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
\[ u = k \backslash f \]

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...”

(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 $\iff$ 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

\[ L_j \]

\[ L(D) \]

\[ L(O) \]

\[ L(C_j, C_j) \equiv L_j^D \]

\[ L(C_j, R_j) \equiv L_j^O \text{ collapsed} \]
Supernode factorization

\[ \mathbf{u}_j^D \leftarrow \mathbf{A}(\mathcal{C}_j, \mathcal{C}_j) \]
\[ \mathbf{u}_j^O \leftarrow \mathbf{A}(\mathcal{R}_j, \mathcal{C}_j) \]

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

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

\[ \mathbf{L}_j^D \leftarrow \text{cholesky}(\mathbf{u}_j^D) \]
\[ \mathbf{L}_j^O \leftarrow \mathbf{u}_j^O (\mathbf{L}_j^D)^{-T} \]

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

\[
\begin{align*}
L^D_j & \approx V_j^T U^T_j \\
L^O_j & \approx \text{Compressed } L^O_j
\end{align*}
\]
Off-diagonal block compression

\[ \mathbf{G} \leftarrow \text{rand}(|\mathcal{C}_j|, r + p) \]
\[ \mathbf{C} \leftarrow (\mathbf{L}_j^O)^T \mathbf{G} \]

\textbf{for} \ i = 1, \ldots, s \ \textbf{do}
\[ \mathbf{C} \leftarrow (\mathbf{L}_j^O)^T \mathbf{C} \]
\[ \mathbf{C} \leftarrow (\mathbf{L}_j^O)^T \mathbf{C} \]
\textbf{end}

\[ \mathbf{U}_j = \text{orth}(\mathbf{C}) \]
\[ \mathbf{V}_j = \mathbf{L}_j^O \mathbf{U}_j \]

Compress \textit{without} explicit \( \mathbf{L}_j^O \) :

- Probe \((\mathbf{L}_j^O)^T\) with random \( \mathbf{G} \)
- Extract orth. row basis \( \mathbf{U}_j \)
- \( \mathbf{L}_j^O = \mathbf{V}_j \mathbf{U}_j^T \implies \mathbf{V}_j = \mathbf{L}_j^O \mathbf{U}_j \)

Where do we get the estimated rank bound \( r \)?
Interaction rank

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 $L_j^O$ for “interior” blocks
(Represent as $L_j^O = A_j^O (L_j^D)^{-1}$ when needed)
Diagonal block compression

$\mathbf{L}^D_j = \begin{pmatrix} \mathbf{L}^D_{j,1} & \mathbf{0} & & & & & \\ & \mathbf{L}^D_{j,2} & \mathbf{L}^D_{j,3} & & & & \\ & & \mathbf{L}^D_{j,4} & & & & \\ & & & \mathbf{L}^D_{j,5} & \mathbf{0} & & \\ & & & & \mathbf{L}^D_{j,6} & \mathbf{L}^D_{j,7} & \\ & & & & & & \end{pmatrix}$

Basic observation: off-diagonal blocks are low-rank. ($H$-matrix, semiseparable structure, quasiseparable structure, ...)

Assumes reasonable ordering of unknowns!
Diagonal block compression

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

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

\[ \mathbf{D}_j, \mathbf{L}_{j,1}, \mathbf{L}_{j,2}, \mathbf{L}_{j,3}, \mathbf{L}_{j,4}, \mathbf{L}_{j,5}, \mathbf{L}_{j,6}, \mathbf{L}_{j,7} \]
Rank-structured supernode factorization

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
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 ($p = 2, N = 35$)
Time and memory comparisons ($p = 1$)

- Solve time (s)
  - RSC
  - Jacobi
  - ICC
  - ML
  - Cholesky

- Memory (GB)
  - RSC
  - 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)

![Diagram showing relative residual over iterations and seconds for different methods: RSC1, RSC2, ML, ICC.](image-url)
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)