# Low-rank Updates in Statistical Physics Applications

**Phani Kumar Nukala**
**Oak Ridge National Laboratory**

**JSTAT P03029 (2010)**
**Phys. Rev. B 80, 195111 (2009)**
**J. Chem. Phys. 130(20), 204105 (2009)**
**JSTAT P08007 (2007)**
**J. Phys. A 37, 2093 (2004)**
**J. Phys. A 36, 11403 (2003)**

# Motivation

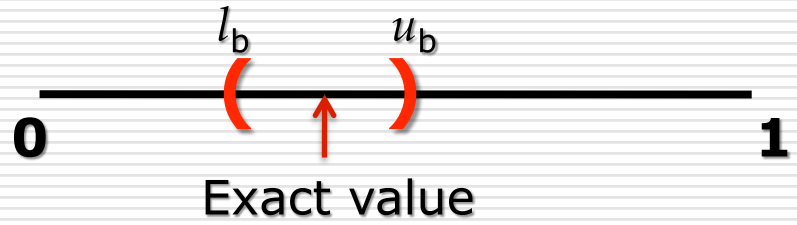
- In a typical MC simulation, computing the transition probability is the most computationally intensive step.

- In many MC models, transition probability is dependent on
  - Computing eigenvalues of Hamiltonian matrix
  - Determinant of Green's function matrix
  - or some basic linear algebra problem

- Naïve repetitive computation is a challenge even for modern supercomputers since a large number of MC steps are required to solve the problem.
  - Linear algebra sub-problem does not scale well
  - Computational complexity increases as $O(N^3)$
- **An important feature of these problems is that successive matrices differ by a low-rank perturbation**

# Motivation

- Since these matrices differ by a low-rank,
  - **Can we devise algorithms that use information from previous step to EXACTLY compute the transition probability of current step?**
  - **Can we find tight bounds for this probability so that we can often avoid exact computation of transition probability?**




Speedup = $1/(u_b - l_b) - 1$

# Problem Definition

- Given $\det\left(f(\mathbf{A}_k)\right)$, find $R_k = \dfrac{\det\left(f(\mathbf{A}_{k+1})\right)}{\det\left(f(\mathbf{A}_k)\right)} \quad \forall k$

where $\mathbf{A_{k+1}} = \mathbf{A_k} + \alpha_k \mathbf{u}_k \mathbf{v}_k^{\mathbf{T}}$

- Model problems
  - Spin-fermion systems $f(\mathbf{A_k}) = \mathbf{I} + e^{\beta \mathbf{A}_k}$
  - Strongly correlated systems $f(\mathbf{A_k}) = \mathbf{A}_k$
  - Quantum Monte Carlo (QMC) $f(\mathbf{A_k}) = \mathbf{A}_k$
- Related Problems
  - Discrete fracture (sparse & iterative) $\mathbf{A_k}\,\mathbf{x}_k = \mathbf{b}_k$

# Outline

- High-temperature superconductivity model (HTSC)
  - Delayed update for Hubbard Model
  - Improved sub-matrix algorithm
- Colossal Magneto-Resistance model (CMR)
  - Low-rank update algorithm for Spin-Fermion Model
  - Bounds for transition probabilities
- Statistical physics of fracture model
  - Recycling Krylov CG
  - Sparse direct solvers

# Application: Hubbard Model for High-Temperature Superconductivity

**Phys. Rev. B 80, 195111 (2009)**
**J. Chem. Phys. 130(20), 204105 (2009)**

# Hubbard Model

$$H = -t \sum_{\langle i,j \rangle} c^{\#}_{i\sigma} c_{j\sigma} + U \sum_{i} n_{i\uparrow} n_{i\downarrow}$$

- Transform interaction term with auxiliary HHS spin fields
- These spin fields are integrated using MC
- Local MC move is accepted based on

$$r' = \frac{\rho(\psi')}{\rho(\psi)} = \frac{\det \mathbf{G}_{\uparrow}^{-1}(\psi') \det \mathbf{G}_{\downarrow}^{-1}(\psi')}{\det \mathbf{G}_{\uparrow}^{-1}(\psi) \det \mathbf{G}_{\downarrow}^{-1}(\psi)} = R_{\uparrow} R_{\downarrow},$$

where $$R_{\sigma} = \frac{\det \mathbf{G}_{\sigma}^{-1}(\psi')}{\det \mathbf{G}_{\sigma}^{-1}(\psi)}.$$

# Hubbard Model

- Repetitive computation of determinant when the Green's function matrix undergoes rank-1 updating

- Given $\det(\mathbf{G}_k)$ , find $R_k = \dfrac{\det(\mathbf{G}_{k+1})}{\det(\mathbf{G}_k)}$

where $\mathbf{G_{k+1}} = \mathbf{G_k} + \alpha_k \mathbf{u}_k \mathbf{v}_k^{\mathbf{T}}$

