Randomized Cholesky Factorization in Parallel

Tianyu Liang
University of Texas at Austin
Austin, Texas

George Biros (advisor)
University of Texas at Austin
Austin, Texas

Chao Chen (advisor)
University of Texas at Austin
Austin, Texas

1 INTRODUCTION

Solving large sparse linear systems is a fundamental building block in scientific computing. We are interested in solving a system of equations

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

where $L \in \mathbb{R}^{N\times N}$ is an irreducible (cannot be reduced to two independent sub-problems) symmetric diagonally dominant[1] (SDD) matrix. Such linear systems appear in many applications in science and engineering, most commonly as discretizations of PDEs.

Existing software packages of sparse linear solvers fall into two categories: sparse direct methods and iterative methods. Sparse direct solvers usually require significant computation and storage, especially for solving problems in three dimensions (3D). While iterative solvers require $O(N)$ computation and memory for every step, the number of iterations typically increases as the square root of the condition number. Therefore, appropriate preconditioning is required to render better practical performance.

In this work, we present a preconditioner based on the randomized sampling scheme in [8, 12]. In particular, we compute an approximate Cholesky factorization

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

where P is a permutation matrix and G is a sparse lower triangular matrix. Our contributions include

(1) parallel algorithm for shared-memory machines implemented using C++ thread library,
(2) evaluation of sparse matrix reordering strategies, which is crucial for practical performance, and
(3) benchmark on variable-coefficient Laplace operators in 3D and general sparse matrices.

2 RANDOMIZED CHOLESKY

For standard Cholesky factorization, we start with $L^{(1)} = L$, and at the k-th step ($k = 1, 2, \ldots, N-1$), we perform the following rank-1 update:

$$L^{(k+1)} = L^{(k)} - \frac{1}{\ell_{kk}} L^{(k)}(:,k)\, L^{(k)}(k,:), \tag{3}$$

where ℓ_{kk}, $L^{(k)}(:,k)$ and $L^{(k)}(k,:)$ denote the k-th diagonal, column and row in $L^{(k)}$, respectively. Suppose $L^{(k)}(k,:)$ has $n+1$ non-zeros; it is well-known that $L^{(k+1)}$ contains a dense $n \times n$ sub-matrix corresponding to the clique among n neighbors of k in the underlying graph of $L^{(k)}$ [3].

Here we follow the randomized sampling scheme in [8, 12] and keep only $O(n)$ fill-in in $L^{(k+1)}$. To that end, we need to first convert Eq. (1) to a closely related Laplacian linear system [5]. Without loss of generality, we assume L is a Laplacian matrix[2].

[1] $L = L^\top$ and $\ell_{ii} \geq \sum_{j\neq i} |\ell_{ij}|$ for all i.
[2] $L = L^\top$, $\ell_{ii} = \sum_{j\neq i} |\ell_{ij}|$ for all i, and $\ell_{ij} < 0$ when $i \neq j$.

Define $L^{(k)}|_k = \sum_{\ell_{ki}\neq 0} (-\ell_{ki})\, \boldsymbol{b}_{ki}\boldsymbol{b}_{ki}^\top$, where ℓ_{ki} is the (k,j)-th entry in $L^{(k)}$, $\boldsymbol{b}_{ki} = e_k - e_i$ is the difference of two standard bases in $\mathbb{R}^N$. Observe that $L^{(k+1)}$ can be rewritten as the sum of two Laplacian matrices as follows [8]:

$$\underbrace{L^{(k)} - L^{(k)}|_k}_{\text{Laplacian matrix}} + \underbrace{L^{(k)}|_k - \frac{1}{s} L^{(k)}(:,k) L^{(k)}(k,:)}_{\text{Laplacian matrix}}.$$

In particular, the second term equals to

$$\sum_{\ell_{ki}\,\ell_{kj}\neq 0} \frac{\ell_{ki}\,\ell_{kj}}{2s}\, \boldsymbol{b}_{ij}\boldsymbol{b}_{ij}^\top, \tag{4}$$

which is a $n \times n$ dense matrix. From a graph perspective, Eq. (4) can be interpreted as a clique with $n(n-1)/2$ edges.

