Matrix Inversion Algorithms Using Intel TBB

Cholesky and LU decomposition

Ruger Stellberger
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: rstellberger@oakland.edu

***Abstract*— This project plans to use the Intel TBB library of threaded applications to simplify and streamline the calculations of a couple of types of matrix inversion algorithms. The two types will be the Cholesky and LU decompositions. The project will use one or more of the following TBB features that are parallel for, reduce, and pipeline. To confirm that the inversion of an input matrix was correct, a MATLAB script was used to confirm it.**

I. Introduction

Matrix inversion is fundamental in various scientific and engineering applications, from signal processing to optimization problems. As the scale and complexity of matrices increase, efficient algorithms become crucial for timely and accurate computations. This project embarks on leveraging the Intel Threading Building Blocks (TBB) library to enhance the performance of matrix inversion algorithms, specifically focusing on Cholesky and LU decompositions.

The primary objective of this project is to simplify and streamline the calculations involved in matrix inversion by utilizing parallel computing features provided by the Intel TBB library. The report will dive into the implementation of TBB's 'parallel_for' functionalities to optimize Cholesky and LU decomposition algorithms. These algorithms are chosen for their significance in numerical linear algebra and their wide-ranging applications in physics, statistics, and machine learning.

The motivation behind this project stems from the escalating size of matrices encountered in real-world applications. Traditional sequential algorithms for matrix inversion can be computationally intensive, especially when dealing with large datasets. By incorporating parallel computing through TBB, the project aims to significantly reduce computation times and improve overall efficiency.

II. Methodology

Intel TBB Parallelization Process

The parallelization of the inversion algorithms is the bread and butter of this project. Intel's TBB suite of functions used in C Plus Plus(CPP) programs provides an excellent way for programmers to create efficient and strong programs that make use of multithreading. TBB has many kinds of different parallelization functions that can be suited to many different kinds of tasks such as looping, pipelining, and reducing sequential tasks. TBB allows such tasks to be done more efficiently therefore reducing the time required to complete them by a significant margin. This project uses the TBB function 'parallel_for', this function allows for an iterative approach to multithreading. For this project 'parallel_for' was chosen because of the iterative nature of the algorithm. Matrix inversion typically revolves around mathematical calculations carried out on each cell of a matrix. In every case, the input matrix will need to be mapped and traversed completely to calculate every value of the output matrix. If these mathematical calculations can be parallelized while the matrix is traversed, major gains in computation time can be achieved.

Each inversion algorithm sees its function with a 'parallel_for' statement and the corresponding inversion calculations. The syntax of 'parallel_for' involves specifying a range of loop iterations and providing a lambda function representing the loop body. The use of the 'parallel_for' function is as follows. For dividing the range, the 'parallel_for' function divides the specified range [begin, end) into chunks, called blocked ranges. Each

```
void choleskyDecompositionTBB(double *matrix, int n) {
    tbb::parallel_for(tbb::blocked_range<int>(0, n), [=](const tbb::blocked_range<int>& r) {

void luDecompositionTBB(double *matrix, int rows, int cols) {
    tbb::parallel_for(tbb::blocked_range<int>(0, std::min(rows, cols)), [=](const tbb::blocked_range<int>& r)
```

blocked range is assigned to a different thread for parallel execution. In the lambda function execution, each thread executes the provided lambda function independently on its assigned blocked range. The synchronization lets TBB take care of the necessary synchronization mechanisms to ensure correct and safe parallel execution. The library manages thread creation, workload distribution, and synchronization, allowing developers to focus on the algorithm's logic. In the case of this project, the blocked range for the Cholesky decomposition represents the parallelization of the rows of the matrix being decomposed. from the first row, row 0, to the nth row depending on the size of the matrix. The LU decomposition works similarly in that the matrix is split up and parallelized although this time the blocked range starts at row 0 but then uses a function called min(rows, cols) instead of the variable n for the last row. This function takes the number of rows and columns and finds which is less than the other and then parallelizes based on whichever of the two parameters is lower for example a 4 by 3 matrix has 4 rows and 3 columns so TBB will parallelize the 3 columns

instead of the 4 rows. Inside the parallelized loop, the Cholesky and LU decomposition steps are applied to different portions of the matrix concurrently. The Cholesky algorithm only requires the rows because the decomposition of a Cholesky matrix can only be done with square matrices, meaning the rows and columns will always be equal in their amount. During lambda function execution, individual threads independently execute the provided lambda function on their respective assigned blocked ranges. The synchronization aspect is seamlessly managed by TBB, ensuring correct and safe parallel execution.