$\mathbf{u}_k = \alpha_k \left[ \mathbf{G}_k(:, p) - \mathbf{e}_p \right]$

$\mathbf{v}_k = \mathbf{G}_k(p, :)$

$$\alpha_k = \frac{\gamma_k}{\left[1 + \gamma_k \left(1 - \mathbf{G}_k(p,p)\right)\right]}$$

$$R_k = \frac{1}{\left[1 + \gamma_k \left(1 - \mathbf{G}_k(p,p)\right)\right]}$$

$O(N^2)$

Updating G over each step poses a significant computational problem

# Algorithm 1: Delayed Update

- Given $G_0$, set $d_0 = \mathrm{diag}(G_0)$
- Initialize $\mathbf{u}_k = \mathbf{G}_0(:, p) \quad and \quad \mathbf{v}_k = \mathbf{G}_0(p, :)$
- Compute

$$R_k = \frac{1}{\left[1 + \gamma_k \left(1 - \mathbf{d}_k(p)\right)\right]}$$

- Update

$$for\ i = 0 : k-1,$$

$$\mathbf{u}_k = \mathbf{u}_k + \mathbf{v}_i(p)\,\mathbf{u}_i$$

$$\mathbf{v}_k = \mathbf{v}_k + \mathbf{u}_i(p)\,\mathbf{v}_i$$

$$end$$

$O(kN)$

- Update diagonal

$$\mathbf{u}_k = \alpha_k \left(\mathbf{u}_k - \mathbf{e}_p\right)$$

$$\mathbf{d}_k = \mathbf{d}_k + \mathbf{u}_k \circ \mathbf{v}_k$$

- **1.35PFlops/s on Cray Jaguar at ORNL.**
- **2008 Gordon Bell Award**

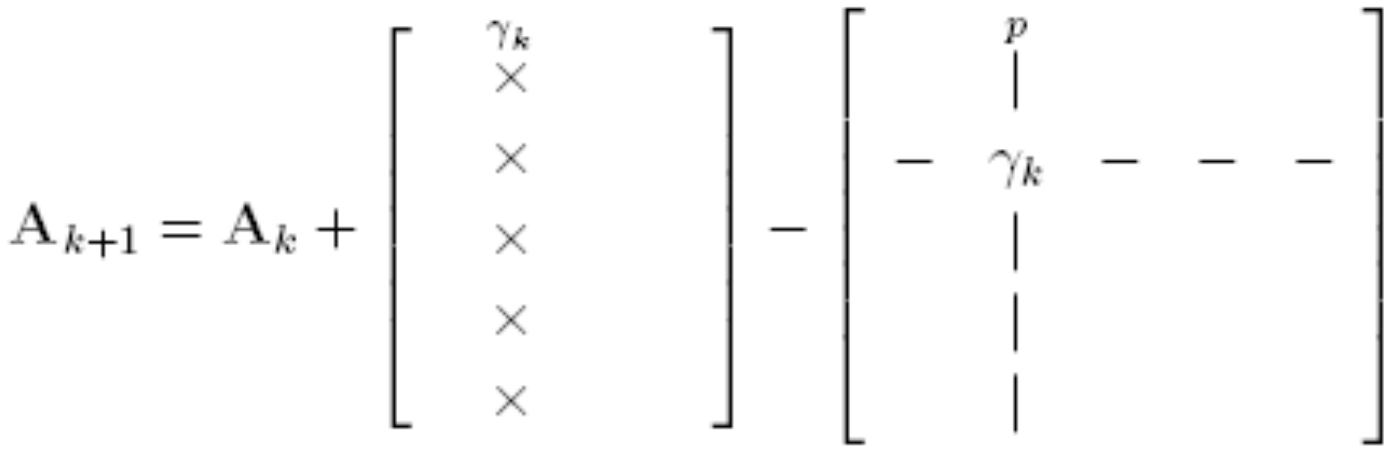
# Algorithm 2: Improved Method

- Instead of updating G, start working with $A = G^{-1}$

$$\mathbf{G_{k+1}} = \mathbf{G_k} + \alpha_k \mathbf{u}_k \mathbf{v}_k^{\mathbf{T}} \quad \text{with} \quad \mathbf{u}_k = \alpha_k \left[ \mathbf{G}_k(:,p) - \mathbf{e}_p \right]$$

$$\mathbf{v}_k = \mathbf{G}_k(p,:)$$

$$\mathbf{A_{k+1}} = \mathbf{A_k} + \gamma_k \left[ \mathbf{A}_k(:,p) - \mathbf{e}_p \right] \otimes \mathbf{e}_p$$



- For k+1 steps,

$$\mathbf{A_{k+1}} = \mathbf{A}_0 + \sum_{j=0}^{k} \gamma_j \left[ \mathbf{A}_0(:, \mathbf{p}(j)) - \mathbf{e}_{\mathbf{p}(j)} \right] \otimes \mathbf{e}_{\mathbf{p}(j)}$$

$$= \tilde{\mathbf{A}}_k - \sum_{j=0}^{k} \gamma_j \, \mathbf{e}_{\mathbf{p}(j)} \otimes \mathbf{e}_{\mathbf{p}(j)} = \tilde{\mathbf{A}}_k - \mathbf{U}\mathbf{V}^T$$

$$\mathbf{A}_{k+1} = \mathbf{A}_0 + \begin{bmatrix} \gamma_0 & & \gamma_k \\ \times & & \times \\ \times & & \times \\ \times & \cdots & \times \\ \times & & \times \\ \times & & \times \end{bmatrix} - \begin{bmatrix} & \mathbf{p}(0) & & \mathbf{p}(k) & \\ & | & & | & \\ - & \gamma_0 & - & - & - \\ & | & & | & \\ - & - & - & \gamma_k & \\ & | & & | & \end{bmatrix}$$

