Term Project Report

Development of a C++ function for factorizing Sparse matrix by Multifrontal method

I) Theoretical basis:

1) Sparse Matrix:

In the field of numerical analysis, a sparse matrix is a matrix populated primarily with zeros as elements of the table. The matrix in *Figure 1* as an example. By contrast, if the number of nonzero elements in a matrix is relatively large, then it is commonly referred as a dense matrix.

1							
			1	1			
	1	1				1	
		1					1
	1			1			
		1	1				1
1		1		1			
			1				1

Figure 1. Sparse Matrix example.

2) Cholesky Factorization:

If A is a real, symmetric, and positive definite matrix, then it has a unique factorization, $A = L * L^T$, in which L is lower triangular with a positive diagonal [(5)].

Algorithm to partition matrices in $A = L * L^T$:

$$\begin{bmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} l_{11} & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} = \begin{bmatrix} l_{11}^2 & l_{11}L_{21}^T \\ l_{11}L_{21} & L_{21}L_{21}^T + L_{22}L_{22}^T \end{bmatrix} \quad (1)$$

Step 1. Determine l_{11} and L_{21}:

$$l_{11} = \sqrt{a_{11}}, L_{21} = \frac{1}{l_{11}} A_{21}$$

Step 2. Compute L_{22} from:

$$A_{22} - L_{21}L_{21}^T = L_{22}L_{22}^T$$

Step 3. Continue factor $L_{22}L_{22}^T$.

3) Cholesky Factorization with Sparse Matrix:

Let $n * n$ real, symmetric, and positive definite matrix A has the format:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

Let $A_n = A$ is respectively the matrix to be factorized of order n. When calculate the Cholesky factorization matrix of order $n-1$ (has $(n-1)*(n-1)$ elements), we have the general form for each element in A_{n-1}:

$$\begin{aligned} A_{n-1}(i,j) &= A_n(i+1,j+1) - \left[\frac{1}{L(1,1)}L(i+1,1)\right] \\ &\quad * \left[\frac{1}{L(1,1)}L(1,j+1)\right] \\ &= A_n(i+1,j+1) - \frac{A_n(i+1,1) * A_n(1,j+1)}{A_n(1,1)} \end{aligned} \tag{2}$$

Therefore, if $A_n(i,1) = A_n(1,i) = 0$, no need to calculate the row $i-1$ and column $i-1$ of the matrix A_{n-1}.

In addition, the form of each element in A_{n-2} is:

$$A_{n-2}(i,j) = A_{n-1}(i+1,j+1) - \left[\frac{1}{L(2,2)}L(i+2,2)\right] * \left[\frac{1}{L(2,2)}L(2,j+2)\right]$$

Hence

$$\begin{aligned} A_{n-2}(i,j) = A_n(i+2,j+2) &- \left[\frac{1}{L(1,1)}L(i+1,1)\right] * \left[\frac{1}{L(1,1)}L(1,j+1)\right] \\ &- \left[\frac{1}{L(2,2)}L(i+2,2)\right] * \left[\frac{1}{L(2,2)}L(2,j+2)\right] \end{aligned}$$

In general, we have:

$$\begin{aligned} A_{n-k}(i,j) &= A_n(i+k,j+k) \\ &\quad - \sum_{m=1}^{k} \frac{1}{L(m,m)^2} * L(i+m,m) * L(m,j+m) \end{aligned} \tag{3}$$

From Equation (3), we can modify the Cholesky Factorization algorithm. In each iteration, we remove all the row and column whose first element is zero. Furthermore, we only fill in the first row and column, other elements will be zero. Therefore $L_{22}L_{22}^T$ in Equation (1) is the Update Matrix for next iterations. This is the basis of Cholesky Factorization based on Multifrontal method.

4) Cholesky Factorization with Sparse Matrix based on Multifrontal method:

a) Elimination Tree:

Let A be an n-by-n sparse symmetric positive definite matrix with Cholesky factor L. The *elimination tree* of matrix A is defined to be the structure with n node $\{1, \dots, n\}$ such that node p is the parent if and only if [2]

$$p = \{ i > j \mid l_{ij} \neq 0 \}$$

We assume that A is irreducible.

