sameera Oblig 2 report

Introduction

This report is for oblig 2 “Parallelization of Matrix Multiplication” in IN3030 2024.
I have tested my code on a laptop with a 2,3 GHz to core Intel Core i5, but it has Hyper-Threading, so I have 4 threads.

Sequential Matrix Multiplication

For the Sequential Matrix Multiplication, I have made 4 methods. One each for three of the multiplication types and one that transposes a matrix.

The classic method multiplies each row of matrix A by each column of matrix B in a nested loop structure. This is the traditional method and follows the definition of matrix multiplication.

The B Transposed method transposes matrix B before performing multiplication with matrix A. It then multiplies the rows of A with these rearranged rows of B (originally B's columns) to produce the resulting matrix.

The A Transposed method transposes matrix A before performing multiplication with matrix B. This means that we multiply columns in the transposed matrix A with the columns of matrix B.

The transpose method is for converting rows of a given matrix into columns and the other way. It's used by the other two methods to transpose matrices A or B as needed before multiplication.

Parallel Matrix Multiplication

In the parallel matrix multiplication approach, I adapted the same methods used in the sequential part but tweaked them for concurrent execution. The goal was to split the workload evenly across multiple threads, corresponding to the number of available cores, to speed up the computation.

For each method, the matrix multiplication task is divided by the number of processing cores. Each thread calculates a specific section of the result matrix.
Once all threads are started, they run their computations in parallel. After finishing their tasks, I used the join method to ensure that the main program waits for all threads to complete before proceeding. The transposition of matrixes is done before the multiplication task is divided among the threads.

Measurements

Table for time taken in ms:

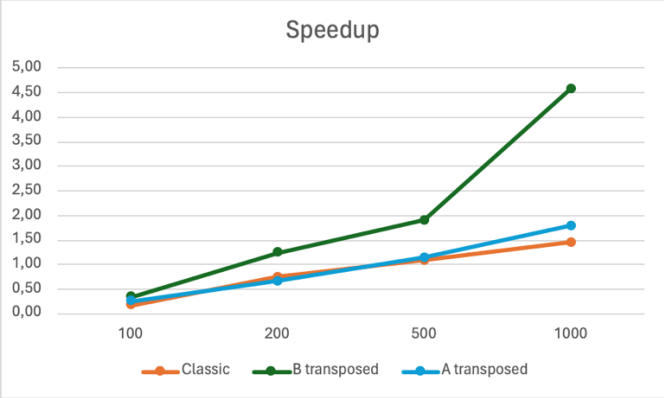
	Sequential			Parallel		
n	Classic	B transposed	A transposed	Classic	B transposed	A transposed
100	6	7	7	35	21	28
200	27	26	36	36	21	54
500	265	243	665	243	128	582
1000	5525	2023	15256	3818	441	8538

Speedup from the Sequential to the parallel version:

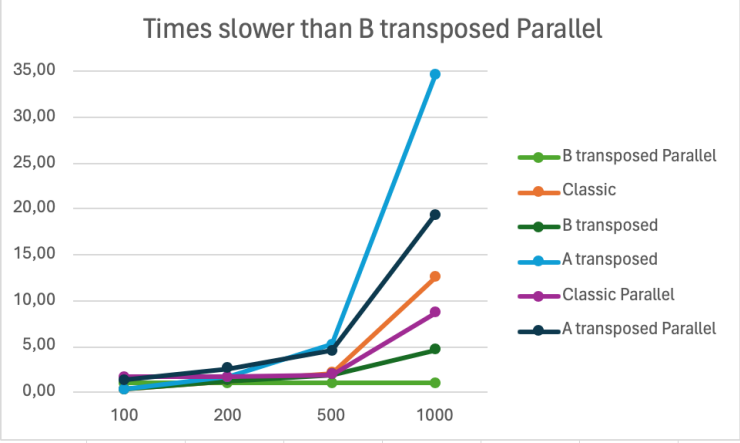
Speedup (Sequential/Parallel)			
n	Classic	B transposed	A transposed
100	0,17	0,33	0,25
200	0,75	1,24	0,67
500	1,09	1,90	1,14
1000	1,45	4,59	1,79

From the data we can clearly see that the sequential multiplication is faster for when n is under 200, except B transposed where the parallel is faster. As the size of the matrix increases, parallel methods outpace their sequential equivalents across the board. The most significant improvement in performance is seen with the parallel B transposed method for n=1000, achieving a speedup of 4.59.

Graph showing speedup:



Graph showing times slower than the fasted method (B transposed parallel):



From these graphs we can see that the sequential multiplication gets very slow when n increases. We also see that B transposed parallel is much faster than every other way, it combines the parallelization with the best cache friendly method. We can see that it is around 35 times faster than A transposed and 20 times faster than A transposed parallel.

User guide

To execute the program, input the size of the matrices 'n' as follows:

java Oblig2 n

The program executes all six methods of matrix multiplication and reports the time taken in milliseconds. It utilizes all available cores by default.

Conclusion

The parallel matrix multiplication methods, especially for B transposed, have shown considerable speedup, particularly for larger matrices. This performance gain confirms the advantages of parallel computation in numerical tasks and highlights the efficiency of optimized cache access in matrix operations.