The University of Texas at Austin
Department of Computer Science
College of Natural Sciences

Randomized Cholesky Factorization in Parallel

Tianyu Liang, George Biros (advisor) and Chao Chen (advisor)
University of Texas at Austin

The University of Texas at Austin
Oden Institute for Computational Engineering and Sciences

Overview

Problem: solve a large sparse linear system

$$L\,x = b, \tag{1}$$

where L is symmetric diagonally-dominant (SDD). Such a linear system arises from discretizing a certain class of Elliptic PDEs using finite element method, semi-supervised learning on graphs, maximum flow problem and others.

Related work

- Direct methods, e.g., SuperLU, MUMPS, PARDISO, CHOLMOD, require significant time and storage, especially for 3D problems.
- Iterative methods, e.g., Conjugate Gradient (CG) and multigrid, may converge slowly when the condition number of L is large.

Approach & Contribution: We compute an approximate Cholesky factorization

$$P^\top LP \approx GG^\top, \tag{2}$$

based on the randomized sampling scheme in [1, 2], where P is a permutation matrix and G is a sparse lower triangular matrix, and we use G as the preconditioner for solving Eq. (1) iteratively. Specific contributions are

1. shared-memory parallel implementation with C++ thread library.
2. evaluation of sparse matrix reordering strategies, which is crucial for practical performance.
3. benchmark on 3D variable-coefficient Laplace operators and general sparse matrices.

Randomized Cholesky Factorization/Elimination

Key idea

- **Bottleneck of Sparse Cholesky**: suppose a row/column i in L has $n+1$ non-zero entries. (Exact) Cholesky elimination leads to a $n \times n$ dense sub-matrix in the Schur complement, corresponding to a clique with $\mathcal{O}(n^2)$ edges among neighbors of i in the graph of L; see Fig. 1 and Fig. 2.
- **Conversion**: Eq. (1) is equivalent to solving a closely related Laplacian system. W.L.G., assume L is a Laplacian matrix with an underlying undirected weighted graph.
- **Randomized Sampling [1, 2]**: sample $\mathcal{O}(n)$ entries in the Schur complement, corresponding to n edges in the clique; see Fig. 3.
- **Invariant**: the expectation of sampled edges equal to the (full/exact) clique.

An example

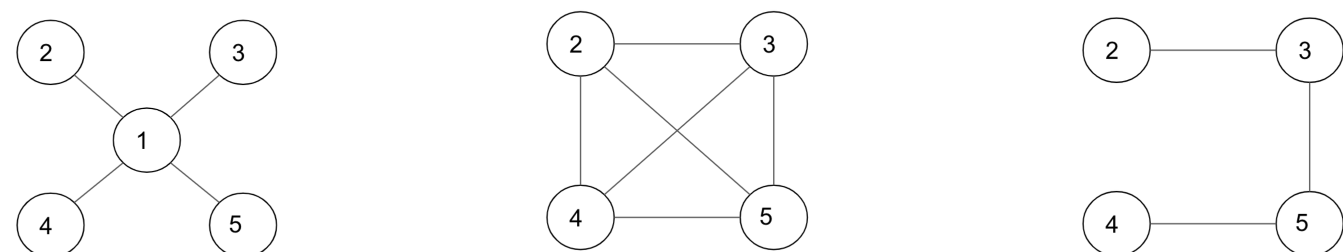


Figure 1: Graph of L; before vertex 1 is eliminated).
Figure 2: Eliminate "1" leads to clique among neighbors
Figure 3: Sample a sparse subset of the clique.

Algorithm 1 Randomized Sampling of Clique [2]

Input: irreducible Laplacian matrix $L \in \mathbb{R}^{N\times N}$ and elimination index k // vertex 1 in Fig. 1
Output: sparse Schur-complement update $C \in \mathbb{R}^{N\times N}$

