Parallel Cholesky Factorization of Sparse Matrices Using Elimination Trees

by

Artun Akdoğan

M.S., Computer Engineering, Boğaziçi University, 2023

MS Thesis Proposal submitted to

Department of Computer Engineering

Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2023

Parallel Cholesky Factorization of Sparse Matrices Using Elimination Trees

APPROVED BY:

Prof. Name Surname (Thesis Supervisor)	. . . .
Assoc. Prof. Name Surname	. . . .
Assist. Prof. Name Surname	. . . .
Name Surname, Ph.D.	. . . .

DATE OF APPROVAL: DD.MM.YYYY

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

LIST OF ACRONYMS/ABBREVIATIONS

CSR	Compressed Sparse Row
API	Application Programming Interface
D-SAB	Delft Sparse Architecture Benchmark
LU	Lower-Upper
DTET	Double Task Elimination Tree

1. INTRODUCTION

Linear algebra practices form the cornerstone of scientific computing, and among these, direct linear solvers hold significant importance. For many situations, the input matrix is sparse, typically containing a limited number of nonzero elements. The elimination tree provides structural information to ease parallelization for sparse matrix factorization, which is less space-consuming than the dense matrix projection, even though there are other algorithms to achieve efficient computation of sparse matrix equations like SuperLU [2]. Cholesky factorization is significantly more efficient when dealing with positive symmetric definite matrices, as only the corresponding lower triangular matrix L has to be calculated for the result, bypassing the requirement to calculate the upper triangular matrix in LU decomposition.

When performing row operations to eliminate nonzero entries in specific locations, it is possible to inadvertently generate new nonzero elements elsewhere, known as ”fill” [3]. Efficient sparse matrix algorithms rely on predictive data structures based on graph models assembled for the factorization algorithm. Those predictions aim to reduce the fill to optimize the efficiency of these methods.

A prevalent approach when tackling systems like $Ax = b$ involves decomposing matrix A into triangular matrices. The Cholesky factorization is a superior choice for symmetric and definitively positive matrices, outpacing alternatives such as LU factorization in speed and efficiency. The Cholesky decomposition expresses matrix A as the product of corresponding lower triangular matrix L and its transpose L^T, which can be succinctly written as $A = LL^T$ [4]. While computing the diagonal cells of the triangular matrix L, equation 1.1 can be used, while for other cells in this triangular matrix, equation 1.2 could be utilized.

$$L_{i,i} = \sqrt{A_{i,i} - \sum_{k=1}^{i-1} L_{i,k}^2} \quad \text{for } i = i \tag{1.1}$$

$$L_{i,l} = \frac{1}{L_{l,l}} \left(A_{i,l} - \sum_{k=1}^{l-1} L_{i,k} L_{l,k} \right) \quad \text{for } i > l \tag{1.2}$$

When analyzing symmetric matrices, undirected graphs can be used. Some matrix A can be illustrated with the corresponding graph $G(A) = (X(A), E(A))$. Nodes in $X(A)$ represent the matrix's columns and rows. In contrast, edges $E(A)$ for the matrix A represent the nonzero cells [5].

$$\begin{pmatrix} 9 & 2 & 4 & 0 & 0 & 0 & 0 & 0 \\ 2 & 10 & 0 & 0 & 0 & 1 & 0 & 0 \\ 4 & 0 & 5 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 6 & 2 & 4 & 2 & 0 \\ 0 & 0 & 0 & 2 & 5 & 0 & 4 & 0 \\ 0 & 1 & 0 & 4 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 2 & 4 & 0 & 5 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 2 \end{pmatrix}$$