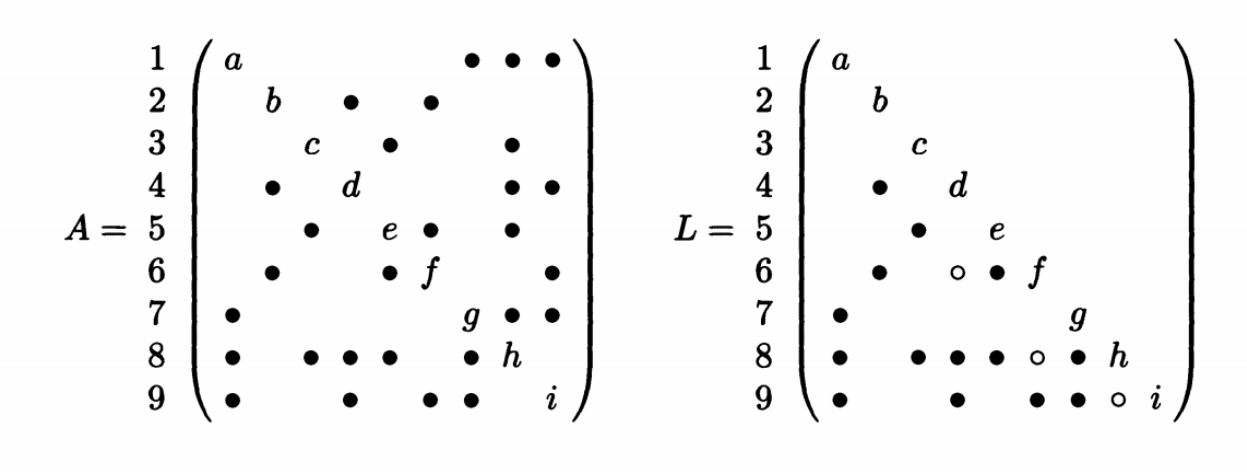

Figure 2. An example of sparse matrix and its Cholesky factor [2].

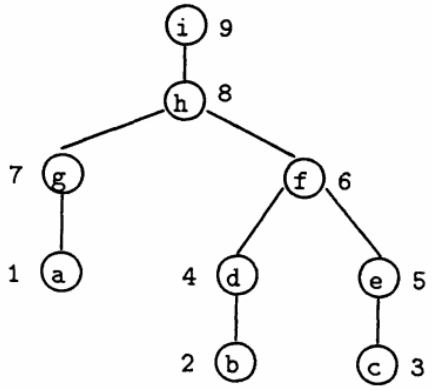

Figure 3. The elimination tree for matrix of Figure 2 [2].

The elimination tree will visualize the relationship between elements of matrix A such that if node j is children of k at any level, $l_{kj} \neq 0$. In some situation, matrix A will not construct an unified tree but a lot of discrete trees. Therefore, we must create some virtual data point (empty dot in Figure 2) to link the discrete trees into one unified tree.

Look at the elimination tree, we can know that if node j is children of k at any level, the update matrix of j-order Cholesky factorization will affect the k-order Cholesky factorization.

Example of construct elimination tree [3]:

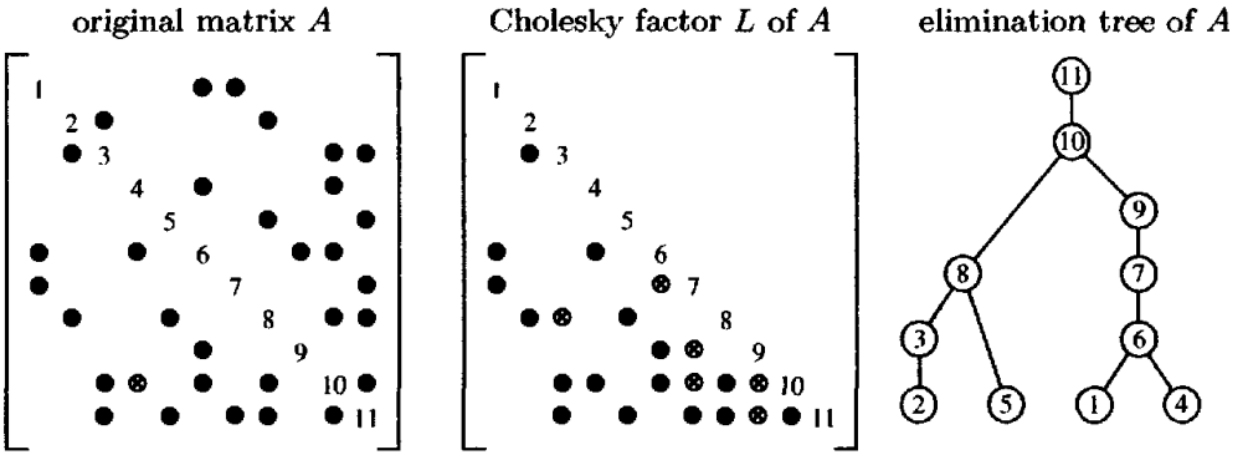

Figure 4. Example of Matrix and its Cholesky factor and Elimination tree.

