
Scaling Laws for Damage Evolution in Disordered Materials: Numerical Aspects

Srdjan Simunovic

Phani Kumar V.V. Nukala

Oak Ridge National Laboratory

Computer Science and Mathematics Division

simunovics@ornl.gov

<http://www-cms.ornl.gov>

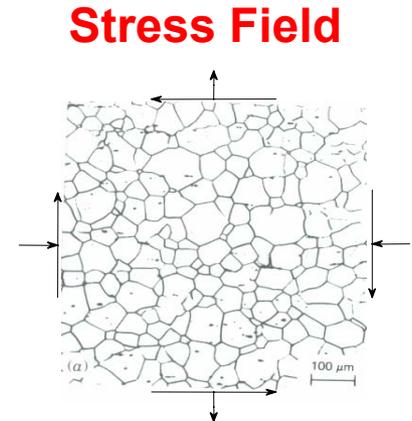
Outline

- Motivation
- Modeling of damage evolution in materials and material breakdown
- Numerical aspects
- Developed algorithm
- Conclusions

Damage Evolution and Fracture in Disordered Materials

Motivation: Stress Induced Microcracking Evolution

Macroscopic properties and behavior of quasi-brittle materials are significantly effected by the internal microstructure and damage/microcracking evolution



Microcracking evolution

Controlling of microstructure state and damage evolution leads to improved macroscopic behavior

Modeling at the mesoscale will lead to a fundamental understanding of the effect of microstructural features on the microcracking evolution in brittle materials

Objective

Mesoscale System Response:

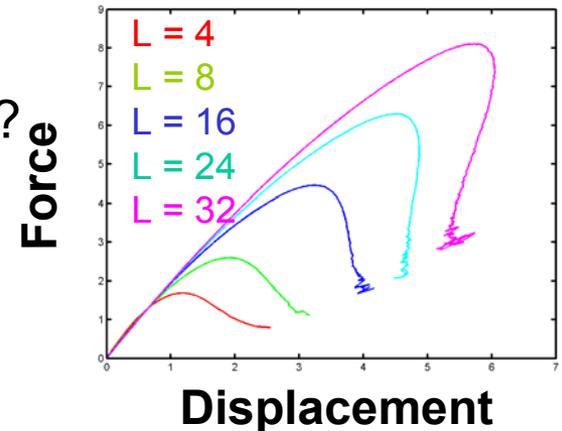
- depends on the system size
- computationally intractable



Scaling Laws

Open Questions?:

- How does a disordered solid breakdown?
- What is the size effect on failure?
- What are the scaling laws of failure?
- How does one quantify damage? and how do we compare the extent of damage between two specimens?
- What is the connection between mesoscopic damage and the phenomenological continuum damage evolution?



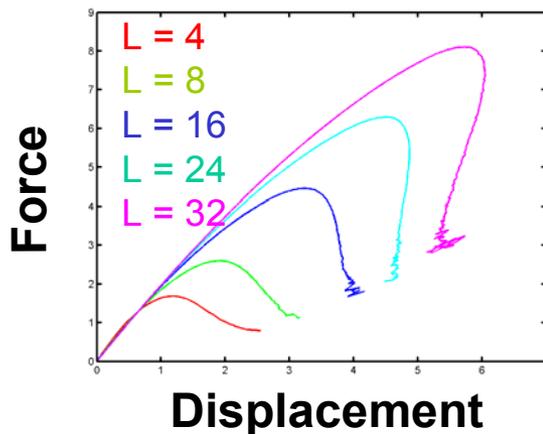
Objective:

- Describe continuum damage evolution based on mesoscopic modeling using scaling laws

Current Status of Material Models

Phenomenological Material Models:

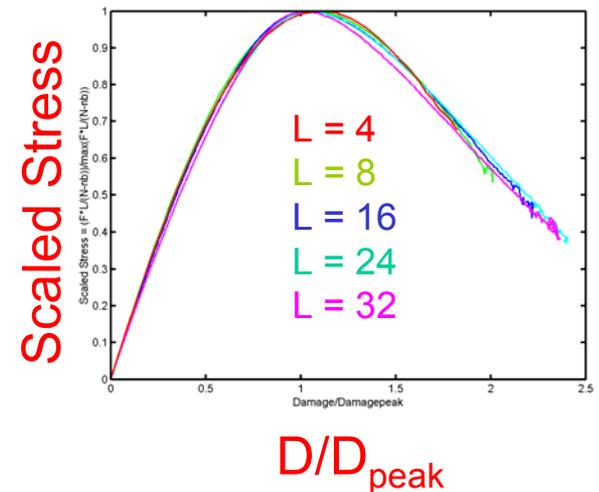
- progressive damage and cracking are microstructure-insensitive
- based on simplified assumptions for the evolution of damage
 - valid only for moderate damage levels
 - local stress field fluctuations and interactions are not considered