$$\det\left(\mathbf{A}_{k+1}\right) = \det\left(\tilde{\mathbf{A}}_k\right) \det\left(\mathbf{I} - \mathbf{V}^T \tilde{\mathbf{G}}_k \mathbf{U}\right)$$

$$\tilde{\mathbf{A}}_{k+1} = \mathbf{A}_0 + \begin{bmatrix} \gamma_0 & & \gamma_k \\ \times & & \times \\ \times & & \times \\ \times & \cdots & \times \\ \times & & \times \\ \times & & \times \end{bmatrix} \Rightarrow \det\left(\tilde{\mathbf{A}}_k\right) = \left[\prod_{j=0}^{k} \left(1 + \gamma_k\right)\right] \det(\mathbf{A}_0)$$

$$\det\left(\mathbf{I}-\mathbf{V}^T\tilde{\mathbf{G}}_k\mathbf{U}\right)=(-1)^{k+1}\left[\prod_{j=0}^{k}\frac{\gamma_j}{1+\gamma_j}\right]\det\left(\Gamma_k\right)$$

where

$$\Gamma_k=\mathbf{G}_0(\mathbf{p})-\begin{bmatrix}(1+\gamma_0)/\gamma_0 & & \\ & \ddots & \\ & & (1+\gamma_k)/\gamma_k\end{bmatrix}$$

$$\mathbf{G}_0(\mathbf{p})=\begin{bmatrix}\mathbf{G}_0(\mathbf{p}(0),\mathbf{p}(0)) & \cdots & \mathbf{G}_0(\mathbf{p}(0),\mathbf{p}(k))\\ \vdots & \ddots & \vdots \\ \mathbf{G}_0(\mathbf{p}(k),\mathbf{p}(0)) & \cdots & \mathbf{G}_0(\mathbf{p}(k),\mathbf{p}(k))\end{bmatrix}$$

- Symbolically,

$$\mathbf{\Gamma}_k = \begin{bmatrix} \mathbf{\Gamma}_{k-1} & \mathbf{s} \\ \mathbf{w}^T & d \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{k-1} & 0 \\ \mathbf{x}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{U}_{k-1} & \mathbf{y} \\ 0 & \beta \end{bmatrix}$$

where

$$\mathbf{s} = \mathbf{G}_0\left(\mathbf{p}(0) : \mathbf{p}(k-1), \mathbf{p}(k)\right)$$

$$\mathbf{w}^T = \mathbf{G}_0\left(\mathbf{p}(k), \mathbf{p}(0) : \mathbf{p}(k-1)\right)$$

$$d = \mathbf{G}_0\left(\mathbf{p}(k), \mathbf{p}(k)\right) - \frac{1+\gamma_k}{\gamma_k}$$

$$\det(\mathbf{\Gamma}_k) = \beta \det(\mathbf{\Gamma}_{k-1})$$

where $\beta = d - \mathbf{x}^{\mathbf{T}}\mathbf{y}$

$\mathbf{L}_{k-1}\mathbf{y} = \mathbf{s}$

$\mathbf{U}_{k-1}^{T}\mathbf{x} = \mathbf{w}$

$O(k^2)$

# Summarizing … $O(k^2)$ algorithm

$$\det\left(\mathbf{A}_{k+1}\right) = \det\left(\tilde{\mathbf{A}}_k\right) \det\left(\mathbf{I} - \mathbf{V}^T \tilde{\mathbf{G}}_k \mathbf{U}\right)$$

$$\det\left(\tilde{\mathbf{A}}_k\right) = \left[\prod_{j=0}^{k} \left(1 + \gamma_k\right)\right] \det(\mathbf{A}_0)$$

$$\det\left(\mathbf{I} - \mathbf{V}^T \tilde{\mathbf{G}}_k \mathbf{U}\right) = (-1)^{k+1} \left[\prod_{j=0}^{k} \frac{\gamma_j}{1+\gamma_j}\right] \det\left(\Gamma_k\right)$$

$$\det\left(\Gamma_k\right) = \beta \det\left(\Gamma_{k-1}\right)$$

$$\det\left(\mathbf{A}_{k+1}\right) = -\beta\, \gamma_k \det\left(\mathbf{A}_k\right) \quad \rightarrow \quad R_k = -\frac{1}{\beta\, \gamma_k}$$