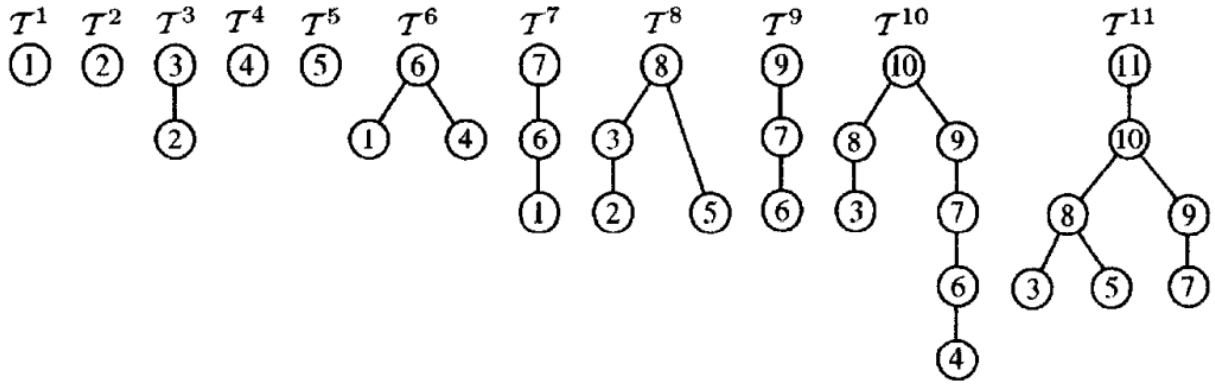

Figure 5. Iteration to construct elimination tree.

b) Shorten Matrix and Extend-Add operator [2]:

A shorten matrix is a matrix without unnecessary row or column. For example, we have a matrix M:

$$\begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

And its shorten matrix is:

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

$$row\_index = [1 \quad 2 \quad 4]$$

$$col\_index = [1 \quad 3 \quad 4]$$

That means:

index	1	3	4
1	M_{11}	M_{13}	M_{14}
2	M_{21}	M_{23}	M_{24}
4	M_{41}	M_{43}	M_{44}

Adding two shorten matrices is extend-add operation, using notation "⨢".

Example:

$$R = \begin{pmatrix} p & q \\ u & v \end{pmatrix}, \quad S = \begin{pmatrix} w & x \\ y & z \end{pmatrix}$$

$$R \uplus S = \begin{pmatrix} p & q & 0 \\ u & v & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} w & 0 & x \\ 0 & 0 & 0 \\ y & 0 & z \end{pmatrix} = \begin{pmatrix} p+w & q & x \\ u & v & 0 \\ y & 0 & z \end{pmatrix}.$$

c) Multifrontal Method algorithm:

```
for j = 1 to n do
    Let j < i1 < ··· < ir be the locations of nonzeros in L*j.
    Let c1, … , cs be the children of j in the elimination tree.
```

$\bar{U} = U_{c_1} \dotplus \ldots \dotplus U_{c_s}$ is the summation of update matrices which affect to j-order Cholesky factorization.

$$F_j = \begin{bmatrix} a_{j,j} & a_{j,i_1} & \cdots & a_{j,i_r} \\ a_{i_1,j} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_r,j} & 0 & \cdots & 0 \end{bmatrix} \dotplus \bar{U}$$

Factor F_j:

$$F_j = \begin{bmatrix} l_{j,j} & 0 \\ l_{i_1,j} & \\ \vdots & I \\ l_{i_r,j} & \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & U_j \end{bmatrix} * \begin{bmatrix} l_{j,j} & l_{j,i_1} & \ldots & l_{j,i_r} \\ 0 & & I & \end{bmatrix}$$

Store U_j which is the j-order update matrix.

end for

Algorithm of Cholesky factorization via frontal and update matrices [(2)].

II) Result:

I developed a C++ function to factorize large sparse matrix based on Multifrontal method. the result will be compared with the result which is computed by Matlab (using the function *chol*). In addition, I will develop a C++ program using ordinary Cholesky factorization or use the Code Generation feature of Matlab to generate C++ program of function *chol*, and then compare the computational time between two algorithms.

1) Elimination tree:

First, we use Figure 2 with the value of the dark dot is 1 and the value of another is 0.

