An Efficient Solver for Sparse Linear Systems based on Rank-Structured Cholesky Factorization

David Bindel and Jeffrey Chadwick

Department of Computer Science
Cornell University

22 July 2013
u = K \ f

Great for circuit simulations, 1D or 2D finite elements, etc.

Standard advice to students: Just try backslash for these problems.
Standard response: What about for the 3D case?
“Try PCG with a good preconditioner. Maybe start with the ones in PETSc. You’ve taken Matrix Computations, right? Blah blah yadda blah...”

I have no idea what you’re talking about...

...so here’s a bunny with a pancake on its head.

(Not an actual student)
Direct or iterative?

CW: Gaussian elimination scales poorly. Iterate instead!

- **Pro:** Less memory, potentially better complexity
- **Con:** Less robust, potentially worse memory patterns

Commercial finite element codes still use (out-of-core) Cholesky. Longer compute times, but fewer tech support hours.
I want a code for sparse Cholesky \((A = LL^T)\) that

- Handles modest problems on a desktop (or laptop?)
  - Inside a loop, without trying my patience
  - \implies Does not need gobs of memory
  - \implies Makes effective use of level 3 BLAS
- Requires little parameter fiddling / hand-holding
- Works with general elliptic problems (esp. elasticity)

Idea is in the air – see talks by Darve and Li (bracketing me!), also work by Xia, Gu, Martinsson, Ying, ...
From ND to “superfast” ND

ND gets performance using just graph structure:

2D: $O(N^{3/2})$ time, $O(N \log N)$ space.

3D: $O(N^2)$ time, $O(N^{4/3})$ space.

Superfast ND reduces space/time complexity via low-rank structure.
Strategy

- Start with CHOLMOD (a good supernodal left-looking Cholesky)
  - Supernodal data structures are compact
  - Algorithm + data layout $\Rightarrow$ most work in level 3 BLAS
  - Widely used already (so re-use the API!)
- Incorporate compact representations for low-rank blocks
  - Outer product for off-diagonal blocks
  - HSS-style representations for diagonal blocks
- Optimize, test, swear, fix, repeat
Supernodal storage structure

\[ L_{ij} \equiv L_{ij}(C_j, R_j) \equiv L_{ij} \]

collapsed

(U. Waterloo, ND40)  Rank-Structured Cholesky  2013-07-22 8 / 27
Supernode factorization

\[
\begin{align*}
\mathbf{u}_j^D & \leftarrow A(\mathcal{C}_j, \mathcal{C}_j) \\
\mathbf{u}_j^O & \leftarrow A(\mathcal{R}_j, \mathcal{C}_j)
\end{align*}
\]

\textbf{for} each \( k \in \mathbb{D}_j \) \textbf{do}

- Build dense updates from \( \mathbf{L}_k^O \)
- Scatter updates to \( \mathbf{u}_j^D \) and \( \mathbf{u}_j^O \)

\[
\begin{align*}
\mathbf{L}_j^D & \leftarrow \text{cholesky}(\mathbf{u}_j^D) \\
\mathbf{L}_j^O & \leftarrow \mathbf{u}_j^O (\mathbf{L}_j^D)^{-T}
\end{align*}
\]

What changes in the rank-structured Cholesky?
Collapsed off-diagonal block is a (nearly low-rank) dense matrix.
Off-diagonal block compression

\[ G \leftarrow \text{rand}(|C_j|, r + p) \]
\[ C \leftarrow (L_j^O)^T G \]

\textbf{for} \ i = 1, \ldots, s \ \textbf{do} \,
\begin{align*}
C & \leftarrow (L_j^O)C \\
C & \leftarrow (L_j^O)^T C
\end{align*}

\[ U_j = \text{orth}(C) \]
\[ V_j = L_j^O U_j \]

Compress \textit{without} explicit \( L_j^O \):
- \text{Probe} \((L_j^O)^T\) with random \( G \)
- \text{Extract orth. row basis} \( U_j \)
- \( L_j^O = V_j U_j^T \implies V_j = L_j^O U_j \)

