to be a constant for a given lattice size. We choose $maxupd = 25$ for $L = \{4, 8, 16, 24, 32\}$, $maxupd = 50$ for $L = 64$, and $maxupd = 100$ for $L = \{128, 256, 512\}$. For $L = 512$, *maxupd* is limited to 100 due to memory constraints. By keeping the *maxupd* value constant, it is possible to realistically compare the computational cost associated with different solver types. Moreover, the relative CPU times taken by these algorithms remains the same even when the simulations are performed on different platforms.

Tables 1 and 2 present the CPU and wall-clock times taken for one configuration (simulation) using the solver types A and B, respectively. These tables also indicate the number of configurations, N_{config}, over which ensemble averaging of the numerical results is performed. The CPU and wall-clock times taken by the *optimal* circulant matrix preconditioned iterative solver is presented in tables 3. For iterative solvers, the number of iterations presented in tables 3 denote the average number of total iterations taken to break one intact lattice configuration until it falls apart.

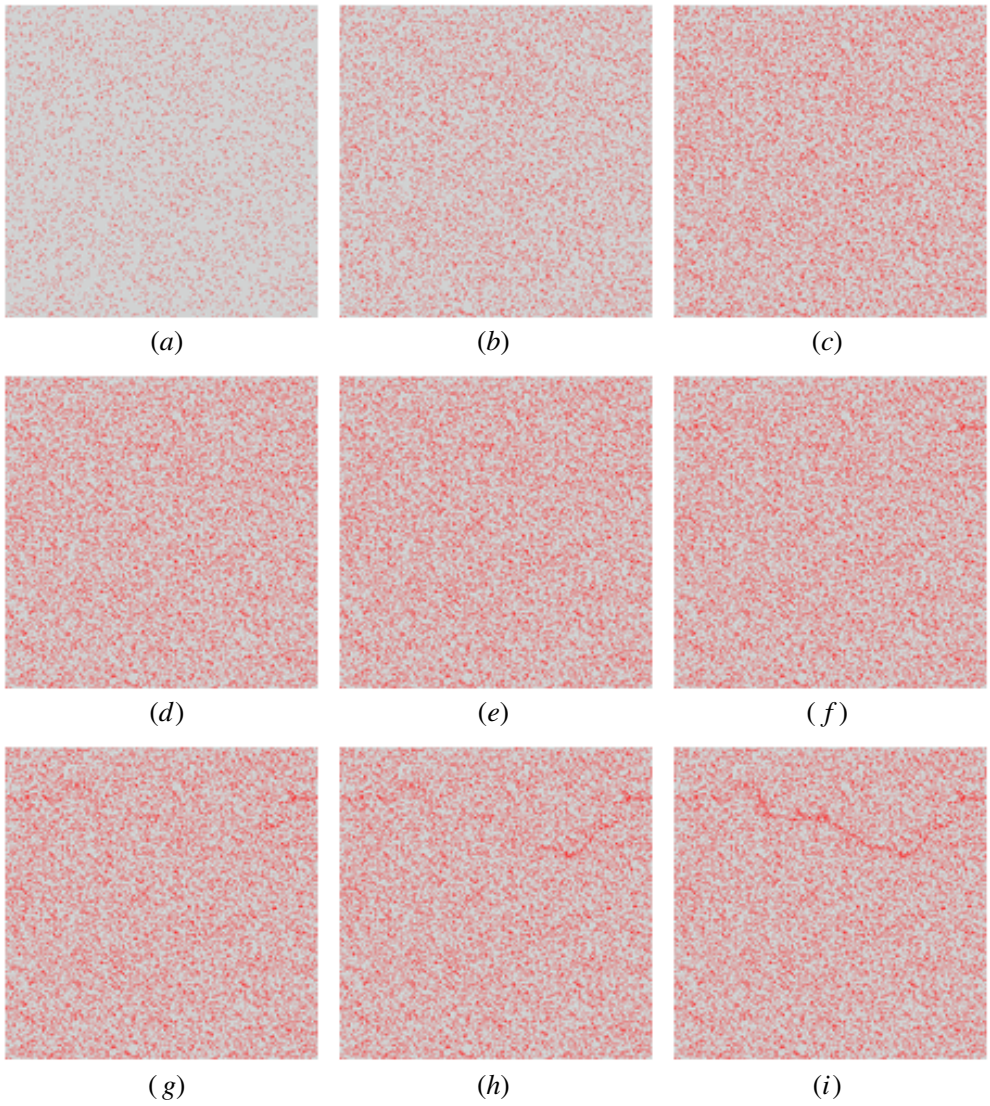

Figure 1. Snapshots of damage in a typical triangular lattice system of size $L = 512$. Number of broken bonds at the peak load and at failure are 83 995 and 89 100, respectively. (*a*)–(*i*) represent the snapshots of damage after breaking n_b number of bonds. (*a*) $n_b = 25\,000$, (*b*) $n_b = 50\,000$, (*c*) $n_b = 75\,000$, (*d*) $n_b = 80\,000$, (*e*) $n_b = 83\,995$ (peak load), (*f*) $n_b = 86\,000$, (*g*) $n_b = 87\,000$, (*h*) $n_b = 88\,000$ and (*i*) $n_b = 89\,100$ (failure).

Based on the results presented in tables 1–3, it is clear that for modelling the breakdown of disordered media as in starting with an intact lattice and successive breaking of bonds until the lattice system falls apart, the solver types A and B based on direct solvers are superior to the Fourier accelerated iterative PCG solver techniques. It should be noted that for larger lattice systems, limitations on the available memory of the processor may decrease the allowable *maxupd* value, as in the case of $L = 512$ using solver type B. However, this is not a concern for simulations performed using solver type A.

Table 4. Number of broken bonds at peak and at failure.

			Triangular		Diamond	