The randomized scheme [12] samples only $n-1$ edges from the clique, which still equals to Eq. (4) in expectation. Because of the reduced amount of fill-in, the randomized Cholesky factorization is much faster and requires much less memory than the exact Cholesky factorization. The pseudo code of the randomized Cholesky factorization and the sampling scheme are shown in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1 Randomized Cholesky Factorization

Input: irreducible Laplacian matrix L
Output: lower triangular matrix G
$G = \mathbf{0}_{N\times N}$
for $k = 1$ **to** $N-1$ **do**
  $G(:,k) = L(:,k)/\sqrt{\ell_{kk}}$
  $L = L + L^{(k)} + \text{SampleClique}(L)$
end for

Algorithm 2 Sample Clique

Input: irreducible Laplacian matrix L and elimination index k
Output: sparse Schur-complement update C
$C = \mathbf{0}_{N\times N}$
Sort $\mathcal{N} = \{i : \ell_{ki} \neq 0, i \neq k\}$ in ascending order based on $|\ell_{ki}|$
$S = \ell_{kk}$
while $|\mathcal{N}| > 1$ **do**
  Let i be the first element in $\mathcal{N}$ // loop over neighbors
  $\mathcal{N} = \mathcal{N}/\{i\}$ // remove i from the set
  $S = S + \ell_{ki}$
  Sample j from $\mathcal{N}$ with probability $|\ell_{kj}|/S$
  $C = C - \frac{S\,\ell_{ki}}{\ell_{kk}}\, \boldsymbol{b}_{ij}\boldsymbol{b}_{ij}^\top$
end while

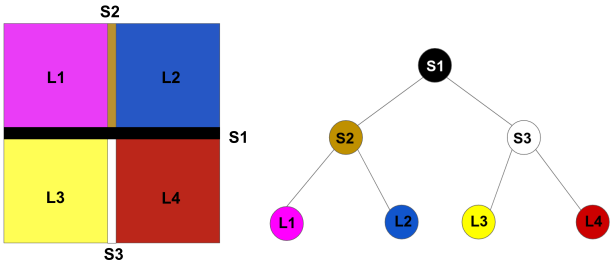

Figure 1: (Left) nested-dissection of a mesh/graph of L. (Right) nested-dissection tree.

3 PARALLEL ALGORITHM

Suppose there exists an underlying mesh for matrix L, and we split it into two disconnected components and a (vertex) separator. Then, we can apply the randomized Cholesky factorization on the two disconnected pieces in parallel. With more than two threads, we can compute the same partitioning recursively on the two independent partitions; see Fig. 1(left) for a pictorially illustration. Technically, the above procedure is known as the nested dissection [4] and can be computed algebraically using METIS/ParMETIS [6, 7] or Scotch/PT-Scotch [2].

The nested dissection partitioning is naturally associated with a tree structure, where leaf nodes correspond to disconnected regions and the other nodes correspond to separators at different levels; see Fig. 1(right). This tree maps to the task graph of a parallel algorithm: every tree node/task stands for applying the randomized Cholesky factorization to associated rows/columns of L. It is obvious that tasks at the same level can be executed in parallel.

Notice a task depends not only on its children but also some of their descendants, and we employ a multi-frontal type approach [9] in our parallel algorithm, where a task receives the Schur complement updates from only its two children and sends necessary updates to its parent (a task communicates only with its children and parent).

To summarize the parallel algorithm, we first compute a nested dissection of $\log_2(n_T)$ levels when n_T threads are available. Then, within each independent region at the leaf level, we employ the approximate minimum degree ordering [1], a fill-in reducing heuristic that can be computed much faster than nested dissection. Finally, we traverse the task tree in post order to generate tasks and map them recursively to all threads/cores following the nested dissection partitioning.