# Numerical Results: Hubbard Model

- Given $\det(\mathbf{G}_k)$ , find $R_k = \dfrac{\det(\mathbf{G}_{k+1})}{\det(\mathbf{G}_k)}$, such that

$$\mathbf{G_{k+1}} = \mathbf{G_k} + \alpha_k \mathbf{u}_k \mathbf{v}_k^{\mathbf{T}}$$

$$\mathbf{u}_k = \alpha_k \left[ \mathbf{G}_k(:, p) - \mathbf{e}_p \right]$$

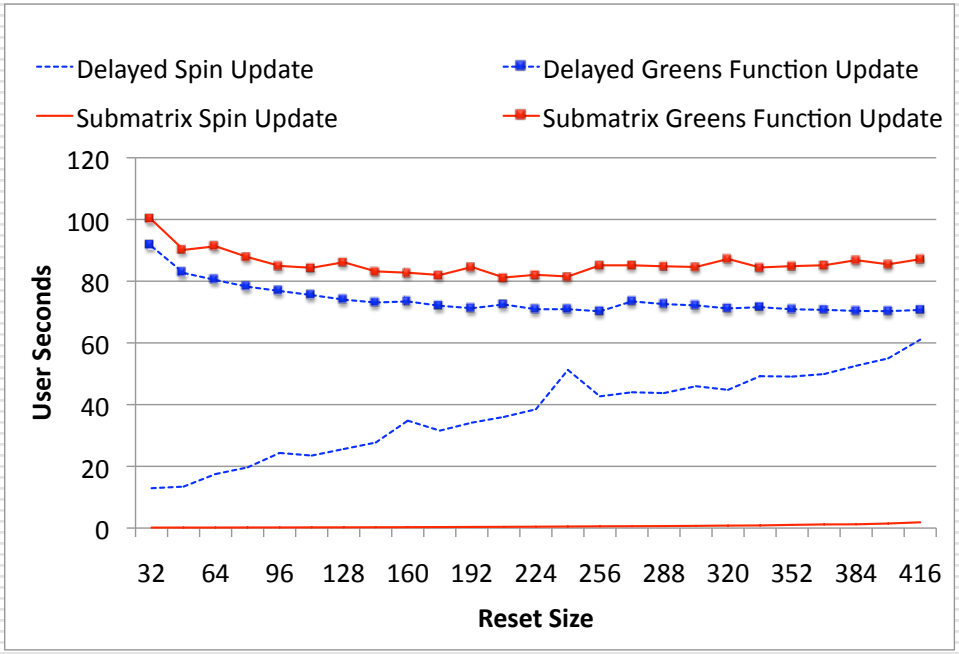
$$\mathbf{v}_k = \mathbf{G}_k(p, :)$$

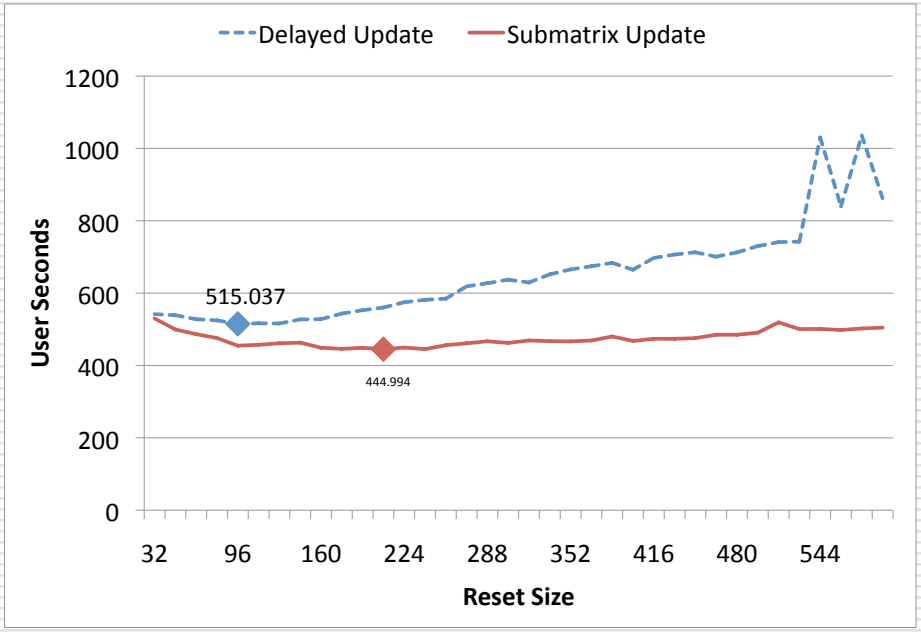
- Initialized with a random matrix
- Updates, m = N/10

| N | Full | Delayed | Recursive | New |
|---|---|---|---|---|
| 1000 | 7.02 | 0.28 | 0.09 | 0.015 |
| 3000 | 195 | 9.86 | 2.31 | 0.49 |

# Numerical Results: Hubbard Model

- 16 site dynamic cluster QMC
- 2D Hubbard model
- Hopping t, and U = 4t; inverse temperature = 40/t



# Application: Colossal Magneto-resistance and Spin-Fermion Systems

JSTAT P08007 (2007)

# Spin-Fermion Systems: Double Exchange Model

$$H(\psi) = c^{\#} A(\psi) c$$

$$\begin{aligned} \mathrm{A}^{\alpha}_{i+\sigma M, j+\sigma' M} &= t_{ij}\delta_{\sigma,\sigma'} + (1-2\sigma) J_{\mathrm{H}} \cos(\theta_i^{\alpha}) \delta_{i,j}\delta_{\sigma,\sigma'} \\ &+ J_{\mathrm{H}} \exp(\mathrm{i}\phi_i^{\alpha}(1-2\sigma)) \sin(\theta_i^{\alpha}) \delta_{i,j}\delta_{\sigma,1-\sigma'}. \end{aligned}$$