III. Experimental Setup

MATLAB Verification

The verification of matrix inversion plays a crucial role in ensuring the accuracy and reliability of the implemented algorithms. MATLAB serves as a powerful tool for this purpose, providing a platform to compare the results obtained from the parallelized Cholesky and LU decomposition algorithms with a reference.

1. Implementation of Matrix Inversion:
   - Utilize MATLAB's 'inv' function to compute the inverse of the input matrix.
2. Generation of Input Matrices:
   - Create various square matrices of different sizes suitable for Cholesky and LU decomposition.
3. Verification Procedure:
   - Compute the matrix inverse using the parallelized Cholesky and LU decomposition algorithms in C++.
   - Use MATLAB to compute the inverse of the same input matrix.
   - Compare the results for accuracy.
4. MATLAB Code

MATLAB verification code is provided:

```
function MatrixInversionAlgotithms()
    % Example matrices for Choledky and LU
    A = [93, 90, 74; 22, 82, 17; 17, 61, 50];
    B = [93, 90, 22; 90, 74, 82; 22, 82, 17];

    % LU Decomposition
    [L, U] = lu_decomposition(A);
    disp('LU Decomposition - L:');
    disp(L);
    disp('LU Decomposition - U:');
    disp(U);

    % Cholesky Decomposition
    L_cholesky = cholesky_decomposition(B);
    disp('Cholesky Decomposition:');
    disp(L_cholesky);
end

% LU Decomposition
function [L, U] = lu_decomposition(A)
    n = size(A, 1);
    L = eye(n);
    U = A;

    for k = 1:n-1
        for i = k+1:n
            L(i, k) = U(i, k) / U(k, k);
                U(i, k:n) = U(i, k:n) - L(i, k) * U(k,
k:n);
        end
    end
end

% Cholesky Decomposition
function L = cholesky_decomposition(B)
    n = size(B, 1);
    L = zeros(n, n);

    for i = 1:n
        for j = 1:i
            if i == j
                    L(i, j) = sqrt(B(i, j) - sum(L(i,
1:j-1).^2));
                if L(j, j) == 0
                    error('Cholesky decomposition
failed. Matrix is not positive definite.');
                end
            else
                L(i, j) = (B(i, j) - sum(L(i, 1:j-1) .*
L(j, 1:j-1))) / L(j, j);
            end
        end
    end
end
```

A test run of the program was ran with a 3x3 matrix with a randomly generated matrix from the CPP program. These matrices were then input into the MATLAB script to be run to confirm that the CPP program was working correctly.

IV. Results

The application of Intel TBB in parallelizing matrix inversion algorithms yields compelling results in terms of computational efficiency. In this section, we present a comprehensive analysis of the performance improvements achieved through parallelization, comparing execution times and scalability metrics between the sequential and parallel implementations. Our investigation focuses on the Cholesky and LU decomposition algorithms, demonstrating the impact of multithreading on the inversion process. For the data representation of the results, a negative value in the difference represents a longer time to compute for TBB whereas a positive is a lower time to compute.

First, we'll look at the computation time results for the Cholesky decomposition based on Figure 1, showcasing the time to sequentially compute the decomposition, as well as the comparison with TBB in terms of their difference in

Matrix Size
5
10
50
100
200
500

computation time. The table tests a few different sizes of matrices to show a significant difference in the resulting times. There is also a graph, Figure 2, that shows a visual representation of the computation time gains we see while using TBB to parallelize the algorithm.

			Cholesky decomposition(Positive difference is TBB improvment)					
Matrix Size -->	5x5	10x10	50x50	100x100	200x200	500x500	750x750	1000x1000
Sequential(us)	41.3	49	1675.6	12117.6	104451	1879731	5885838.6	14406541.6
TBB(us)	3686.5	3509.7	5035.1	15728	107175.2	1371099.8	3840201	9089741.6
Difference	-3645.2	-3460.7	-3359.5	-3610.4	-2724.2	508631.2	2045637.6	5316800