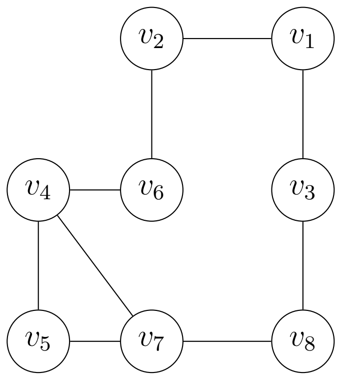

Figure 1.1. Matrix A and the adjacency graph

This undirected graph can be converted into a tree structure with the following procedures. Assume A to be an NxN irreducible sparse symmetric positive-definite matrix and L to be an affiliated lower triangular matrix that holds for the Cholesky decomposition of A is $A = LL^T$ shown in Fig. 1.1. Let $F = L + L^T$ similar to Fig. 1.2. Remove all the nonzeros per column besides the first nonzero below the diagonal for every column of L. Let the resulting matrix be L_t and corresponding matrix F

$$\begin{pmatrix}
3.00 & 0.67 & 1.33 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00\\
0.67 & 3.09 & -.29 & 0.00 & 0.00 & 0.32 & 0.00 & 0.00\\
1.33 & -.29 & 1.77 & 0.00 & 0.00 & 0.05 & 0.00 & 0.56\\
0.00 & 0.00 & 0.00 & 2.45 & 0.82 & 1.63 & 0.82 & 0.00\\
0.00 & 0.00 & 0.00 & 0.82 & 2.08 & -.64 & 1.60 & 0.00\\
0.00 & 0.32 & 0.05 & 1.63 & -.64 & 2.41 & -.13 & -.01\\
0.00 & 0.00 & 0.00 & 0.82 & 1.60 & -.13 & 1.32 & 0.75\\
0.00 & 0.00 & 0.56 & 0.00 & 0.00 & -.01 & 0.75 & 1.05
\end{pmatrix}$$

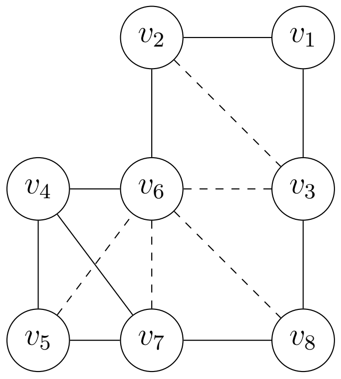

Figure 1.2. Matrix F and adjacency graph with fill

to become $F_t = L_t + L_t^T$ like in Fig. 1.3. The resulting graph can be mentioned as the elimination tree of matrix A, which entirely depends on the structure and initial ordering of the matrix [5].

$$\begin{pmatrix}
3.00 & 0.67 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00\\
0.67 & 3.09 & -.29 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00\\
0.00 & -.29 & 1.77 & 0.00 & 0.00 & 0.05 & 0.00 & 0.00\\
0.00 & 0.00 & 0.00 & 2.45 & 0.82 & 0.00 & 0.00 & 0.00\\
0.00 & 0.00 & 0.00 & 0.82 & 2.08 & -.64 & 0.00 & 0.00\\
0.00 & 0.00 & 0.05 & 0.00 & -.64 & 2.41 & -.13 & 0.00\\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -.13 & 1.32 & 0.75\\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.75 & 1.05
\end{pmatrix}$$

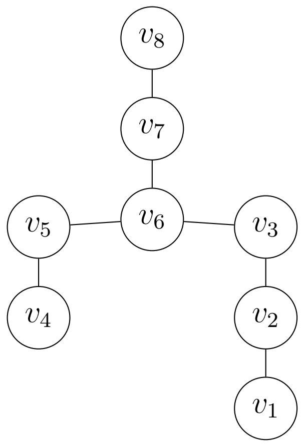

Figure 1.3. Matrix F_t and the elimination tree

If a cell of the sparse matrix is nonzero, then a path exists between two related vertices on the matrix graph. In order to determine the layout of an N-sized elimination tree T(A), the parent vectors of elements from 1 to N in T(A) must be computed. An algorithm to build the PARENT vector from matrix F_t has been given in figure 1.4.

```
for $i := 1$ to $N$ do
  $PARENT[i] := 0$;
  for $k < i$ and $l_{ik} \neq 0$ do
    if $PARENT[k] = 0$ then
      $PARENT[k] := i$;
    end if
  end for
end for
```

Figure 1.4. Elimination Tree Determination Algorithm [1]

This algorithm allows the elimination tree $T(A)$ to be stored while its layout is protected. With further optimizations, such as storing the current subtree's actual root in a temporary matrix as Liu et al. described [5], it is possible to obtain $O(mlogN)$ time complexity. The resulting PARENT vector is shown in table 1.

