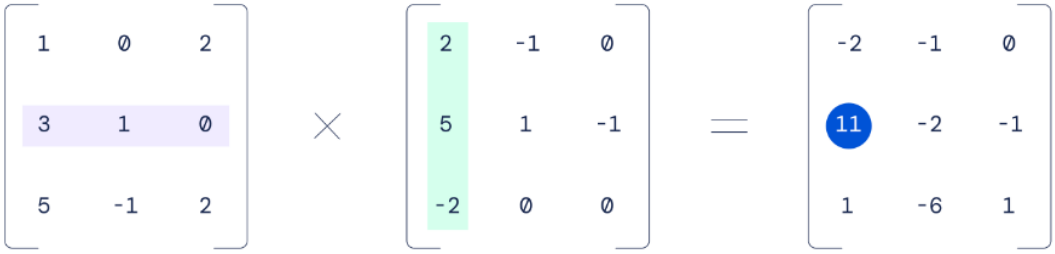
Kennesaw State University

Parallel and Distributed Computing

Project - OpenMP

Instructor: Kun Suo
Points Possible: 100



The following code implements multiplication of two matrices. The order of the matrix is 2048. Function matrixInit() initializes a double type value for all elements in the matrix. Function matrixMulti() performs the multipy calculation. However, the program executes in the sequential implementation.

https://github.com/kevinsuo/CS4504/blob/master/Matrix_Multiple_Sample.c

---

```
#include <stdio.h>
#include <omp.h>
#include <time.h>
#include <stdlib.h>

#define N 2048
#define FactorIntToDouble 1.1;

double firstMatrix [N] [N] = {0.0};
double secondMatrix [N] [N] = {0.0};
double matrixMultiResult [N] [N] = {0.0};


void matrixMulti()
{
    for(int row = 0 ; row < N ; row++){
        for(int col = 0; col < N ; col++){
            double resultValue = 0;
```

```
            for(int transNumber = 0 ; transNumber < N ; transNumber++) {
                resultValue += firstMatrix [row] [transNumber] *
secondMatrix [transNumber] [col] ;
            }

            matrixMultiResult [row] [col] = resultValue;
        }
    }
}


void matrixInit()
{
    for(int row = 0 ; row < N ; row++ ) {
        for(int col = 0 ; col < N ;col++){
            srand(row+col);
            firstMatrix [row] [col] = ( rand() % 10 ) * FactorIntToDouble;
            secondMatrix [row] [col] = ( rand() % 10 ) *
FactorIntToDouble;
        }
    }
}


int main()
{
    matrixInit();

    clock_t t1 = clock();
    matrixMulti();
    clock_t t2 = clock();
    printf("time: %ld", t2-t1);


    //double t1 = omp_get_wtime();
    //matrixMulti();
    //double t2 = omp_get_wtime();
    //printf("serial   time: %3f\n", ((double)t2 - t1) / CLOCKS_PER_SEC *
1000000.0);
    return 0;
}
```

Note: You have two ways to measure the execution time.

(1) clock() records the number of ticks of the CPU. When multiple processes are calculated simultaneously in parallel, the number of CPU ticks increases multiplied. You should divide the clock() by N if you use N processes.
(2) omp_get_wtime() returns the timestamp, which is irrelevant to the number of processes.

Task 1 (50 points):

Write a parallel program using OpenMP based on this sequential solution.

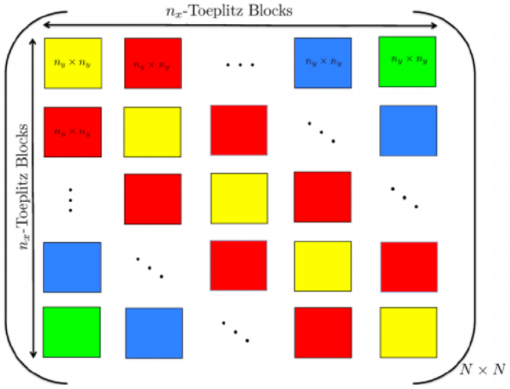
To compile the program with OpenMP, use:
$ gcc program.c -o program.o -fopenmp

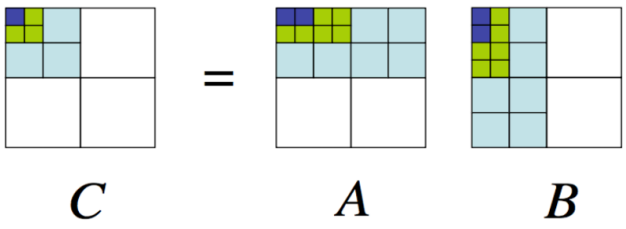
Please write a one-page report (with number and figures), which compares the execution time of sequential solution and parallel solution under different matrix orders (value of N). To get stable values, try to get the average time for each execution.

Order of Matrix	1024	2048	4096
Sequential Time			
Parallel Time			
Speedup			

Task 2 (50 points):

In order to further improve the performance, the matrix can be divided into blocks, and a part of the matrix can be calculated at one time. Under such the implementation, the CPU can move a part of the matrix data into the cache, which can improve the cache hit rate and the program performance.