- Boltzmann weight P(ψ) for field configuration ψ is given by

$$\begin{aligned} P(\psi) &= \exp(-S_{\mathrm{eff}}(\psi))/Z \\ Z &= \sum_{\psi} \exp(-S_{\mathrm{eff}}(\psi)) \\ S_{\mathrm{eff}} &= \sum_{\nu} F(\lambda_\nu(\psi)) \\ F(\lambda) &= -\log(1+\exp(-\beta(\lambda-\mu))) \end{aligned}$$

# Spin-Fermion Systems

- Simulation of colossal magnetoresistance using spin-fermion systems poses the following problem:

- Given $\det\left(\mathbf{I}+e^{\beta\mathbf{A}_k}\right)$, find $\det\left(\mathbf{I}+e^{\beta\mathbf{A}_{k+1}}\right)$

  where $\mathbf{A}_{k+1}=\mathbf{A}_k+\rho\mathbf{u}\mathbf{u}^{\mathrm{T}}$ for all k = 0,1,2…

  - Updating all the eigenvalues of $A_{k+1}$ based on the eigenvalues of $A_k$

# Spin-fermion Systems

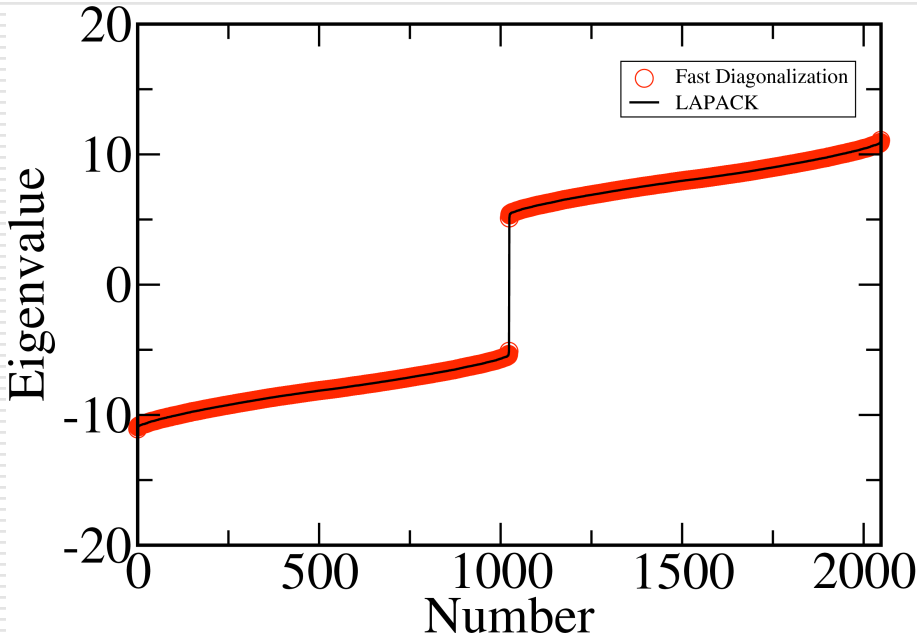
Comparison of CPU times taken for 10 steps

| Size (N) | LAPACK (sec) | Low-rank update (sec) |
|---|---|---|
| 288 | 0.6 | 0.34 |
| 800 | 16.1 | 2.5 |
| 1152 | 60.1 | 9.7 |
| 2048 | 298.5 | 32.1 |