Material response is
size dependent



Scaling laws are required
to obtain a “normalized”
response that couple
mesoscopic and continuum
length scales



Solution:

Explicit modeling of material **microstructure** combined with the **scaling theory** accounts for **size effects** and local stress field interactions during damage/microcrack evolution

Numerical Methodology

Mesososcopic Simulation: Discrete Lattice Models

Focus of the study is not on any particular material

Focus is on capturing the generic features of damage evolution

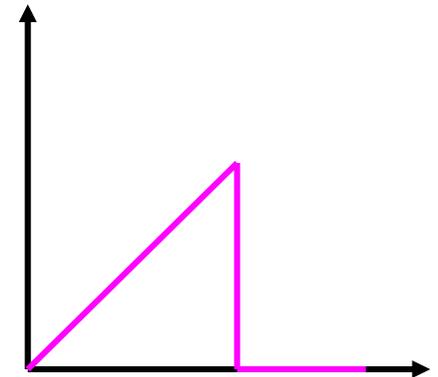
Essential ingredients of breakage process:

- Initial material disorder (inhomogeneities)
- redistribution of stresses due to damage evolution

Discrete Lattice Models:

- disorder in bond strength and stiffness
- elastic response characteristics of the bonds
- bond breaking rule (failure criteria)

Perfectly brittle bond

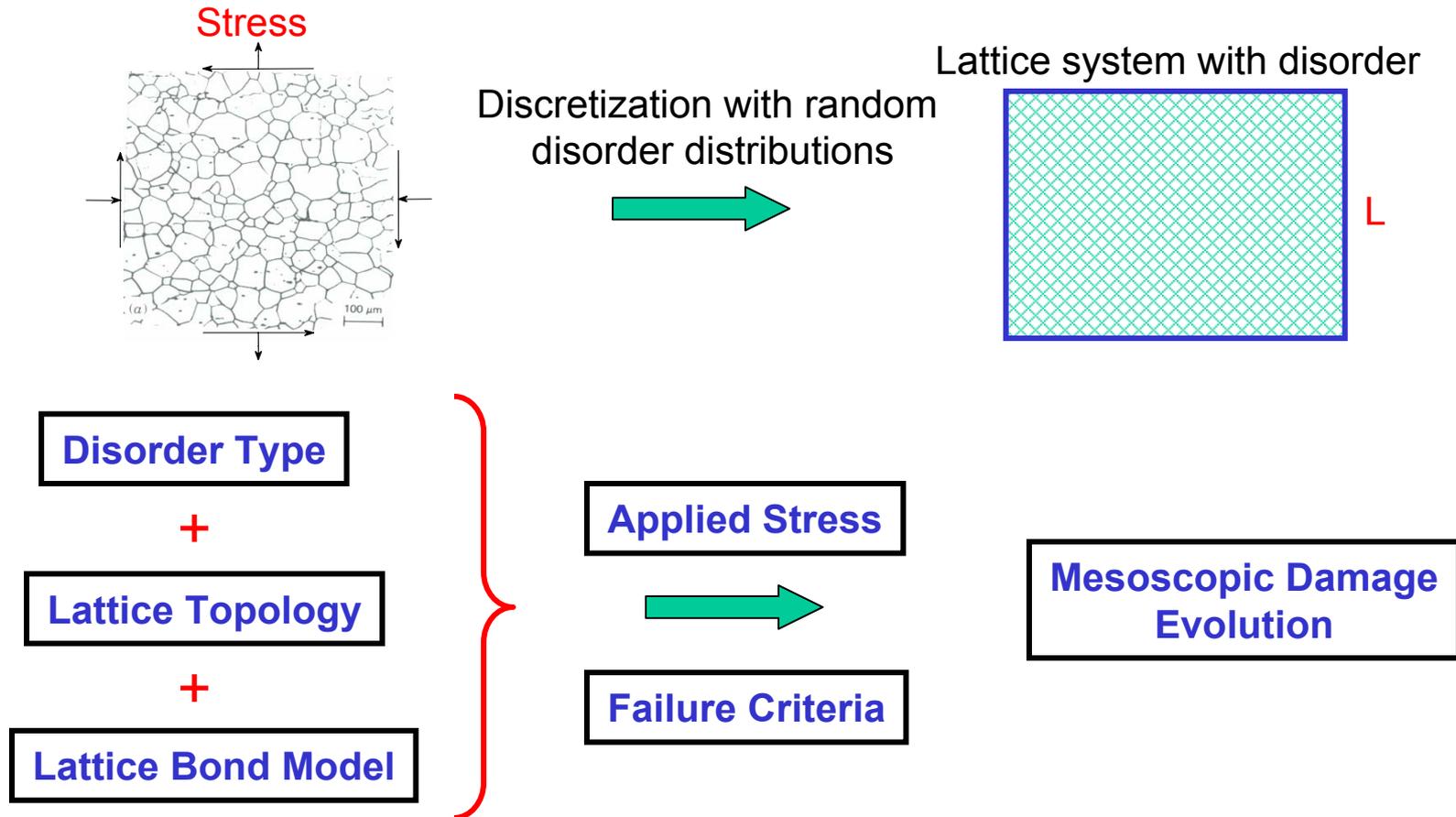


Any realistic damage evolution description must be capable of reproducing the behavior of these idealized discrete lattice models

Mesososcopic Modeling Approach

Failure of a bond is governed by

- weakest bond of the disordered medium
- stress concentration around material inhomogenities



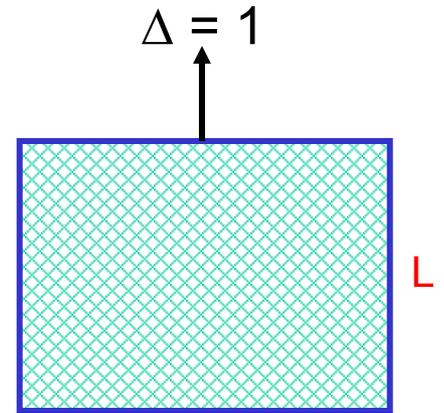
Analysis Procedure

Procedure:

Step 0: For each bond in the lattice system, assign unit stiffness and random force threshold f_i^{th}

Step 1: Impose a unit macroscopic displacement

Step 2: Calculate the force f_i in each bond through lattice equilibrium



Lattice system with disorder

$$\mathbf{K} = \sum_i \mathbf{f}_i^2 \quad \mathbf{K} \text{ Global stiffness}$$

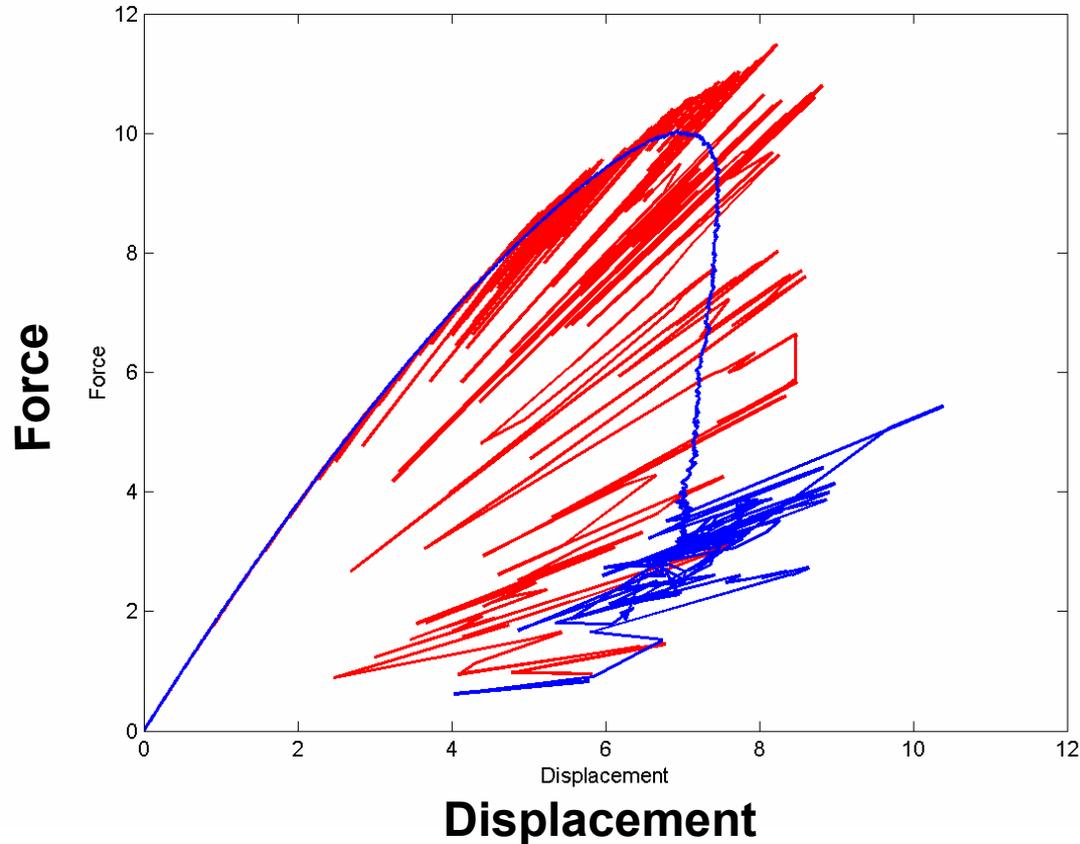
Step 3: Determine the bond i_c for which