1: $C = \mathbf{0}_{N\times N}$
2: Sort $\mathcal{N}$ in ascending order based on $|\ell_{ki}|$ for $i \in \mathcal{N}$ // neighbors of k in the graph
3: $S = \ell_{kk}$
4: **while** $|\mathcal{N}| > 1$ **do**
5: Let i be the first element in $\mathcal{N}$ // loop over neighbors (i is the red node in Fig. 4)
6: $\mathcal{N} = \mathcal{N}/\{i\}$ // remove i from the set
7: $S = S + \ell_{ki}$
8: Sample j from $\mathcal{N}$ with probability $|\ell_{kj}|/S$ // j is the blue node in Fig. 4
9: $C = C - \frac{S\,\ell_{ki}}{\ell_{kk}}\,\boldsymbol{b}_{ij}\boldsymbol{b}_{ij}^\top$
10: **end while**

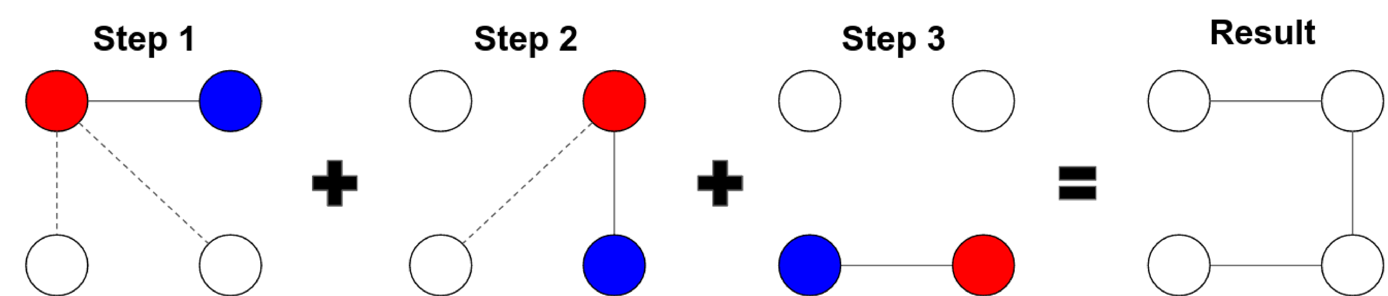


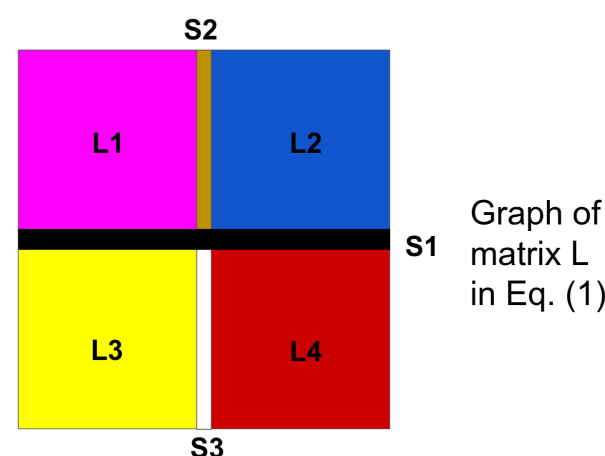
Figure 4: Step-by-step illustration of randomized sampling [1, 2]. Red node goes over neighbors of the eliminated node. Blue node is sampled from the rest of neighbors, and the edge between red and blue is selected.

Parallel Algorithm and Matrix Reordering

Sparse matrix reordering given n_T threads:

- compute $\log_2(n_T)$-level nested dissection algebraically with ParMETIS or PT-Scotch; see Fig. 5.
- compute approximate minimum degree (AMD) ordering for partitions at the leaf level; see Fig. 6.

Nested dissection of sparse matrix (left) and the associated tree (right)



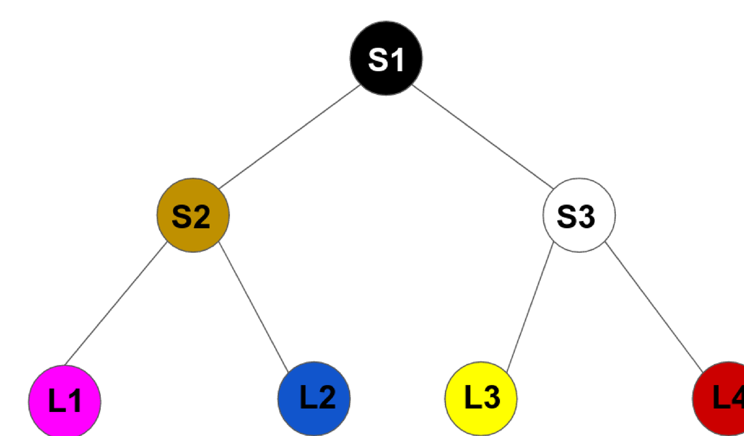


Figure 5: S1 (black) is the top separator, S2 (brown) and S3 (white) are two (decoupled) separators at the second level, and L1, L2, L3, L4 (pink, blue, yellow and red) are four leaf nodes decoupled from each other.