$O(N^2)$

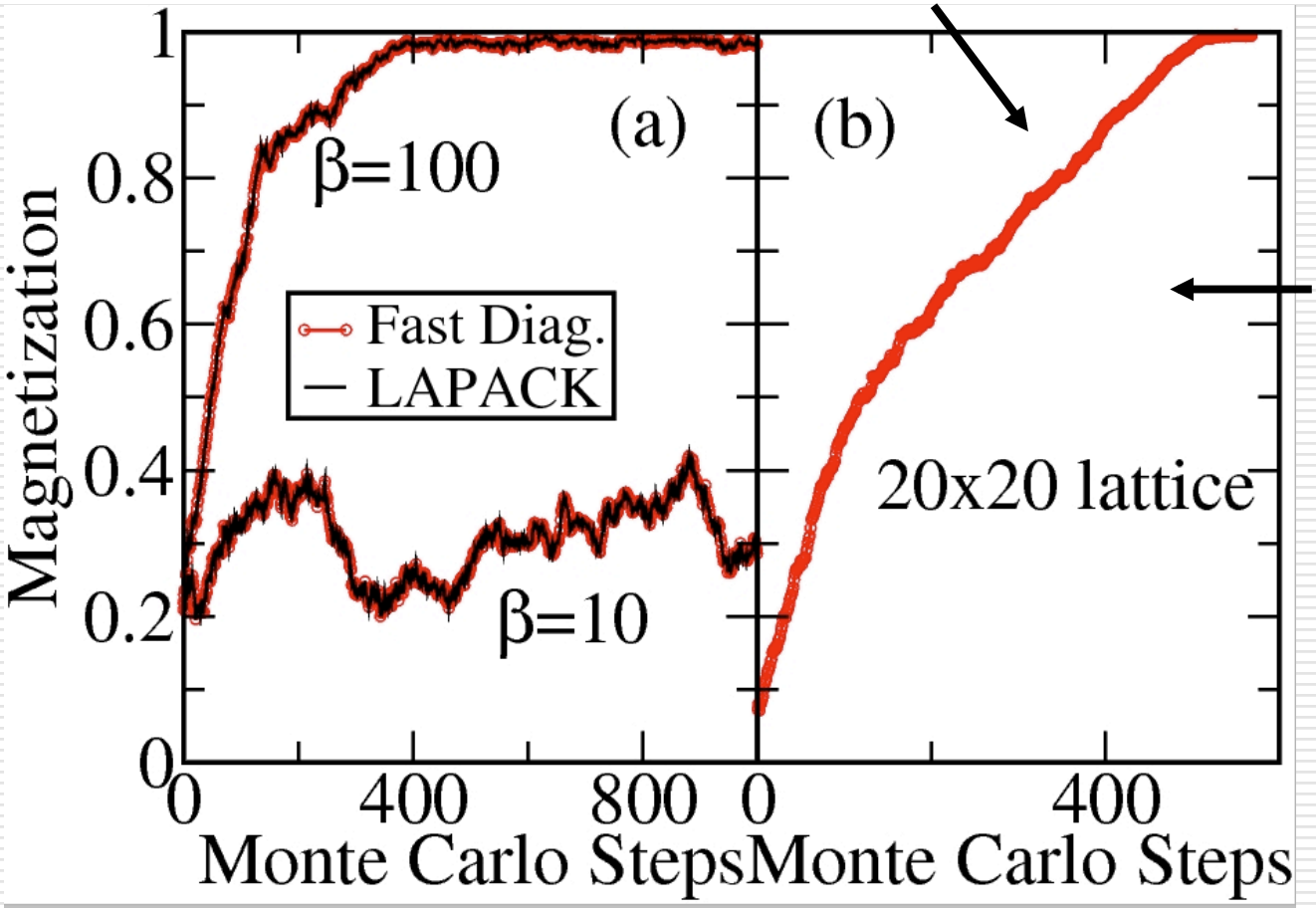
$O(N^3)$

- Fast Multipole Method further reduces the computational complexity to O(N log N)
- Excellent accuracy of eigen spectrum even after many updates



# Large System Simulation

- Simulation not readily accessible to traditional method of direct diagonalization (DDM) during each step




DDM ~ 5 days
Low-rank ~ 1 day

32 x 32 Lattice:
DDM ~ 1 year
Low-rank ~ 15 days!

# Estimating Bounds for Transition Probabilities in QMC

- Given $\det\left(f(\mathbf{A}_k)\right)$, find bounds $C_l$ and $C_u$ such that

$$C_l \leq R_k = \frac{\det\left(f(\mathbf{A}_{k+1})\right)}{\det\left(f(\mathbf{A}_k)\right)} \leq C_u$$

where $\mathbf{A_{k+1}} = \mathbf{A_k} + \alpha_k \mathbf{u}_k \mathbf{v}_k^{\mathbf{T}}$

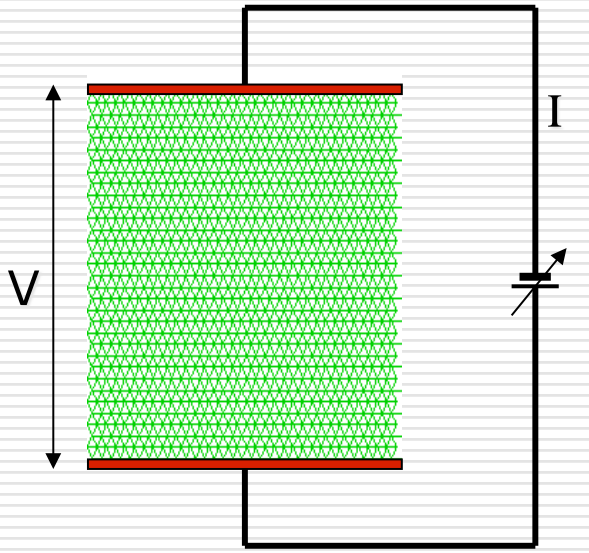
- Accurate bounds would significantly speedup Monte Carlo simulation

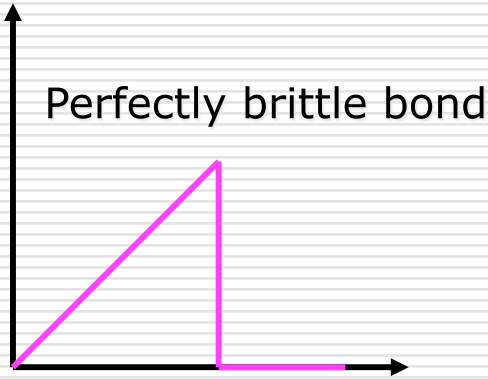
# Application: Statistical Physics of Fracture

JSTAT P03029 (2010)
J. Phys. A 37, 2093 (2004)
J. Phys. A 36, 11403 (2003)