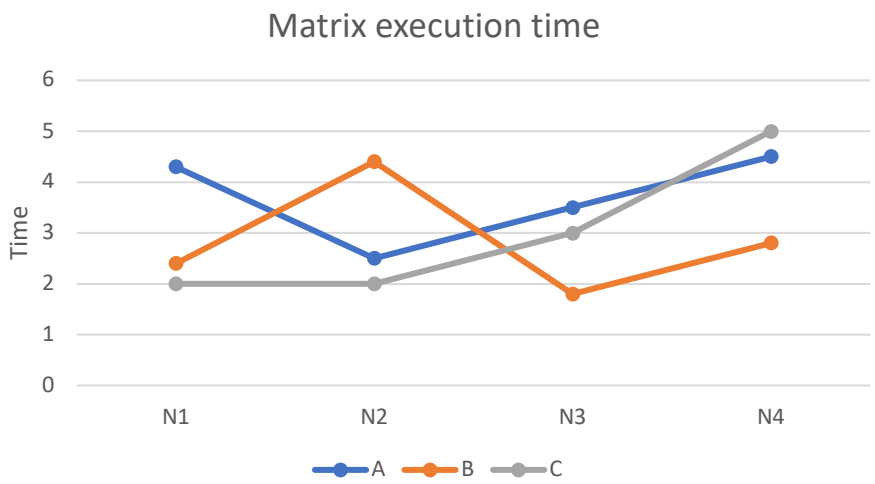




Please write a block-optimized matrix multiplication program and use OpenMP to parallel its execution. Compare the program execution time with that in Task 1 and write another report with data and figures. To get stable values, try to get the average time for each execution.

You can use the following template:
https://github.com/kevinsuo/CS4504/blob/main/OpenMP_block_optimized_template.c

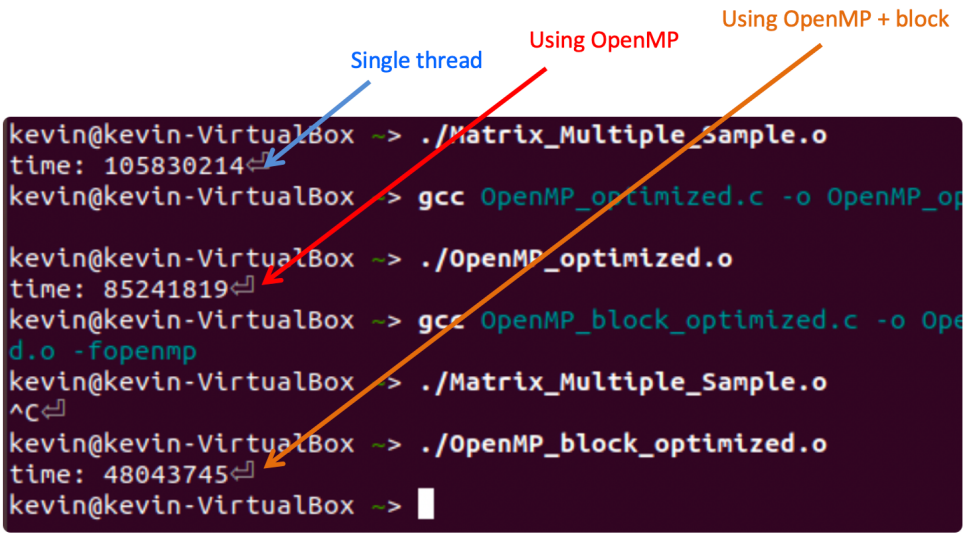
Order of Matrix	1024	2048	4096
Block-optimized Sequential Time			
Block-optimized Parallel Time			
Speedup			



Expected Output

Normally, for a certain size of the matrix, the execution time of a single-thread program (ST), OpenMP-optimized program (OMP), and OpenMP with block-optimized program (OMP-b) should be:

ST > OMP > OMP-b

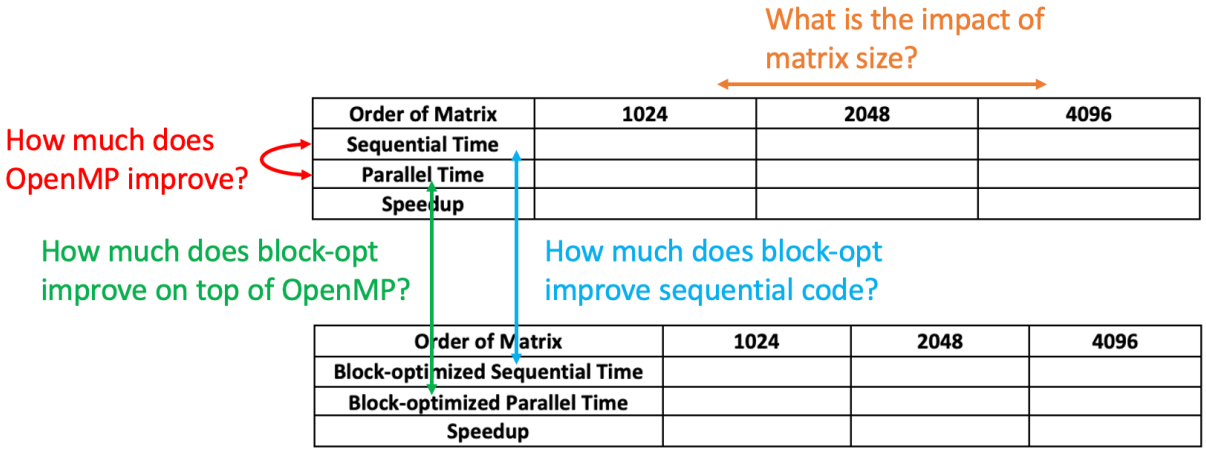


Submitting Assignment

Submit your assignment file through D2L using the appropriate link.
The submission must include the ***source code***, and ***a report describe your code logic***. ***Output screenshot of your code*** should be included in the report.

Reference

When you compare the results in above Task 1 and Task 2, here are some aspects you can compare and analyze:



Order of Matrix	1024	2048	4096
Sequential Time			
Parallel Time			
Speedup			

Order of Matrix	1024	2048	4096
Block-optimized Sequential Time			
Block-optimized Parallel Time			
Speedup			