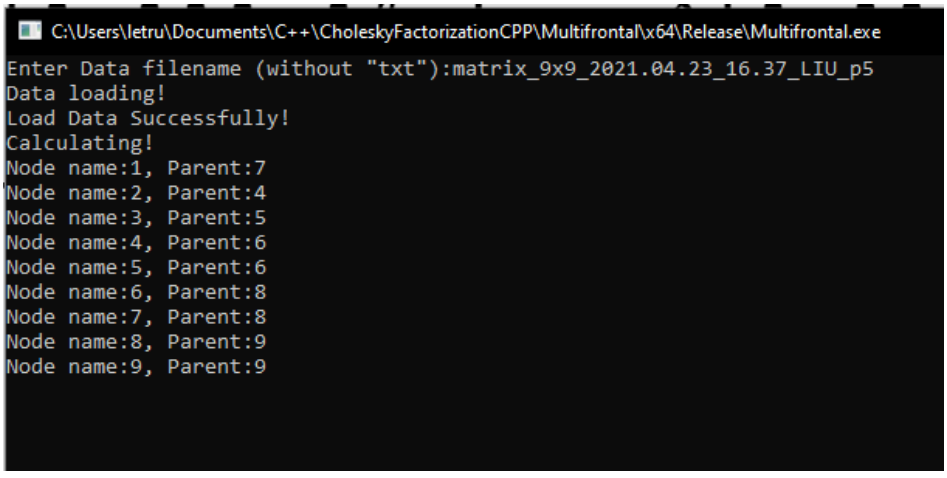

Figure 6. Result of elimination tree of Figure 2.

Next, we use Figure 4 to evaluate.

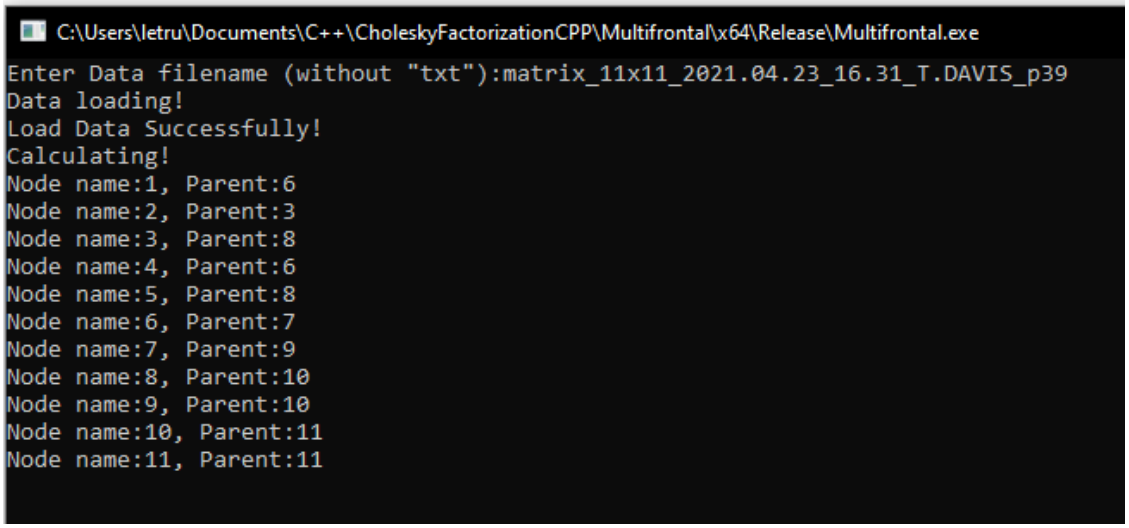

Figure 7. Result of elimination tree of Figure 4.

Conclusion, the result of elimination tree is correct.

2) Cholesky factor L:

The dataset will be get from *the Matrix Market (https://math.nist.gov /MatrixMarket/),* which is s visual repository of test data for use in comparative studies of algorithms for numerical linear algebra, featuring nearly 500 sparse matrices from a variety of applications, as well as matrix generation tools and services.

Testing computer specification:

- CPU: Intel Core i5-3360M @ 2.80GHz (2 cores 4 threads).
- Memory: 2x4GB DDR3L PC3-12800 (800MHz) Dual-Channel.
- Solid state drive.

a) Result comparation:

First dataset:

Max eigenvalue	9.6688590706791765E+03
Min eigenvalue	3.8956266582265278E-07
Condition number	2.481978E+10.
Size	5489 x 5489, 112505 entries
Type	real symmetric positive definite
Nonzeros elements	217669

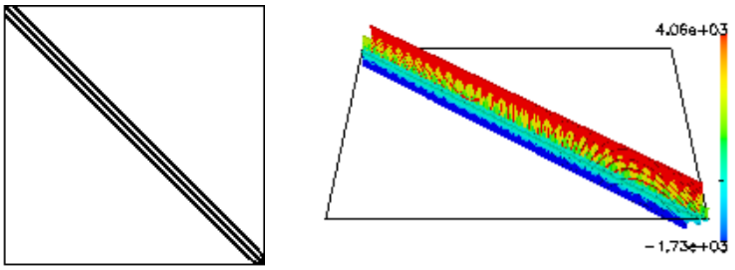

Figure 8. Plot of the first dataset.