$$\frac{1}{\lambda} = \max_i \left(\frac{f_i}{f_i^{th}} \right)$$

Step 4: Record the lattice displacement and force $(\mathbf{x}, \mathbf{Kx})$

Step 5: Remove the bond i_c and repeat steps 1-4, until the entire lattice system breaks apart

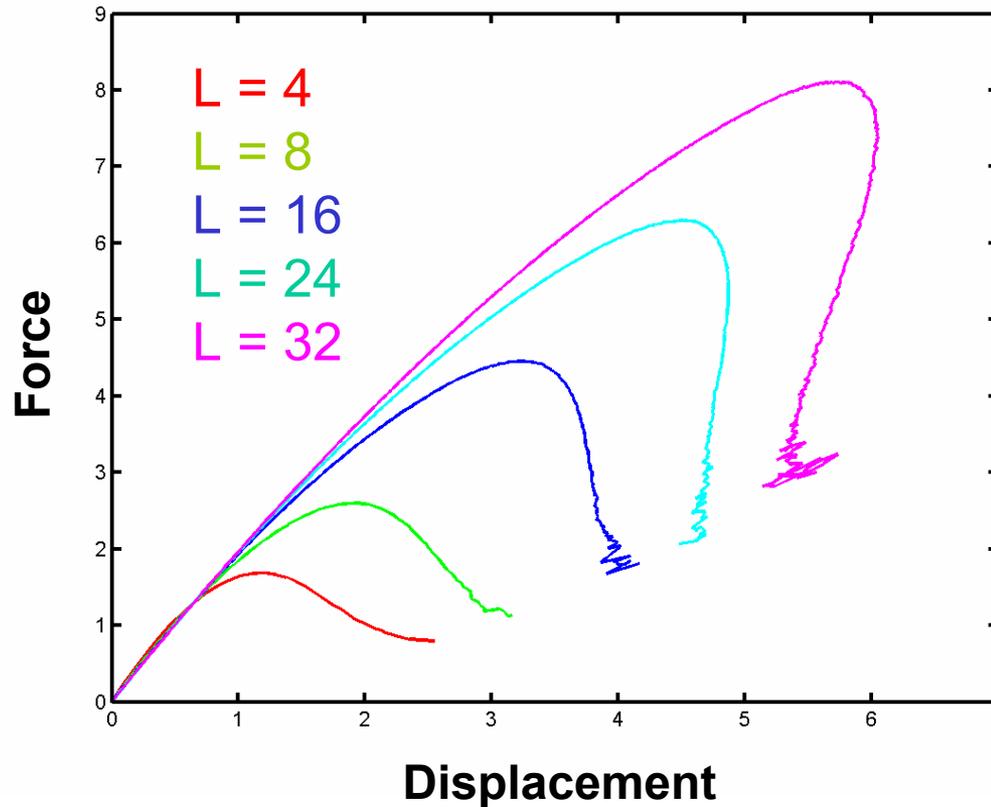
Typical Loading Response



Single lattice response
Lattice response averaged
over 5000 samples

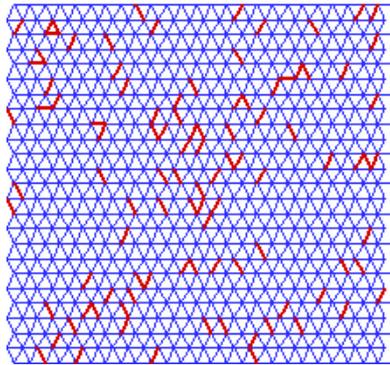
In the **hardening regime**, average material response is obtained with **fewer number of samples**, whereas in the **softening regime**, averaging over **many number of samples** is required to obtain a representative material response