Where do we get the estimated rank bound \( r \)?
Could dynamically estimate the rank of $L_j^O$.
Practice: empirical rank bound $\approx \alpha \sqrt{k} \log(k)$. 
Compress off-diagonal blocks of sufficiently large supernodes \((j_1, j_2)\).
Don’t store *any* of $L^O_j$ for “interior” blocks
(Represent as $L^O_j = A^O_j (L^D_j)^{-1}$ when needed)
Diagonal block compression

\[ \mathbf{L}_j^D = \begin{pmatrix} \mathbf{L}_{j,1}^D & \mathbf{0} & \cdots & \cdots & \cdots & \cdots & \mathbf{0} \\ \mathbf{L}_{j,2}^D & \mathbf{L}_{j,3}^D & \mathbf{0} & \cdots & \cdots & \cdots & \mathbf{0} \\ \mathbf{L}_{j,4}^D & \cdots & \mathbf{0} & \mathbf{L}_{j,5}^D & \cdots & \cdots & \mathbf{0} \\ \mathbf{L}_{j,6}^D & \cdots & \cdots & \mathbf{L}_{j,7}^D & \cdots & \cdots & \cdots \end{pmatrix} \]

Basic observation: off-diagonal blocks are low-rank.
\((H\text{-matrix, semiseparable structure, quasiseparable structure, …})\)

Assumes reasonable ordering of unknowns!
Diagonal block compression

\[
\mathbf{L}_j^D \approx \begin{pmatrix}
\mathbf{L}_{j,1}^D & \mathbf{0} & \mathbf{0} \\
\mathbf{V}_{j,2}^D \mathbf{U}_{j,2}^{D T} & \mathbf{L}_{j,3}^D & \mathbf{0} \\
\mathbf{V}_{j,4}^D \mathbf{U}_{j,4}^{D T} & \mathbf{L}_{j,5}^D & \mathbf{0} \\
\mathbf{V}_{j,6}^D \mathbf{U}_{j,6}^{D T} & \mathbf{L}_{j,7}^D \\
\end{pmatrix}
\]

How do we get directly to this without forming \( \mathbf{U}_j^D \) explicitly?
Forming compressed updates

\[ \mathbb{D}_j, \mathbb{L}_{j,1}^D, \mathbb{L}_{j,2}^D, \mathbb{L}_{j,3}, \mathbb{L}_{j,4}^D, \mathbb{L}_{j,5}, \mathbb{L}_{j,6}^D, \mathbb{L}_{j,7} \]
Basic ingredients:

- Randomized algorithms form $\mathbf{u}_j^D$
- Rank-structured factorization of $\mathbf{u}_j^D$
- Randomized algorithm forms $\mathbf{L}_j^O$ (involves solves with $\mathbf{L}_j^D$)

Plus various optimizations.
Example: Large deformation of an elastic block
Example: Large deformation of an elastic block

Benchmark based on example from deal.II:
- Nearly-incompressible hyperelastic block under compression
- Mixed FE formulation (pressure and dilation condensed out)
- Tried both $p = 1$ and $p = 2$ finite elements
- Two load steps, Newton on each (14-15 steps)

Experimental setup:
- 8-core Xeon X5570 with 48 GB RAM
- LAPACK/BLAS from MKL 11.0
- PCG + preconditioners from Trilinos
RSC vs standard preconditioners \((p = 1, N = 50)\)
RSC vs standard preconditioners \((p = 2, N = 35)\)

- **RSC**
- **Jacobi**
- **ML**
- **ICC**

Graphs showing relative residual over iterations and seconds.
Time and memory comparisons ($p = 1$)

![Graphs showing time and memory comparisons for different methods.]

- Solve time (s) vs. $n$
- Memory (GB) vs. $n$

Methods compared:
- RSC
- ICC
- ML
- Jacobi
- Cholesky
Effect of in-separator ordering

Semi-sep diag relies on variable order – don’t want any old order!

- Apply recursive bisection based on spatial coords
- Use coordinates if known
- Else assign spectrally
Example: Trabecular bone model ($\approx 1M$ dof)

![Graph 1: Iterations vs. Relative Residual](image1)

- RSC2
- RSC1
- ML
- ICC

![Graph 2: Seconds vs. Relative Residual](image2)

- ICC
- ML
- RSC2
- RSC1

(U. Waterloo, ND40)
Example: Steel flange ($\approx 1.5M$ dof)
Conclusions

For more:
www.cs.cornell.edu/~bindel
bindel@cs.cornell.edu

Current status:
- Jeff has graduated! But...
- Code basically ready (still tweaking build system)
- Paper is mostly there (submit+arXiv before September)