Result:

```
Error Statistics:


   Error Max: 1.013409e-05
   Error Min: 0.000000e+00
   Error Mean: 4.890205e-11


     <1e-14 :      44518 value, absolute max value: 5.1849e+01, absolute
min value: 3.8723e-53
     <1e-13 :      95673 value, absolute max value: 5.1829e+01, absolute
min value: 3.8846e-53
     <1e-12 :     212328 value, absolute max value: 2.2177e+01, absolute
min value: 6.9871e-28
     <1e-11 :     385973 value, absolute max value: 2.8525e+00, absolute
min value: 2.5444e-29
     <1e-10 :     232529 value, absolute max value: 1.4790e+00, absolute
min value: 4.8326e-27
     <1e-09 :      28043 value, absolute max value: 3.8770e-01, absolute
min value: 5.3594e-25
     <1e-08 :       2778 value, absolute max value: 2.4731e-02, absolute
min value: 7.7802e-19
     <1e-07 :        295 value, absolute max value: 5.9287e-05, absolute
min value: 4.1170e-16
     <1e-06 :         27 value, absolute max value: 1.2794e-06, absolute
min value: 3.3417e-14
     <1e-05 :          3 value, absolute max value: 4.4391e-09, absolute
min value: 9.0391e-11
     <1e-04 :          1 value, absolute max value: 2.9691e-12, absolute
min value: 2.9691e-12
     <1e-03 :          0 value, absolute max value: -Inf, absolute min
value: Inf
     <1e-02 :          0 value, absolute max value: -Inf, absolute min
value: Inf
     <1e-01 :          0 value, absolute max value: -Inf, absolute min
value: Inf
     >1e-01 :          0 value, absolute max value: -Inf, absolute min
value: Inf, Infinity error: 0 number
```

Second dataset:

Max eigenvalue	4.6016534362482462E+03
Min eigenvalue	2.4268618877322254E-08
Condition number	1.896133E+11
Size	90449 x 90449
Type	real symmetric positive definite
Nonzero elements	4427725

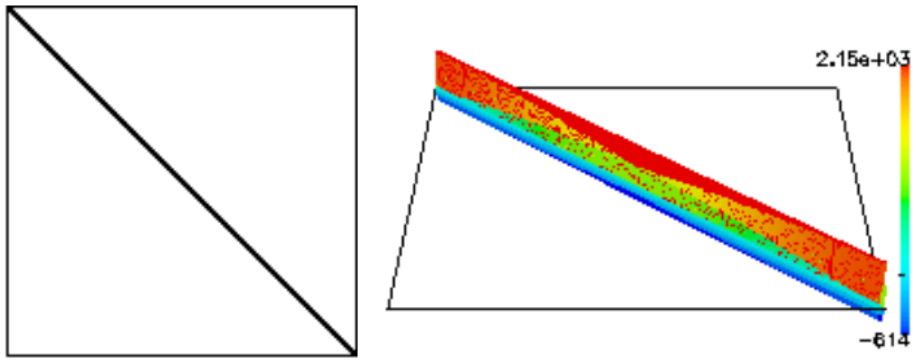

Figure 9. Plot of the second dataset.

Result:

```
Error Statistics:

   Error Max: 6.292069e-03
   Error Min: 0.000000e+00
   Error Mean: 3.751223e-10


     <1e-14 :    3984263 value, absolute max value: 4.4883e+01, absolute
min value: 1.2332e-57
     <1e-13 :    6356365 value, absolute max value: 3.5344e+01, absolute
min value: 1.7139e-58
     <1e-12 :   10734523 value, absolute max value: 1.5223e+01, absolute
min value: 5.5204e-57
     <1e-11 :   16978739 value, absolute max value: 1.4193e+00, absolute
min value: 8.4919e-14
     <1e-10 :   12772221 value, absolute max value: 4.6454e-01, absolute
min value: 4.1692e-14
     <1e-09 :    3612595 value, absolute max value: 1.2534e-02, absolute
min value: 2.4666e-15
     <1e-08 :     553855 value, absolute max value: 2.4730e-02, absolute
min value: 3.1408e-13
     <1e-07 :      57896 value, absolute max value: 6.7560e-05, absolute
min value: 8.4653e-14
     <1e-06 :       5732 value, absolute max value: 4.7104e-06, absolute
min value: 2.0642e-13
     <1e-05 :        580 value, absolute max value: 3.4476e-07, absolute
min value: 2.5447e-13
```

```
     <1e-04 :         65 value, absolute max value: 3.1013e-08, absolute
min value: 2.7878e-13
     <1e-03 :          8 value, absolute max value: 1.6812e-09, absolute
min value: 8.8056e-14
     <1e-02 :          2 value, absolute max value: 3.4537e-11, absolute
min value: 3.6072e-14
     <1e-01 :          0 value, absolute max value: -Inf, absolute min
value: Inf
     >1e-01 :          0 value, absolute max value: -Inf, absolute min
value: Inf, Infinity error: 0 number
```

Through 2 datasets above, we can conclude that the result generate by my own developed Cholesky factorization base on Multifrontal method function is very precisely. The relative errors become bigger (come to 1%) when the value of element is very small, therefore we can ignore these errors.

b) Computational time and memory usage comparation:

Using first dataset in section a) to comparation.