k	1	2	3	4	5	6	7	8
PARENT[k]	2	3	6	5	6	7	8	0

Table 1.1. PARENT vector of T(A)

Different types of parallel factorization graphs exist for solving linear equations. The granularity of tasks in those graphs corresponds to the amount of work performed by this task. An operation task graph is a fine-grained graph where each task is related to a simple arithmetic procedure of the factorization procedure. In medium-grained operations, simple arithmetic operations are grouped into subtasks. For example, columns of a sparse matrix might be grouped to solve its Cholesky decomposition. Finally, if we treat the subtrees in the elimination tree structure as a single procedure, it would become a coarse-grained approach. For this case, some coarse-grained algorithm examples presented by Liu et al. are the submatrix-Cholesky, row-Cholesky, and column-Cholesky approaches, which group matrices as described by their name [6]. When an approach's granularity increases, the algorithm's parallelization also increases, but with the drawback of the overhead increase when ordering those subtasks.

Some elements of the subtask may require data from other partitions. In this scenario, the current subtask might be suspended, and the processor can be freed to compute another subtask. Still, if not properly organized, this situation can increase the processors' busy time when no valuable computation can be done. Another approach could be taking preemptive scheduling measures before the main task begins, such as adding priority values for each subtask to minimize this busy time [6].

2. RELATED WORK

Sparse matrix optimization has been extensively researched across diverse solutions. For instance, Philipp Herholz et al. propose an approach for geometry processing by reusing Cholesky factorizations on the entire mesh to compute linear solutions on sub-meshes [7]. Nevertheless, parallelization stands out as one of the most crucial optimizations for solving sparse matrix equations by allowing simultaneous computation on non-dependent elements of the matrix.

Even though working with symmetric matrices is straightforward and efficient, this limitation can obstruct some of the essential methods in real-world scenarios. Constructing the elimination tree of unsymmetric matrices using directed graphs is possible. S. C. Eisenstat et al. present an algorithm to build the elimination trees of unsymmetric matrices and how to reorder them to their bordered block triangular (BBT) form [8].

Loris Marchal et al. focused on the multifrontal method for factorizing sparse matrices. They introduced a piecewise linear speedup model, demonstrating its utility in creating better schedules for linear algebra kernels [9]. Another study focused on task scheduling to accomplish load balancing and significant concurrency for Cholesky factorization in sparse matrices [10]. Besides the scheduling, Changjiang Gou et al. focused on a memory-efficient algorithm for tree partitioning [11].

Some other studies include the optimizations on the sparse linear equations. Mingzhen Li et al. proposed the swCholesky system, an optimized sparse Cholesky factorization implementation for Sunway Manycore Architecture [12]. Valentin Le Fevre et al. implemented the OPT-D algorithm that offers automatic and dynamic selective nesting for the parallel Cholesky factorization, which provides an average speedup of 1.75x [13], which they also optimized the algorithm for the A64FX processor to obtain an average speedup of 1.46x [14]. Chao Liu et al. presented the DTED algorithm, which parallelizes the forward and backward substitutions of sparse linear equations [15].

Besides, CHOLMOD is a software package offering Cholesky factorization procedures for sparse matrices, including both supernodal and non-supernodal approaches. Supernodal factorization groups multiple columns of the sparse matrix into larger, dense blocks called supernodes. Additionally, CHOLMOD supports efficient update and downdate operations, which allows the modification of the Cholesky factorization when new rows or columns are added or removed from the original matrix [16].

3. METHOD

With this paper, an optimized and parallelized Cholesky factorization algorithm will be implemented. As sparse matrices will be used in this work, the file type for the input and output will be in CSR format to minimize storage requirements. The methods to prepare this algorithm will be as follows:

- A sparse symmetric positive definite Cholesky factorization algorithm will be developed.
- A multicore CPU-based algorithm will be implemented. Medium-grained and coarse-grained approaches will be considered, and the running times of those algorithms will be compared.
- Preemptive scheduling approaches such as prioritizing will be used to optimize the system's running time.
- Memory and cache usage enhancements specific to the CPU will be made while optimizing the parallel algorithm for optimal runtime information.
- Different approaches utilizing the elimination trees to structure the parallelization strategy will be tested and compared with previous parallelized methods. The Intel Cypress Cove architecture CPU and 32GB RAM will be used in the testing environment.
- If time permits, we will also look into GPU acceleration on computationally intensive parts and observe the runtime results to present how much it differs from our CPU-only algorithm.