Figure 6: Task dependency: every node depends on its children (if exist); nodes at the same level can be processed in parallel.

Algorithm 2 Parallel Randomized Cholesky Factorization (level-by-level traversal)

1: Compute sparse matrix reordering as stated earlier.
2: **for** leaf level **to** root **do**
3: **for all** nodes at this level **do** // parallel for loop
4: **if** this node has children **then**
5: Merge Schur complement updates from both children.
6: **end if**
7: Eliminate vertices owned by this node with Algorithm 1.
8: **if** this node has parent **then**
9: Send sampled edges (Schur complement updates) to the parent
10: **end if**
11: **end for** // synchronize all threads
12: **end for**

- Our approach similar to the classical *multifrontal solver* (MF). The difference is that our fronts stay sparse because of the randomized sampling scheme, whereas the fronts are dense in MF.
- We implemented a task-based post-order tree traversal using the C++ thread library to avoid synchronization between levels.
- We use the Intel TBB memory allocator to alleviate memory contention among multiple threads.

Numerical Experiments and Results

`rchol`: our randomized Cholesky factorization
`ichol`: thresholding incomplete Cholesky factorization.
nnz: # non-zeros in L in Eq. (1)
nnz ratio: # non-zeros in `rchol` divided by nnz
AMD: approximate minimum degree ordering
PCG: preconditioned Conjugate Gradient method

N : size of L in Eq. (1)
t_p : time to compute AMD ordering in seconds
t_f : time to compute Eq. (2) in seconds
t_s : PCG time of solving Eq. (1) in seconds
n_{it} : PCG iterations required for tolerance 1e−10
n_T: number of threads/cores

Serial Test # 1: Poisson Equation $\nabla(a(x)\cdot\nabla u(x)) = f, x \in [0,1]^3$ with random high-contrast coefficients

$$a(x) = \begin{cases} \rho^{-1/2} & \text{in white region} \\ \rho^{1/2} & \text{in black region} \end{cases} \tag{3}$$

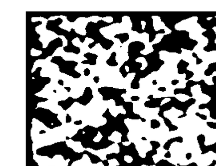


Table 1: *Comparison between* **rchol** *preconditioner and* **ichol** *preconditioner (N=2.0e+6, nnz=1.4e+7).* ***ρ is the contrast of coefficients*** *in Eq. (3).* **rchol** *w/ AMD ordering,* **ichol** *better w/o reorderinig.*

ρ	rchol					ichol			
	nnz ratio	t_p	t_f	t_s	n_{it}	nnz ratio	t_f	t_s	n_{it}
1e+0	3.2	3.5	2.5	15	49	3.4	0.7	29	102
1e+1	3.4	3.6	2.7	18	53	3.5	0.9	52	175
1e+2	3.6	3.6	2.9	29	88	3.6	1.0	68	228
1e+3	3.6	3.5	2.9	39	118	3.7	0.9	76	247
1e+4	3.6	3.5	3.0	42	124	3.8	1.0	83	272

- Under similar nnz ratio, `rchol` required much less iterations and time to solve Eq. (1).
- PCG may stagnate for higher ratio, but both methods reached similar relative residual(accuracy).

Results (cont.)