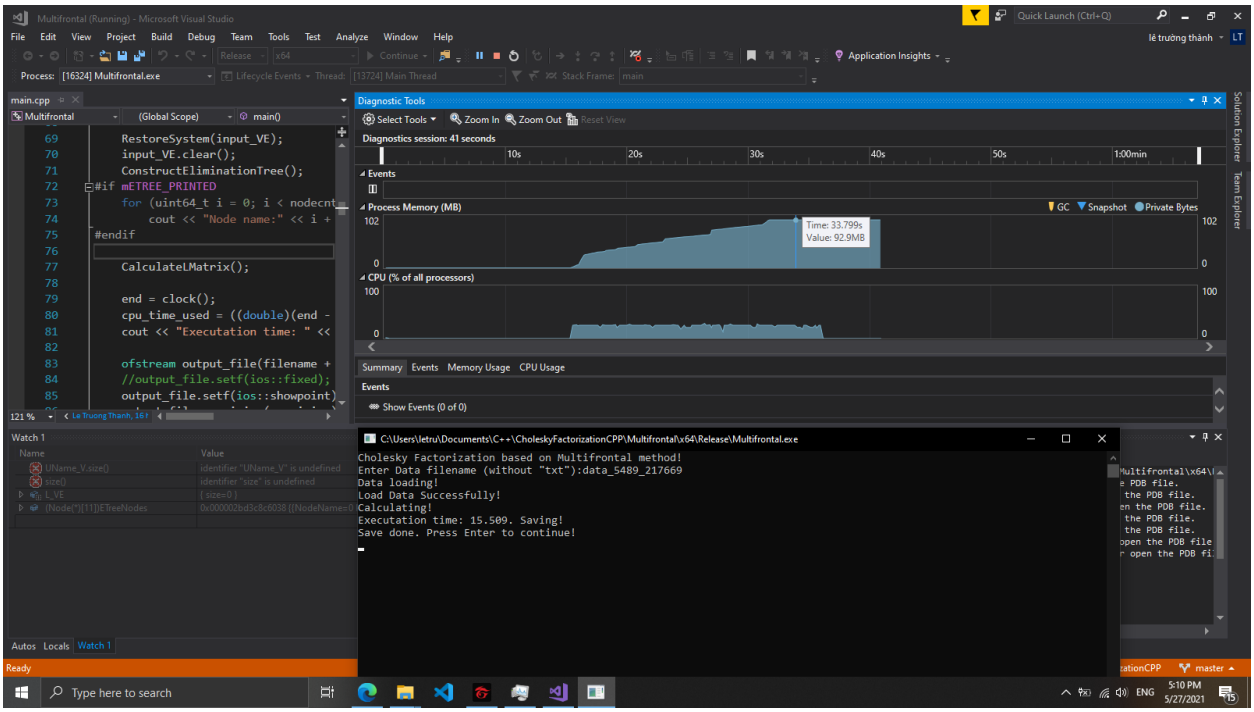

Figure 10. Computational time and memory usage of Multifrontal method (first dataset).