Lattice Response versus System Size

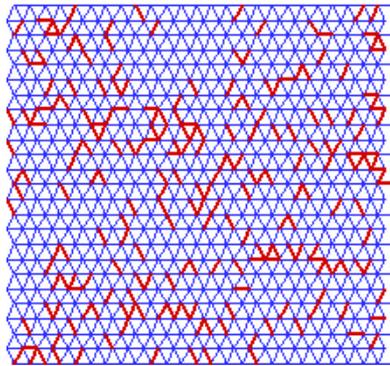
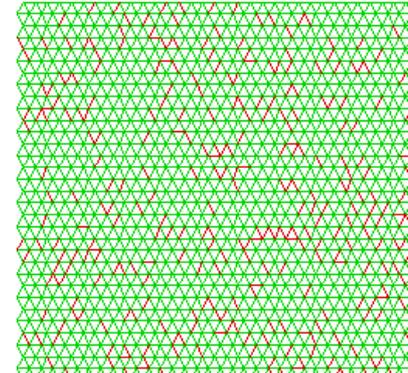


- Lattice response depends on the **system size**
- **Scaling laws** are required to obtain a “normalized” response that couples the mesoscopic scale response to the continuum scale response

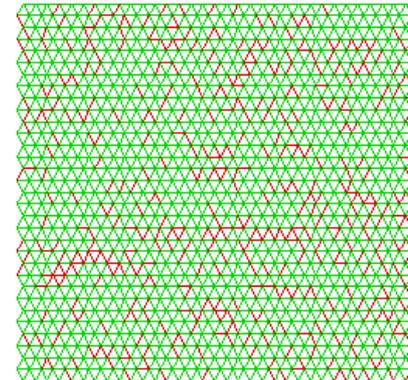
Example 1: Damage Evolution and Fracture in Tension



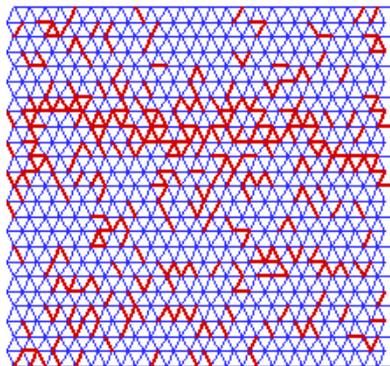
**Nucleation Phase:
Diffusive Damage**



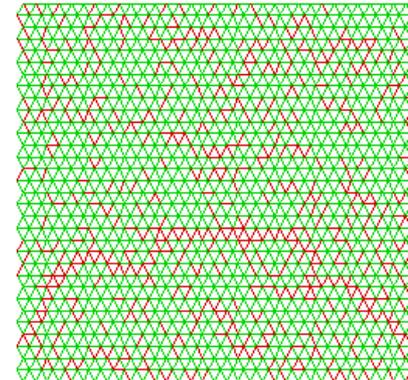
**Growth Phase:
Stress Concentration
effects are dominant**