Figure 1

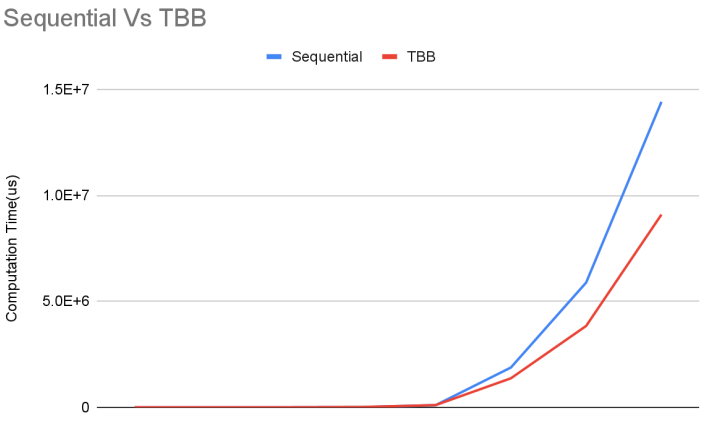

Figure 2

Looking at Figures 1 and 2, we can see that in small sizes of matrices, TBB has an inverse effect of what we intended to show. There isn't much need for parallelization for small data sets due to how simple they tend to be. Up until the 200x200 matrix we do not see any type of performance gain while using TBB. I would like to point out though that as we increase the matrix sizes we can notice that while TBB does not see performance uplift in the beginning, TBB does get closer and closer to matching the sequential algorithms performance. at 200x200 while there is still a TBB net loss, the computation times are nearly identical. Looking at the 500x500 matrix, there is a definite change in outcome for TBB. There is a very significant performance uplift and this translates even more to the matrices that are even bigger than that. Looking solely at Figure 2 it can be seen that TBB nearly cuts computation time in half when the matrices become larger and larger. The results from the Cholesky algorithm show that there is a definite advantage to parallelizing our algorithm when compared to a sequential approach.

Secondly, we'll look at the results of the LU decomposition algorithm. Below are another 3 figures, Figures 3, 4, and 5, where figures 4 and 5 are set up very similar to that of figures 1 and 2 although with the computation times from running the LU algorithm.

	LU decompostion(Positive difference is TBB improvment)					
Matrix Size	5	10	50	100	200	500
5	-26.6	-23	-55.3	-187.4	-154.3	18.7
10	-48	-53.6	-140.7	-51.3	-410.4	89
50	-44.7	-50	-406.8	1443.3	3650.6	4161.6
100	-77.7	-65.7	1296.7	3122.6	5986.2	24546.3
200	-24.7	77.7	2654	6838.8	25436.6	74089.6
500	-73.5	142.6	9732.4	16380.7	47813.5	682935.5

Figure 3

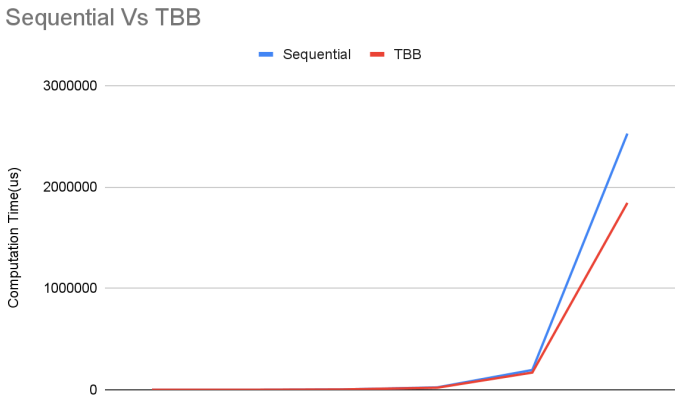

Figure 4

		LU decompostion(Positive difference is TBB improvment)				
	5x5	10x10	50x50	100x100	200x200	500x500
Sequential(us)	3.4	23.6	2755.2	23505.6	194946.8	2525344.5
TBB(us)	30	77.3	3162	20383	169510.2	1842409
Difference	-26.6	-53.7	-406.8	3122.6	25436.6	682935.5
		Computation Times(us) measured over average of 10 runs				