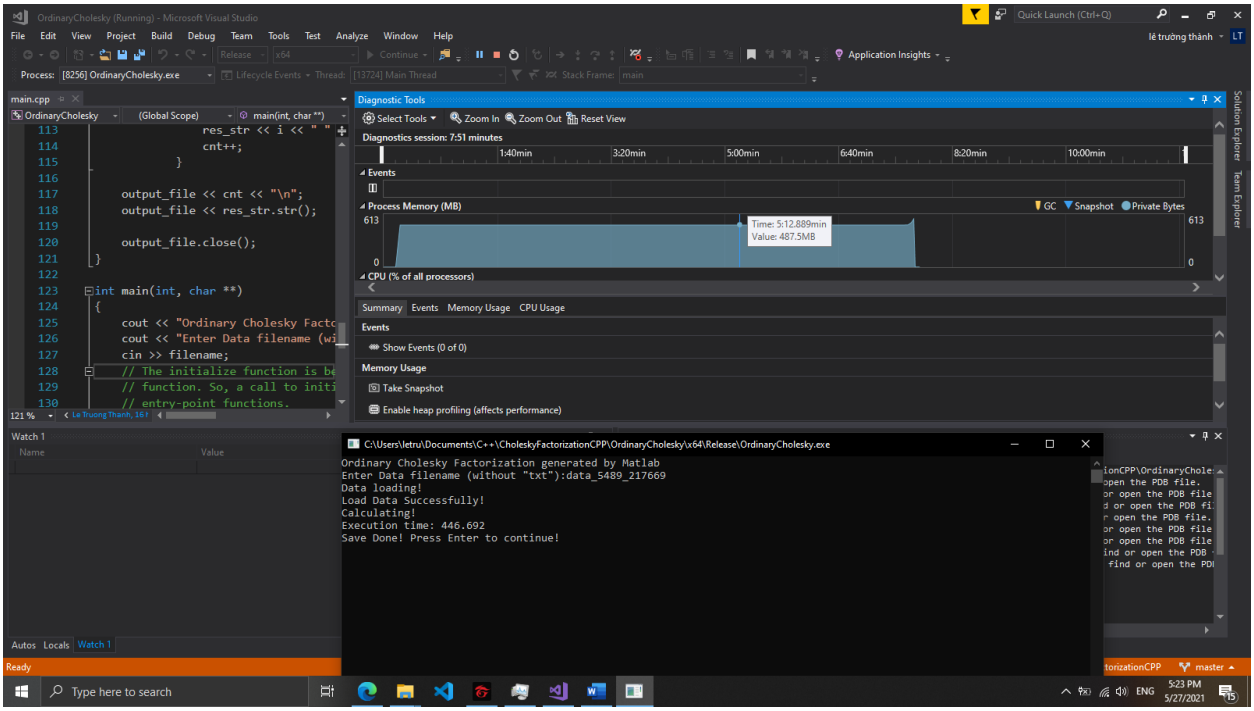

Figure 11. Computational time and memory usage of Ordinary Cholesky (first dataset).

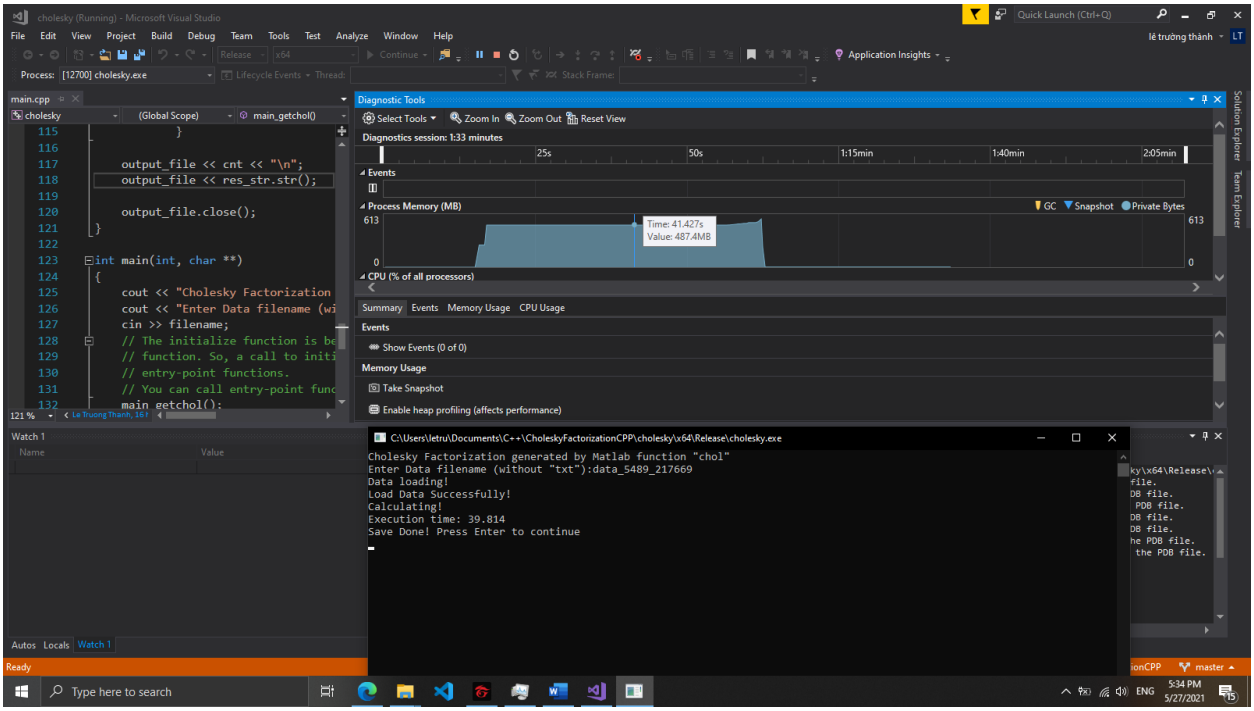

Figure 12. Computational time and memory usage of built-in Cholesky (first dataset).