$L = 24$



**Coalescence:
Localization of damage to
form a percolating crack**

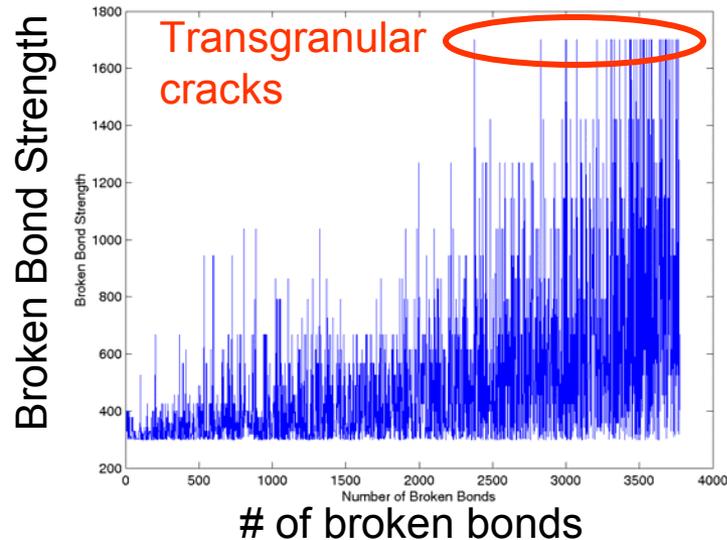


$L = 32$

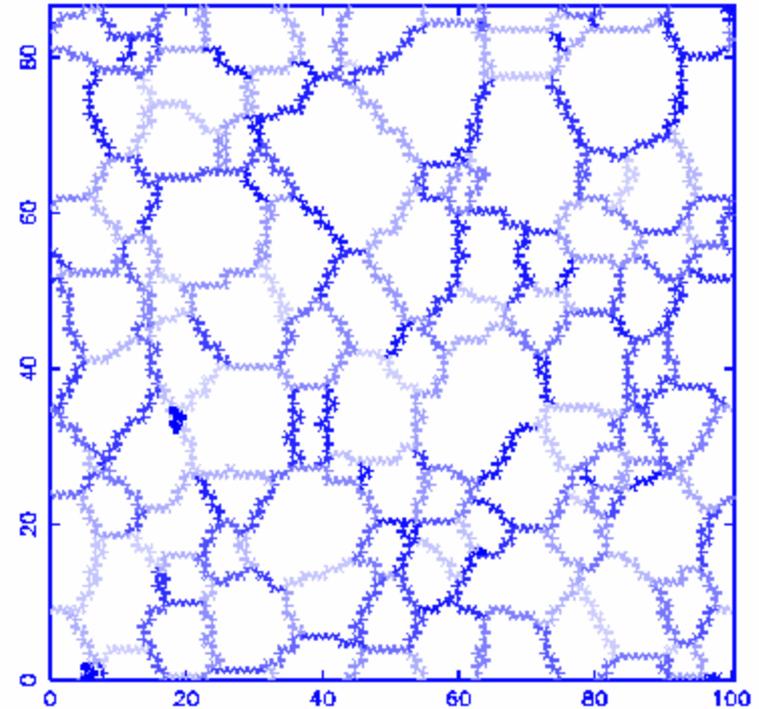
Example 2: Temperature Induced Microcracking

Investigation of the effect of microstructural features on microcrack evolution in brittle materials due to thermal expansion anisotropy

Movie



Microstructure 10 links broken



L = 100

Simulation Data:

Molybdenum:

$E = 324.054 \text{ GPa}$

$\alpha = 4.8\text{E-}06 \text{ /K}$

$\nu = 0.31$

Misorientation dependent

low angle grain boundary

fracture strength for Mo

(Watanabe et. al)

Developed Scaling Laws

$$p_{fL} - p^* = cL^{-\alpha_\rho}$$

$$\Delta_{pfL} = c_\Delta L^{-\alpha_\rho}$$

$$(p_{fL} - p^*) = c_p \Delta_{pfL}$$

p_{fL} and p^* denote the fracture thresholds in a lattice system size of L and infinity, and $c_p = \frac{c}{c_\Delta}$.