The new algorithm will be written according to the OpenMP API Specifications [17]. While OpenMP simplifies the parallelization process with abstraction, it also has optimization parameters like "simd" to make parallelization as vector-friendly as possible. The resulting code will be tested using the D-SAB sparse matrix benchmark suite [18] with different input matrices from SuiteSparse Matrix Collection [19]. Finally, this work will be compared with other algorithms to further discuss the new algorithm's enhancements.

Tang et al. [20] were able to observe a 10%-25% performance boost compared to CHOLMOD (SuiteSparse v4.6.0-beta) by applying the pipelining method over the subtree algorithm that is introduced in the CHOLMOD beta version which surpasses the older version with four times performance gain. We will investigate runtime differences between our CPU-only algorithm and the CHOLMOD [16] package. Our main aim with this work will be to create an efficient CPU-only algorithm.

4. TIMELINE

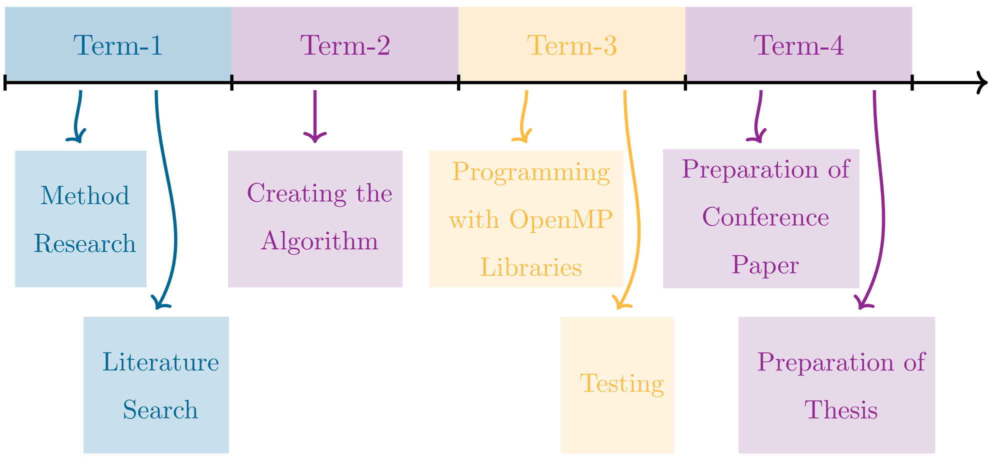

Figure 4.1. Planned timeline for the thesis

In the first term, method and literature research has been completed. For the planned timeline, the second term will consist of creating the algorithm for optimized Cholesky factorization. In contrast, in the third term, this algorithm will be implemented in OpenMP [17] and tested with sparse matrix benchmark suite [18]. Finally, the conference paper and thesis will be prepared in the fourth term.

It is crucial to note that while this timeline sets specific deadlines for the thesis, there is flexibility for potential acceleration. If circumstances permit, the schedule may be expedited, allowing for earlier completion.

5. CONCLUSION

The planned methods of parallel Cholesky factorization research and related works in parallel sparse matrix computation have been successfully researched in the first term. In the subsequent terms, the implementation of the algorithm and writing of the relevant conference paper and thesis are planned.

Sparse matrix representations are critical in computational studies, and elimination trees are essential for effectively parallelizing sparse matrix operations. In specific scenarios, Cholesky factorization outperforms alternative methods in computational speed, and its further optimization could be imperative for accelerating particular applications. Undoubtedly, further research and exploration in this realm hold promise for optimizing related applications.

REFERENCES

1. Liu, J. W., "The Role of Elimination Trees in Sparse Factorization", *SIAM Journal on Matrix Analysis and Applications*, Vol. 11, No. 1, p. 149, 1990, `https://doi.org/10.1137/0611010`.

2. Li, X. S., "An Overview of SuperLU: Algorithms, Implementation, and User Interface", *ACM Trans. Math. Softw.*, Vol. 31, No. 3, p. 302–325, sep 2005, `https://doi.org/10.1145/1089014.1089017`.