Summary:

	Multifrontal	Ordinary Cholesky	Built-in Cholesky
Time(s)	15.509	446.692	39.814
Memory(MB)	92.9	487.5	487.4

Next, we will use the second dataset in section a) to evaluate. Because this dataset's size is 90449x90449 which require approximately 61GB (Ordinary Cholesky and built-in Cholesky require full matrix to be stored), we will run this dataset with Multifrontal method.

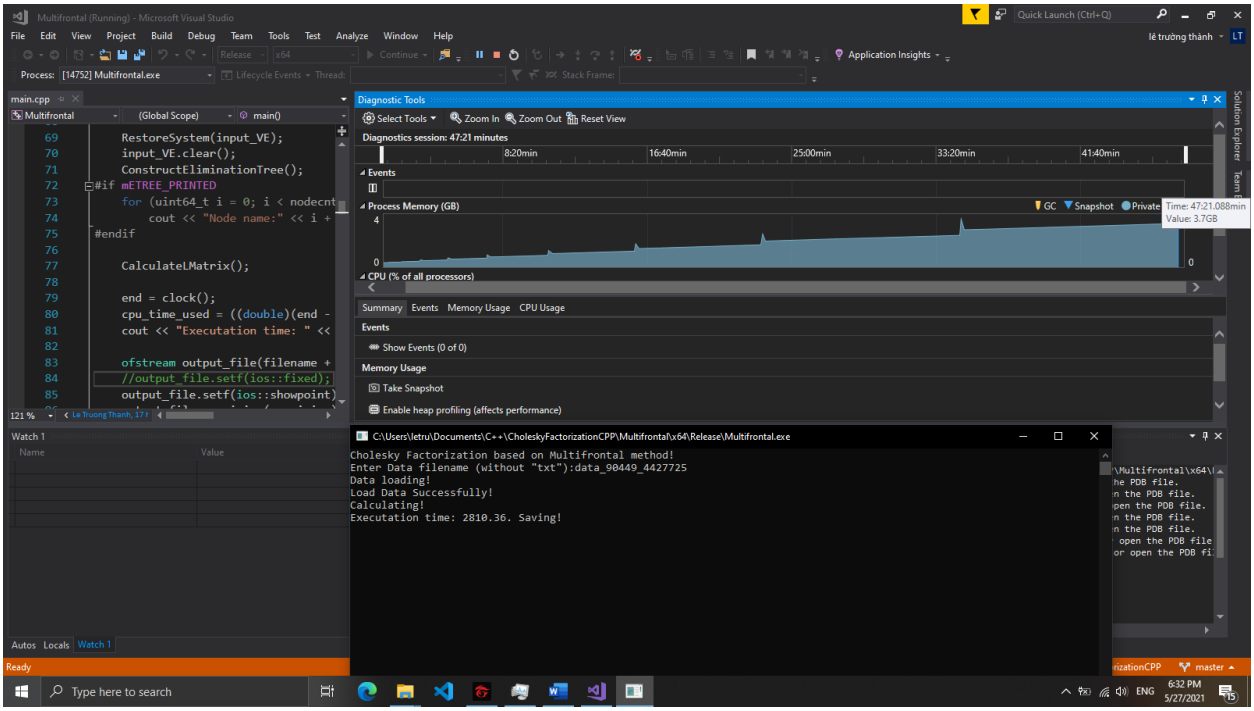

Figure 13. Computational time and memory usage of Multifrontal method (second dataset).

Summary:

- Computational time: 2810.36s.
- Memory usage: 3.7GB (peak 4.0GB).

In conclusion, the Cholesky factorization based on Multifrontal method is not only very precise but also faster and lighter than ordinary Cholesky factorization.

References:

(1) Yan, D., T. Wu, Y. Liu, and Y. Gao. *An Efficient Sparse-Dense Matrix Multiplication on a Multicore System*. 2017 IEEE 17th International Conference on Communication Technology (ICCT), 2017.

(2) Liu, Joseph W. H. "The Multifrontal Method for Sparse Matrix Solution: Theory and Practice." *SIAM Review* 34, no. 1 (1992/03/01 1992): 82-109. https://dx.doi.org/10.1137/1034004.

(3) Davis, Timothy A. "4. Cholesky Factorization." In *Direct Methods for Sparse Linear Systems*. Fundamentals of Algorithms: Society for Industrial and Applied Mathematics, 2006.

(4) Duff, I. S., A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. 2 ed. *Numerical Mathematics and Scientific Computation*. Oxford: Oxford University Press, 2017. https://oxford.universitypressscholarship.com/10.1093/acprof:oso/9780198508380.001.0001/acprof-9780198508380.

(5) Kincaid, David, and Ward Cheney. "Numerical Analysis : Mathematics of Scientific Computing, Third Edition." [In English]. (2009). https://public.ebookcentral.proquest.com/choice/publicfullrecord.aspx?p=4717637