# Random Thresholds Models

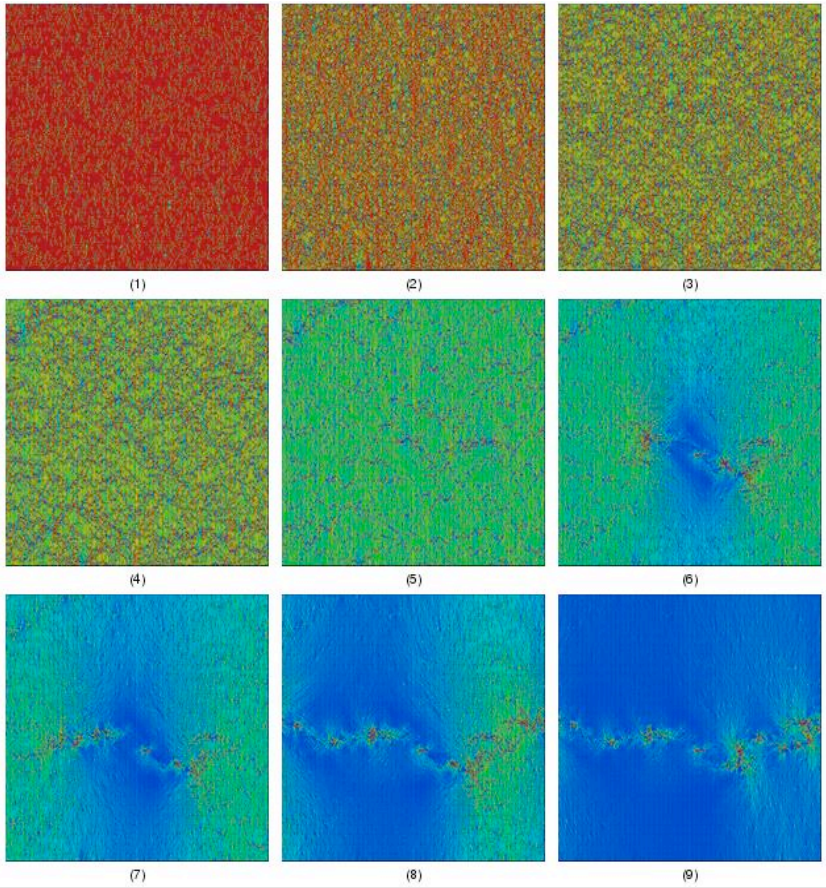
- For each bond, assign unit conductance and the thresholds are prescribed based on a random thresholds distribution
- The bond breaks irreversibly whenever the current (stress) in the fuse exceeds the prescribed thresholds value
- Currents (stresses) are redistributed instantaneously
- The process of breaking one bond at a time is repeated until the lattice falls apart



# Fracture of a 2D lattice system

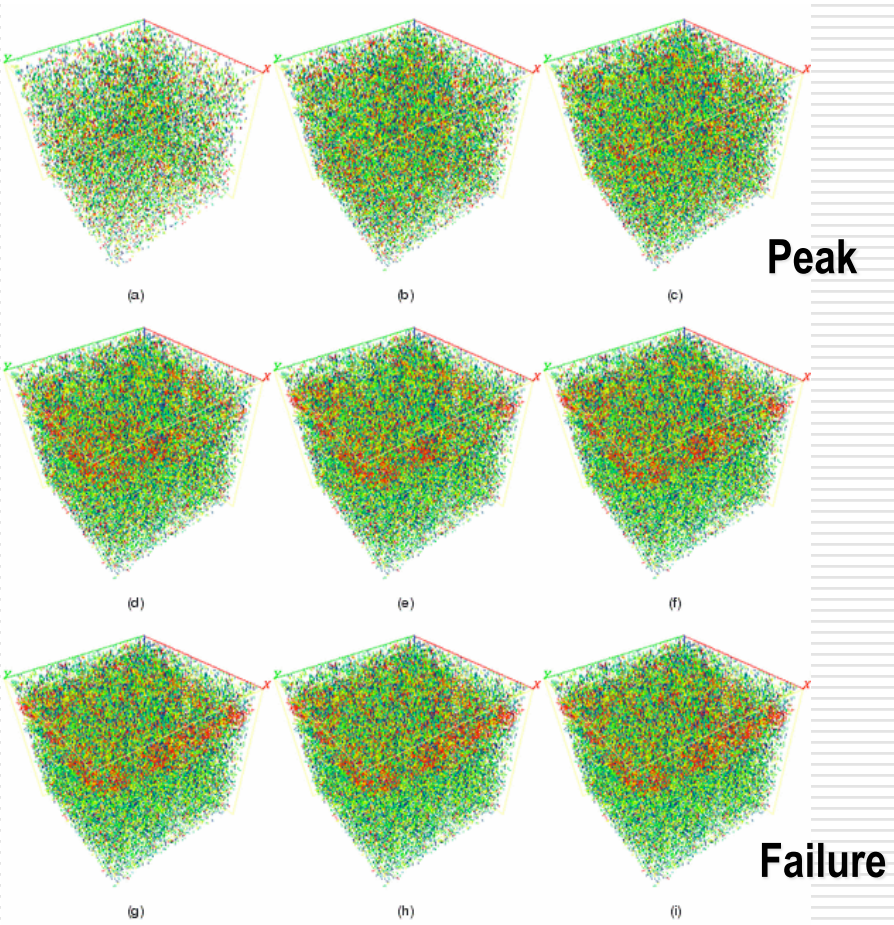
- CPU ~ $O(L^{4.5})$
- Capability issue: Previous simulations have been limited to a system size of L = 128
- Largest 2D lattice system (L = 1024) analyzed for investigating fracture and damage evolution
- Effective computational gain ~ 80 times