Serial Test # 2: SuiteSparse Matrix Collection

	Name	N	nnz	Property
# 1	ecology2	1.0e+5	5.0e+6	SDD
# 2	parabolic_fem	5.3e+5	3.7e+6	SDD
# 3	apache2	7.2e+5	4.8e+6	SPD (not SDD)
# 4	G3_circuit	1.6e+6	7.7e+6	SPD (not SDD)

Table 2: *Comparison between* **rchol** *preconditioner and* **ichol** *preconditioner (***rchol** *with AMD ordering,* **ichol** *better w/o reorderinig).*

	rchol					ichol			
	nnz ratio	t_p	t_f	t_s	n_{it}	nnz ratio	t_f	t_s	n_{it}
# 1	2.41	0.4	0.5	16	89	2.72	0.3	132	798
# 2	2.27	0.4	0.5	10	65	2.29	0.3	58	411
# 3	2.93	0.6	0.8	10	60	2.96	0.3	52	322
# 4	2.68	1.4	1.1	21	96	2.75	0.4	86	379

- W/ similar nnz ratio, `rchol` required much less iterations and time to solve Eq. (1).
- `rchol` designed for SDD matrices and remain efficient for symmetric positive definite (SPD) but non-SDD Matrices.

Parallel Scalability Test with n_T threads

Matrices from discretizing Poisson equation with $\rho = 1$ in Eq. (3) on 3D regular grids using standard 7-point finite difference with single precision (float).

n_T	nnz ratio	t_f
1	3.56	19.9
2	3.60	10.7
4	3.61	5.7
8	3.63	3.3
16	3.66	2.3

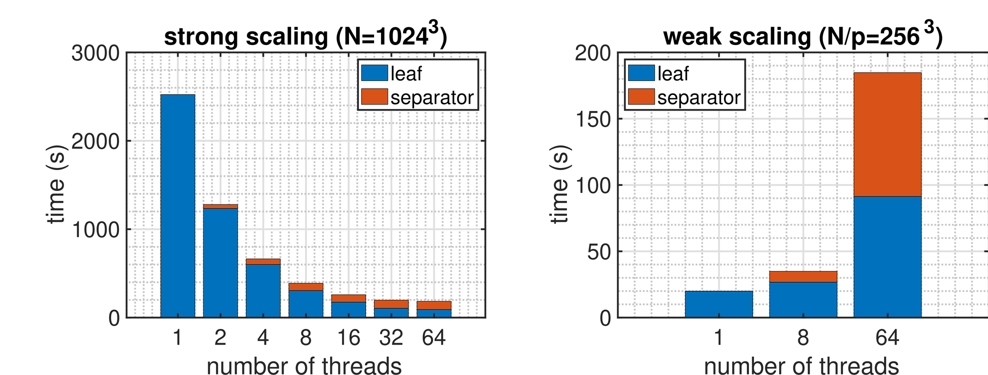
Table 3: N = 256^3, nnz = 1.2e+8

n_T	nnz ratio	t_f
1	3.93	226
2	3.98	113
4	3.98	58
8	3.99	35
16	4.00	23
32	4.02	18
64	4.02	16

Table 4: N = 512^3, nnz = 9.4e+8

n_T	nnz ratio	t_f
1	4.31	2523
2	4.37	1279
4	4.39	664
8	4.38	388
16	4.38	258
32	4.39	197
64	4.38	184

Table 5: N = 1024^3, nnz = 7.5e+9



- One thread per core on Intel Xeon Platinum 8280M ("Cascade Lake") with 112 cores on four sockets (28 cores/socket).
- Memory of G in Eq. (2): about 200 GB for $N = 1024^3$ with single precision.
- Scaling bottleneck: work on separators increases (parallelism on leaf nodes decreases) as n_T increases. Future work is to parallelize work on separators.

Conclusion

1. `rchol` preconditioner required less iteration and time to solve Eq. (1) than incomplete Cholesky factorization with drop tolerance.
2. `rchol` is easy to implement and typically requires only a few times more memory than storing the input sparse matrix (a light-weight preconditioner).
3. Parallel algorithm/implementation on a multicore CPU scales up to 64 threads/cores for a discretized Poisson equation on a $1024 \times 1024 \times 1024$ grid, i.e., one billion unknowns.
4. Source code for the implementation can be found at
`https://github.com/ut-padas/randchol`

Reference

[1] Rasmus Kyng and Sushant Sachdeva. 2016. Approximate gaussian elimination for laplacians-fast, sparse, and simple. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 573–582.
[2] Daniel A. Spielman. 2020. `https://github.com/danspielman/Laplacians.jl`.

LaTeX TikZposter