4 RESULTS

Our poster presents three numerical experiments to answer the following three questions:

(1) How does our preconditioner (`rchol`) compared with the thresholding incomplete Cholesky factorization [10, 11] (`ichol`)?
(2) What type of problems can be solved with the `rchol` preconditioner?
(3) How does our parallel algorithm scale on a multi-core CPU?

To answer the first question, we compared `rchol` with `ichol` on a discretized variable-coefficient Poisson's equation on a regular grid with Dirichlet boundary condition. Specifically, we construct a high-contrast random field of coefficients $a(x)$ as follows. First, generate standard uniform random numbers a_i on a regular grid and compute the median μ. Second, convolve the coefficients with an isotropic Gaussian of width $4h$, where h is the grid spacing. Last, quantize the coefficients by setting

$$a_i = \begin{cases} \rho^{-1/2}, & a_i \le u \\ \rho^{1/2}, & a_i > u \end{cases} \tag{5}$$

The resulting linear system has a condition number of $O(\rho N)$, where ρ ranges from 1 to 10^4. Our results show that the number of iterations and the preconditioned Conjugate Gradient (PCG) total time required with `rchol` are always much less than those with `ichol` under the same memory constraint.

For the second question, we tested `rchol` on four symmetric positive definite (SPD) matrices from the SuiteSparse Matrix Collection[3] (with unknown mesh), and we compared `rchol` with `ichol` through PCG. Although two of the matrices are not SDD, we were still able to apply the `rchol` preconditioner successfully, and the results also show `rchol` leads to much less number of iterations and total PCG time than `ichol`.

Lastly, we presented parallel scalability on the discretized variable-coefficient Poisson's equation with $\rho = 1$ in Eq. (5) on a 1024×1024×1024 grid (*one billion* unknowns). The preconditioner required about 200 GB in single precision demonstrating the superiority of memory footprint over direct solvers. Moreover, the algorithm scaled up to 64 threads with every thread bound to a single core. As the number of threads increases, the load imbalance from graph partitioning, the upsurge in time spent on separator along with memory contention among threads become the bottleneck.

5 CONCLUSION

Randomized Cholesky is unique because it is one of the few existing non-deterministic preconditioner for iterative methods. Compared with the reference method, `rchol` attain better performance in terms of iteration counts and total PCG time. Moreover, `rchol` preconditioner typically requires only a few times more memory than storing the original problem. Last but not least, `rchol` achieved satisfactory speedup with up to 64 threads on Intel Xeon Platinum 8280M. Our implementation of the algorithm can be found at

https://github.com/ut-padas/randchol

REFERENCES

[1] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.* 17, 4 (1996), 886–905.
[2] Cédric Chevalier and François Pellegrini. 2008. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel computing* 34, 6-8 (2008), 318–331.
[3] Timothy A Davis, Sivasankaran Rajamanickam, and Wissam M Sid-Lakhdar. 2016. A survey of direct methods for sparse linear systems. *Acta Numerica* 25 (2016), 383–566.
[4] Alan George. 1973. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.* 10, 2 (1973), 345–363.
[5] Keith D Gremban. 1996. *Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems.* Ph.D. Dissertation. Carnegie Mellon University.
[6] George Karypis and Vipin Kumar. 1999. A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing* 20, 1 (1999), 359–392.

[3] https://sparse.tamu.edu/

[7] George Karypis, Kirk Schloegel, and Vipin Kumar. 1997. Parmetis: Parallel graph partitioning and sparse matrix ordering library. *Version 1.0, Dept. of Computer Science, University of Minnesota* 22 (1997).
[8] 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.
[9] Joseph WH Liu. 1992. The multifrontal method for sparse matrix solution: Theory and practice. *SIAM review* 34, 1 (1992), 82–109.
[10] Thomas A Manteuffel. 1980. An incomplete factorization technique for positive definite linear systems. *Mathematics of computation* 34, 150 (1980), 473–497.
[11] Yousef Saad. 2003. *Iterative methods for sparse linear systems*. SIAM.
[12] Daniel A. Spielman. 2020. https://github.com/danspielman/Laplacians.jl.