Discussion on Scaling Laws

- Developed scaling laws imply existence of finite critical fracture threshold, below which, fracture of infinite system does not occur
 - Earlier results based on power laws imply that critical threshold approaches zero in an infinite system
- Existence of finite threshold may be associated with a critical crack size
- Numerical results substantiate the proposed scaling laws

Numerical Aspects

Computational Analysis

1. External loads are gradually increased until a bond threshold is reached
2. The bond is broken
3. Forces redistribute within the system instantaneously
4. Process of breaking bonds irreversibly one after another is repeated until the lattice network finally becomes disconnected

Computational Requirements

- New set of governing linear equations have to be solved every time a lattice bond is broken
- To obtain statistically significant results, large number of lattices, threshold distributions and loadings have to be analyzed
 - Different bond threshold distributions for essentially the same lattice result in different breaking sequences
 - Different loading conditions change breaking sequences as well
 - Original lattice and its stiffness matrix is the same

Previous Approaches

- Reduce scope of analysis
 - Assume percolation character of damage before breakdown
 - Use diluted network and look for a breaking of the first bond
 - Does not evolve the damage in natural way
- Jacobi iteration method
 - Slow
- PCG
 - Critical slowing down close to lattice breakdown
 - Fourier accelerated solvers show best performance
 - Do not work well on central force lattices, bond bending, etc.
- Small systems

Characteristics of Bond Breaking

- A faster algorithm can be developed if we recognize what breaking of a single bond does to the system of equations

$$\mathbf{A}_n \mathbf{x}_n = \mathbf{b}_n$$

- Bond breaking (removal) is equivalent to rank-one update of a matrix

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

$$\mathbf{v}^t = \{ 0 \quad \dots \quad 1 \quad \dots \quad -1 \quad \dots \quad 0 \}$$

System Update

- Assuming inverse of \mathbf{A}_n is available, the inverse of \mathbf{A}_{n+1} can be obtained by updating inverse of \mathbf{A}_n using Sherman-Morrison-Woodbury formula

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

$$\mathbf{u} = \mathbf{A}_n^{-1} \mathbf{v} = \mathbf{A}_n^{-1} (i-j) = [(i^{th} - j^{th}) \text{ columns of } \mathbf{A}_n^{-1}]$$

- Inverse is not usually explicitly calculated, and \mathbf{u}_n is obtained using \mathbf{A}_n through a back solve operation on vector \mathbf{v} .

Broken Bond Location

- Bond internal to the lattice

$$\mathbf{v}^t = \{ 0 \quad \dots \quad 1 \quad \dots \quad -1 \quad \dots \quad 0 \}$$

- Bond attached to a prescribed DOF

$$\mathbf{v}^t = \{ 0 \quad \dots \quad 1 \quad \dots \quad 0 \}$$

$$\mathbf{u} = \mathbf{A}_n^{-1}{}_{(i)} = [i^{th} \text{ columns of } \mathbf{A}_n^{-1}]$$

- Updating the load vector

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

$$\mathbf{w}^t = k_{ij} \{ 0 \quad 0 \quad \dots \quad -1 \quad \dots \quad 0 \quad 0 \}$$

Solution Update after Bond Break

- Before the break of $n+1$ bond

$$\mathbf{A}_n \mathbf{x}_n = \mathbf{b}_n$$

- After the break

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

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

Solution Update after Bond Break

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

Solution Update Algorithm

- Only unknown is vector \mathbf{u} which can be obtained through back solve operation
- It is not necessary to explicitly assemble \mathbf{A}_{n+1} and perform factorization to do back solve operation
- We can use already factorized \mathbf{A}_m ($m < n$) to obtain vector \mathbf{u}
 - \mathbf{m} is the latest load increment (bond break index) for which factorize A is available

Solution Update Algorithm

- Decompose

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

where

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

- \mathbf{C} is never explicitly calculated or stored
 - \mathbf{u} vectors are stored instead and used as

$$\mathbf{C}_{(j-i)} = \sum_{l=1}^{p=(n-m)} k_l \frac{(\mathbf{u}_{lj} - \mathbf{u}_{li})}{(1 - k_l \mathbf{v}_l^t \mathbf{u}_l)} \mathbf{u}_l$$

Obtaining New Factorizations

- $\mathbf{C}_{(i-j)}$ reduces storage and computational requirements each to $\sim O(p n_{DOF})$
- Even with this reduction calculation of $\mathbf{C}_{(i-j)}$ may become prohibitive as p increases
 - Factorize or update \mathbf{A}
- Options:
 - Factorize again after $p = \text{maxupd}$
 - Update factorization using multiple-rank sparse Cholesky factorization update (Davis et al.)
 - Update using series of rank-one updates
 - Update factorization using modified dense Cholesky factorization update (Davis et al.)