Figure 5

Here, looking at figure 3 we have a large table of matrix sizes that were tested from a 5x5 matrix to a 500x500 and everything in between. For LU decomposition there is no specific requirement for the matrix to be square, although it is typically used in that fashion which we will talk about in Figure 5, I did this to show a broader spectrum of potential TBB improvement scenarios. Figure 3 however, shows a similar story to Figure 1 and the Cholesky decomposition. At small matrix sizes, we don't see any kind of gain by using TBB. We do not see any TBB gain until around a matrix size of 50x100 which is significantly smaller than the 200x200 for the Cholesky. This may be due to the simpler mathematical calculations that are used in the LU decomposition. As we move to matrices higher than 50x100 they all show an improvement in comutation time for TBB, with the 500x500 showing a difference close to 0.75 seconds. Again I have included a visual representation of of these results with a graph seen in Figure 4. The graph clearly shows how TBB makes a large difference in computation times as we keep increasing the matrix size. Moving over to Figure 5 where the table is set up the same as Figure 1 to show the results using just square matrices we can see the raw change in computation times for the sequential and TBB. Again there is a tightening of the

difference in times between the two computation types as we move from a lower-sized matrix to a higher one. Although I did not test with matrices as large as 750x750 for the LU results, an examination of the graph suggests that TBB could approach cutting computation times in half.

Conclusions

Key Take-Away Points:

1. Significant Performance Gains: The results of applying TBB to the Cholesky decomposition showcased a notable performance uplift, particularly for larger matrix sizes. Although TBB did not exhibit substantial gains for smaller matrices, it nearly halved computation times for matrices of 500x500 and beyond.
2. Algorithm-Specific Behavior: The LU decomposition, with its different mathematical calculations, demonstrated a distinct behavior compared to Cholesky. TBB gains were observed for matrices larger than 50x100, emphasizing the impact of algorithm intricacies on parallelization benefits.
3. Verification through MATLAB: The use of MATLAB for result verification ensured the accuracy of the parallelized algorithms. This step is crucial in real-world applications where precision is paramount.

Further Work:

4. Exploration of Larger Matrices: While the project demonstrated TBB's effectiveness for matrices up to 500x500, further exploration with even larger matrices, such as 750x750, could provide insights into the scalability of the parallelization approach.
5. Algorithm-Specific Optimization: Tailoring TBB parallelization techniques to the specific characteristics of Cholesky and LU decomposition algorithms could lead to further improvements. Fine-tuning parameters and exploring additional TBB functionalities could enhance performance.
6. Generalization to Other Algorithms: Extending the application of TBB to other matrix-related algorithms could broaden the impact of parallelization. Investigating its effectiveness for different types of linear algebra operations is an avenue for future exploration.

Remaining Issues:

7. Performance Thresholds: Identifying the threshold at which TBB becomes advantageous for specific matrix sizes and algorithms is a nuanced challenge. Determining optimal conditions for parallelization remains an area for refinement.

Algorithm Complexity:

8. The complexity of matrix inversion algorithms influences TBB's effectiveness. A deeper understanding of how algorithm intricacies impact parallelization benefits could guide future optimizations.

In conclusion, this project successfully demonstrated the efficiency gains achievable through TBB parallelization for Cholesky and LU matrix inversion algorithms. The results highlight the algorithm-specific nature of TBB benefits and the importance of accurate verification. As we look to the future, there is room for further exploration into larger matrices, algorithm-specific optimizations, and the application of parallelization techniques to a broader spectrum of linear algebra operations. Overall, this project contributes to the ongoing pursuit of enhancing computational efficiency in numerical linear algebra.

References

[1] "L U decomposition of a system of linear equations," GeeksforGeeks, https://www.geeksforgeeks.org/l-u-decomposition-system-linear-equations/ (accessed Dec. 16, 2023).

[2] "Cholesky decomposition : Matrix decomposition," GeeksforGeeks, https://www.geeksforgeeks.org/cholesky-decomposition-matrix-decomposition/ (accessed Dec. 16, 2023).

[3] By, "Getting started with Intel® threading building blocks (Intel® TBB)," Intel, https://www.intel.com/content/www/us/en/developer/articles/guide/get-started-with-tbb.html (accessed Dec. 16, 2023).

[4] "Intel® oneapi threading building blocks," Intel® oneAPI Threading Building Blocks, https://www.intel.com/content/www/us/en/developer/tools/oneapi/onetbb.html#gs.1p3ze8 (accessed Dec. 16, 2023).

[5] M. Voss, *Pro Tbb*. Apress, 2019.

[6] [1] D. LLamocca, "Reconfigurable Computer Research Laboratory," High-performance embedded programming with the Intel Atom Platform, https://www.secs.oakland.edu/~llamocca/emb_intel.html (accessed Dec. 16, 2023).