3. Pothen, A. and S. Toledo, "Elimination Structures in Scientific Computing", *Handbook of Data Structures and Applications*, 2004, `https://api.semanticscholar.org/CorpusID:12306726`.

4. Kincaid, D. R. and E. W. Cheney, *Numerical Analysis: Mathematics of Scientific Computing*, Brooks/Cole, Pacific Grove, Calif., 1991.

5. Liu, J. W., "The Role of Elimination Trees in Sparse Factorization", *SIAM Journal on Matrix Analysis and Applications*, Vol. 11, No. 1, pp. 134–172, 1990, `https://doi.org/10.1137/0611010`.

6. Liu, J. W., "Computational models and task scheduling for parallel sparse Cholesky factorization", *Parallel Computing*, Vol. 3, No. 4, pp. 327–342, 1986, `https://www.sciencedirect.com/science/article/pii/0167819186900141`.

7. Herholz, P. and M. Alexa, "Factor Once: Reusing Cholesky Factorizations on Sub-Meshes", *ACM Trans. Graph.*, Vol. 37, No. 6, dec 2018, `https://doi.org/10.1145/3272127.3275107`.

8. Eisenstat, S. C. and J. W. H. Liu, "Algorithmic Aspects of Elimination Trees for Sparse Unsymmetric Matrices", *SIAM Journal on Matrix Analysis and Applications*, Vol. 29, No. 4, pp. 1363–1381, 2008, `https://doi.org/10.1137/050643581`.

9. Marchal, L., B. Simon, O. Sinnen and F. Vivien, “Malleable Task-Graph Scheduling with a Practical Speed-Up Model”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 29, No. 6, pp. 1357–1370, 2018.

10. Geist, G. A. and E. Ng, “Task scheduling for parallel sparse Cholesky factorization”, *International Journal of Parallel Programming*, Vol. 18, No. 4, pp. 291–314, 1989, `http://dx.doi.org/10.1007/BF01407861`.

11. Gou, C., A. Benoit and L. Marchal, “Memory-Aware Tree Partitioning on Homogeneous Platforms”, *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pp. 321–324, 2018.

12. Li, M., Y. Liu, H. Yang, Z. Luan, L. Gan, G. Yang and D. Qian, “Accelerating Sparse Cholesky Factorization on Sunway Manycore Architecture”, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 31, No. 7, pp. 1636–1650, 2020.

13. Le Fèvre, V., T. Usui and M. Casas, “A Selective Nesting Approach for the Sparse Multi-threaded Cholesky Factorization”, *2022 IEEE/ACM 7th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*, pp. 1–9, 2022.

14. Fèvre, V. L., T. Usui and M. Casas, “Optimization of the Sparse Multi-Threaded Cholesky Factorization for A64FX”, , 2022.

15. Liu, C., H. Yang, D. Wang, T. Wu, X. Wu, Z. Luo and X. Tang, “New Parallel Algorithms for Direct Solution of Large Sparse Matrix Equations”, *2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, pp. 354–357, 2018.

16. Chen, Y., T. A. Davis, W. W. Hager and S. Rajamanickam, “Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate”, *ACM Trans. Math. Softw.*, Vol. 35, No. 3, oct 2008, `https://doi.org/10.1145/1391989.1391995`.

17. OpenMP Architecture Review Board, *OpenMP Application Program Interface Version 5.2*, Tech. rep., OpenMP Architecture Review Board, November 2021, `https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf`.

18. Stathis, P., S. Vassiliadis and S. Cotofana, “D-SAB: A sparse matrix benchmark suite”, Vol. 2763, pp. 549–554, 09 2003.

19. Davis, T. A. and Y. Hu, “The university of Florida sparse matrix collection”, *ACM Trans. Math. Softw.*, Vol. 38, No. 1, dec 2011, `https://doi.org/10.1145/2049662.2049663`.

20. Tang, M., M. Gadou, S. Rennich, T. A. Davis and S. Ranka, “Optimized sparse Cholesky factorization on hybrid multicore architectures”, *Journal of Computational Science*, Vol. 26, pp. 246–253, 2018, `https://www.sciencedirect.com/science/article/pii/S1877750317312164`.