L	N_{config}	Time (s)	n_p	n_f	n_p	n_f
4	50 000	0.002	13	19	9	14
8	50 000	0.006	41	54	26	37
16	50 000	0.042	134	168	80	107
24	50 000	0.186	276	335	161	208
32	50 000	0.592	465	554	268	337
64	50 000	10.72	1 662	1 911	942	1 126
128	12 000	212.2	6 068	6 766	3 406	3 901
256	1 200	5 647	22 572	24 474	12 571	13 846
512	200	93 779	84 487	89 595		

Using the solver type A, we have performed numerical simulations on two-dimensional triangular and diamond (square lattice inclined at 45° between the bus bars) lattice networks. Table 4 presents the number of broken bonds at peak load, n_p, and at fracture, n_f, for each of the lattice sizes considered. In addition, table 4 also presents the number of configurations, N_{config}, over which statistical averaging is performed for different lattice sizes. The numerical results presented in tables 1–3 are performed on a single processor of Cheetah (27 Regatta nodes with 32 1.3 GHz Power4 processors each), the eighth fastest supercomputer in the world (http://www.ccs.ornl.gov). However, the numerical simulation results presented in table 4 are performed on Eagle (184 nodes with four 375 MHz Power3-II processors) supercomputer at the Oak Ridge National Laboratory to run simulations simultaneously on more number of processors. Figure 1 presents the snapshots of progressive damage evolution for the case of a broadly distributed random thresholds model problem in a triangular lattice system of size $L = 512$.

4. Conclusions

The paper presents an algorithm based on rank-1 update of the inverse of the stiffness matrix and the multiple-rank downdating of the sparse Cholesky factorization for simulating fracture and damage evolution in disordered quasi-brittle materials using discrete lattice networks. Using the proposed algorithm, the average computational cost associated with breaking a bond reduces to the same order as that of a simple backsolve (forward elimination and backward substitution) operation using the already LU factored matrix. This algorithm based on direct solver techniques eliminates critical slowing down observed in fracture simulations using the conventional iterative schemes. Numerical simulations on random resistor networks demonstrate that the present algorithm is computationally superior to the commonly used Fourier accelerated preconditioned conjugate gradient iterative solver.