Circulant Preconditioners for PCG

- Proposed method is compared against PCG method
- Circulant preconditioners can be diagonalized using discrete Fourier matrices
- It is possible to choose a circulant preconditioner that minimizes the condition number of the preconditioned system
- Exhibit favorable clustering of eigenvalues
- Circulant preconditioners are better than ensemble-averaged circulant preconditioner used in the past for similar analyses
- Block-circulant preconditioners were also used for comparison

Solvers Based on the Proposed Method

- *Solver Type 1:* Factorization of the matrix \mathbf{A} is performed every *maxupd* steps. Assuming that the factorization \mathbf{L}_m of the stiffness matrix \mathbf{A}_m is available after burning the m^{th} fuse, determine the solution vector \mathbf{x}_{n+1} after the $(n+1)^{\text{th}}$ fuse is burnt.
- *Solver Type 2:* Factorization of the matrix \mathbf{A} is performed every *maxupd* steps. Assuming that the factorization \mathbf{L}_m of \mathbf{A}_m is available, the dense matrix update is used to obtain the solution vector \mathbf{x}_{n+1} after the $(n+1)^{\text{th}}$ fuse is burnt.
- *Solver Type 3:* Given the factorization \mathbf{L}_m of \mathbf{A}_m , rank-1 sparse Cholesky update/downdate is used to update the factorization \mathbf{L}_{n+1} for all subsequent values of $n = m, m+1, \dots$. 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.
- *Solver Type 4:* Given the factorization \mathbf{L}_m of \mathbf{A}_m , determine the solution vector \mathbf{x}_{n+1} after the $(n+1)^{\text{th}}$ fuse is burnt. The difference between *Types* 1 and 4 is that instead of refactorizing the matrix after *maxupd* steps, we use rank-p sparse Cholesky update/downdate (algorithm 1) to obtain the factorization $\mathbf{L}_{m+\text{maxupd}}$ of the matrix $\mathbf{A}_{m+\text{maxupd}}$.

Performance of Developed Solvers

1

Size	CPU(sec)	Wall(sec)	Simulations
32	0.566	0.655	20000
64	9.641	10.59	4000
128	203.1	213.4	800
256	6121	6139	96

3

Size	CPU(sec)	Wall(sec)	Simulations
32	0.592	0.687	20000
64	10.72	11.26	4000
128	212.2	214.9	800
256	5647	5662	96
512	93779	96515	16

2

Size	CPU(sec)	Wall(sec)	Simulations
32	0.679	0.771	20000
64	11.18	12.28	4000
128	254.4	260.2	800
256	6112	6147	96

4

Size	CPU(sec)	Wall(sec)	Simulations
32	0.543	0.633	20000
64	11.15	12.01	4000
128	211.5	214.1	800
256	6413	6701	96

PCG Performance

CG Iterative Solver (No preconditioner)

Size	CPU(sec)	Wall(sec)	Iterations	Simulations
32	7.667	8.016	66254	20000
64	203.5	205.7	405510	1600

CG Iterative Solver (Incomplete Cholesky)

Size	CPU(sec)	Wall(sec)	Iterations	Simulations
32	2.831	3.008	5857	20000
64	62.15	65.61	29496	4000
128	1391	1430	148170	320

T. Chan's optimal circulant PCG

Size	CPU(sec)	Wall(sec)	Iterations	Simulations
32	11.66	12.26	25469	20000
64	173.6	178.8	120570	1600
128	7473	7725	622140	128

T. Chan's block-circulant PCG

Size	CPU(sec)	Wall(sec)	Iterations	Simulations
32	10.00	10.68	11597	20000
64	135.9	139.8	41207	1600
128	2818	2846	147510	192
256	94717	96500		32

Conclusions

- For material damage and fracture analyses, the proposed algorithms have shown better performance than available PCG solvers
- Algorithms take advantage of the bond breaking process to speed up the *overall* computations
 - Completely eliminates critical slowing down near lattice breakdown
- For very large problems, matrix factorizations can be done using parallel processing and then factored matrix can be distributed to each processor to continue independent fracture simulations

Disclaimer

The submitted manuscript has been authored by a contractor of the U. S. Government under Contract No. DE-AC05-00OR22725. Accordingly, the U. S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U. S. Government purposes.

This research is sponsored by the Mathematical, Information and Computational Sciences Division, Office of Advanced Scientific Computing Research, U. S. Department of Energy under contract number DE-AC05-00OR22725 with UT-Battelle, LLC.