**Stress redistribution in the lattice due to progressive damage/crack propagation**

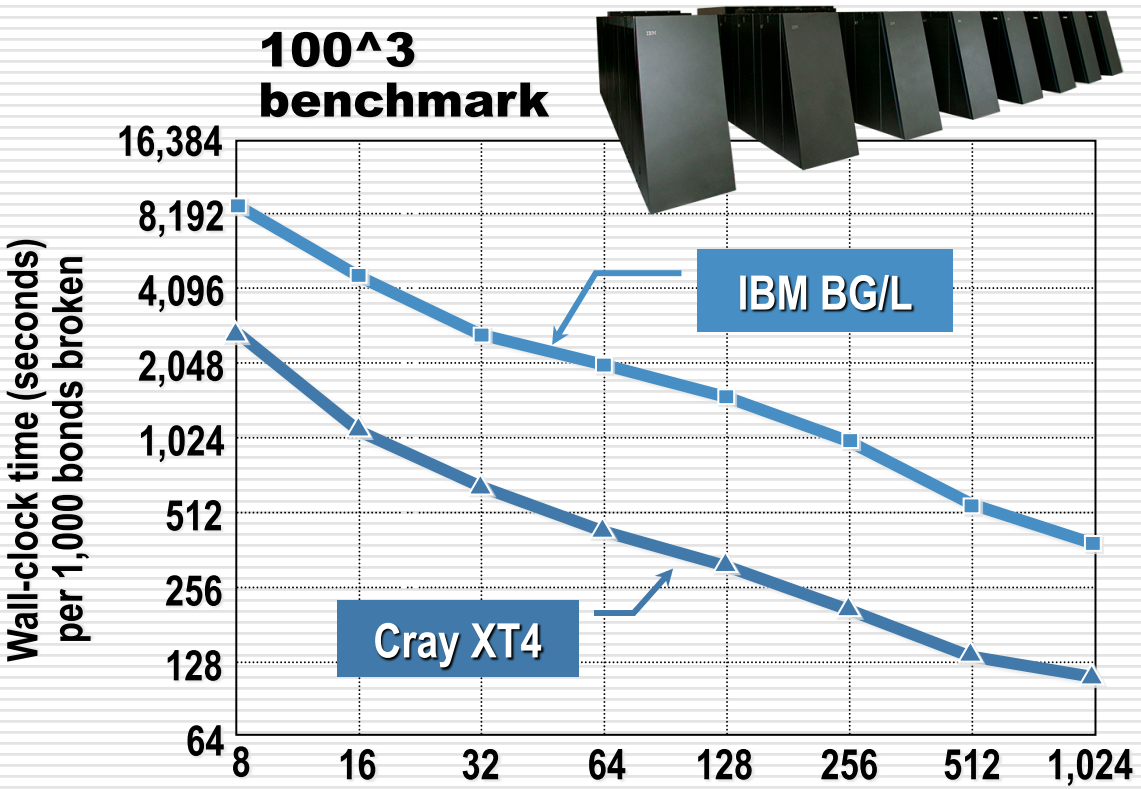
# Fracture of 3D lattice system

- CPU ~ $O(L^{6.5})$
- Largest cubic lattice system analyzed for investigating fracture and damage evolution in 3D (L = 64)
- On a single processor, a 3D system of size L = 64 requires 15 days of CPU time!



# High-performance computing

| High-performance computing | Processing time |
|---|---|
| L = 64 on 128 | 3 hours |
| L = 100 on 1024 | 12 hours |
| L = 128 on 1024 | 3 days |
| L = 200 on 2048 | 20 days (est.) |




# Recycling Conjugate Gradients for Fracture Modeling

- Recycling approximate invariant subspaces (lowest eigenvectors) between linear systems
- Projection step in CG to enforce all estimates are orthogonal to span($u_1$,$u_2$,…,$u_k$)
- Algorithm to generate new recycling space for next system by using conjugate directions to update estimates of harmonic Ritz vectors
- Development of model specific optimization in low rank update (A <- A + $\sigma$ vv')
- High cost for accurate computation of invariant subspace can be amortized over many solves

# Summary

- For Monte Carlo applications,
  - a low-rank updating scheme combined with bounds on transition probabilities can significantly speed up computation.
  - Typical speedups ~ 5-50 times

- Applications
  - Hubbard model for strongly correlated systems (QMC)
    - Low-rank schemes and bounds on transition probability
  - Spin-Fermion models
    - Repetitive eigen-decompositions and bounds on transition probability
  - Discrete Lattice models of fracture
    - Recycled CG
    - Sparse direct solvers

# Monte Carlo Simulation Scheme

- Estimate thermodynamic properties
- Start with an initial configuration (chosen at random)
- During each MC step,
  - Propose a local change
  - **Determine the probability to change**
    - **Either a Boltzmann weight or some other positive unit normalized measure**
  - Accept or reject change (Metropolis or heat-bath)
- Estimate time averages during measurement stage, and by ergodic theory, equate them to thermodynamics properties