For analysis of fracture simulations using discrete lattice networks, ensemble averaging of numerical results is necessary to obtain a realistic representation of the lattice system response. In this regard, for very large lattice systems with large number of system of equations, this methodology is especially advantageous as the LU factorization of the system of equations can be performed using a parallel implementation on multiple processors. Subsequently, this factored LU decomposition can then be distributed to each of the processors to continue with independent fracture simulations that only require less intensive backsolve operations.

Acknowledgment

This research is sponsored by the Mathematical, Information and Computational Sciences Division, Office of Advanced Scientific Computing Research, US Department of Energy under contract number DE-AC05-00OR22725 with UT-Battelle, LLC. The first author wishes to thank Ed F D'Azevedo for many helpful discussions and excellent suggestions.

References

[1] de Arcangelis L, Redner S and Herrmann H J 1985 A random fuse model for breaking processes *J. Phys. (Paris) Lett.* **46** 585–90
[2] Sahimi M and Goddard J D 1986 Elastic percolation models for cohesive mechanical failure in heterogeneous systems *Phys. Rev.* B **33** 7848–51
[3] Duxbury P M, Beale P D and Leath P L 1986 Size effects of electrical breakdown in quenched random media *Phys. Rev. Lett.* **57** 1052–5
[4] Duxbury P M, Leath P L and Beale P D 1987 Breakdown properties of quenched random systems: the random-fuse network *Phys. Rev.* B **36** 367–80
[5] Hansen A and Roux S 2000 *Statical Toolbox for Damages and Fracture* (New York: Springer) pp 17–101
[6] Herrmann H J and Roux S (ed) 1990 *Statistical Models for the Fracture of Disordered Media* (Amsterdam: North-Holland)
[7] Sahimi M 1998 Non-linear and non-local transport processes in heterogeneous media from long-range correlation percolation to fracture and materials breakdown *Phys. Rep.* **306** 213–395
[8] Chakrabarti B K and Gilles Benguigui L 1997 *Statistical Physics of Fracture and Breakdown in Disordered Systems* (Oxford: Oxford Science Publications)
[9] Batrouni G G, Hansen A and Nelkin M 1986 Fourier acceleration of relaxation processes in disordered systems *Phys. Rev. Lett.* **57** 1336–9
[10] Batrouni G G and Hansen A 1988 Fourier acceleration of iterative processes in disordered-systems *J. Stat. Phys.* **52** 747–73
[11] Batrouni G G and Hansen A 1998 Fracture in three-dimensional fuse networks *Phys. Rev. Lett.* **80** 325–8
[12] Golub G H and van Loan C F 1996 *Matrix Computations* (Baltimore, MD: The Johns Hopkins University Press)
[13] Davis T A and Hager W W 1999 Modifying a sparse Cholesky factorization *SIAM J. Matrix Anal. Appl.* **20** 606–27
[14] Davis T A and Hager W W 2001 Multiple-rank modifications of a sparse Cholesky factorization *SIAM J. Matrix Anal. Appl.* **22** 997–1013
[15] Nukala P K V V and Simunovic S unpublished
[16] Gill P E, Golub G H, Murray W and Saunders M A 1974 Methods for modifying matrix factorizations *Math. Comp.* **28** 505–35
[17] Gill P E, Murray W and Saunders M A 1975 Methods for computing and modifying the LDV factors of a matrix *Math. Comp.* **29** 1051–77
[18] Chan T 1988 An optimal circulant preconditioner for Toeplitz systems *SIAM J. Sci. Stat. Comput.* **9** 766–71
[19] Chan R H 1989 Circulant preconditioners for Hermitian Toeplitz systems *SIAM J. Matrix Anal. Appl.* **10** 542–50
[20] Chan R and Chan T 1992 Circulant preconditioners for elliptic problems *Numer. Linear Algebra Appl.* **1** 77–101
[21] Chan R H and Ng M K 1996 Conjugate gradient methods for Toeplitz systems *SIAM Rev.* **38** 427–82