Extra Credit Assignment (Version 2)

Due Date:	Nov. 15, 2024, 11:59PM (Pacific Time)
Assignment Delivery Method:	Electronic Submission to Blackboard (CMPT-2395 “Assignments” Tab)
Late Policy:	No Credit (Grade of Zero)
Files to be Submitted:	“**project.c**” and “**report.pdf**”

NOTICE

Do NOT upload this document or its contents to any third-party, unauthorized websites including ChatGPT, Course Hero, Discord, and Chegg. Doing so would violate the Douglas College Academic Integrity Policy. Any unauthorized sharing of this document or its contents will be investigated using the IP/MAC address of the account holder on Blackboard. Uploading a solution for this assignment to any unauthorized website is also a violation of the Academic Integrity policy and will be investigated.

Description

Consider the multiplication of a pair of $n \times n$ matrices. For instance, if $n = 2$, then we could write

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

where

$$\begin{aligned} c_{11} &= a_{11}b_{11} + a_{12}b_{21} \\ c_{12} &= a_{11}b_{12} + a_{12}b_{22} \\ c_{21} &= a_{21}b_{11} + a_{22}b_{21} \\ c_{22} &= a_{21}b_{12} + a_{22}b_{22} \end{aligned}$$

We can implement a matrix multiplication function using three nested for-loop structures. Permutating the loops will allow us to create six equivalent implementations, as shown below.

```c
void ver_ijk(matrix A, matrix B, matrix C, int n)
{
    int i, j, k;
    double sum;

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++) {
            sum = 0.0;
            for (k = 0; k < n; k++)
                sum += A[i][k] * B[k][j];
            C[i][j] += sum;
        }

}

void ver_jik(matrix A, matrix B, matrix C, int n)
{
    int i, j, k;
    double sum;

    for (j = 0; j < n; j++)
        for (i = 0; i < n; i++) {
            sum = 0.0;
            for (k = 0; k < n; k++)
                sum += A[i][k] * B[k][j];
            C[i][j] += sum;
        }
}

void ver_ikj(matrix A, matrix B, matrix C, int n)
{
    int i, j, k;
    double temp;

    for (i = 0; i < n; i++)
        for (k = 0; k < n; k++) {
            temp = A[i][k];
            for (j = 0; j < n; j++)
                C[i][j] += temp * B[k][j];
        }
}

void ver_kij(matrix A, matrix B, matrix C, int n)
{
    int i, j, k;
    double temp;

    for (k = 0; k < n; k++)
        for (i = 0; i < n; i++) {
            temp = A[i][k];
            for (j = 0; j < n; j++)
                C[i][j] += temp * B[k][j];
        }
}
```

```c
void ver_kji(matrix A, matrix B, matrix C, int n)
{
    int i, j, k;
    double temp;

    for (k = 0; k < n; k++)
        for (j = 0; j < n; j++) {
            temp = B[k][j];
            for (i = 0; i < n; i++)
                C[i][j] += A[i][k] * temp;
        }
}

void ver_jki(matrix A, matrix B, matrix C, int n)
{
    int i, j, k;
    double temp;

    for (j = 0; j < n; j++)
        for (k = 0; k < n; k++) {
            temp = B[k][j];
            for (i = 0; i < n; i++)
                C[i][j] += A[i][k] * temp;
        }
}
```

You first need to convince yourself that all six versions are indeed equivalent. You may do so by following a 2×2 multiplication example.

Then, your next task is to implement the six functions shown above in a C program, so you could measure the time it takes for each version to perform the matrix multiplication given a specific value for matrix size n. This will allow you to plot the amount of time it takes in seconds for each version to complete the computation as a function of n. Our objective is to identify the version that takes the least amount of time for different values of n.

Your submission must meet the following requirements for full credit:

1. It must be able to be executed using the Cygwin64 command line. The executable object must accept two input arguments. One that indicates which version of the matrix multiplication the user would like to test, the other n.

2. Your program must use dynamic memory allocation and deallocation correctly for the purpose of creating and discarding all matrices (because if you use static memory allocation, your program will experience stack overflow for very large values of n).

3. If the user does not provide the correct number of arguments to the executable object, your program must show its correct usage.

4. All dynamically allocated arrays must be one-dimensional.

5. Your plot must show a range of values for n between 300 and 10000 with a reasonable increment in between.

6. Your time measurements should come from the **`time`** tool available in Cygwin64.

7. Your time measurements must exclude the time spent on filling the matrices.

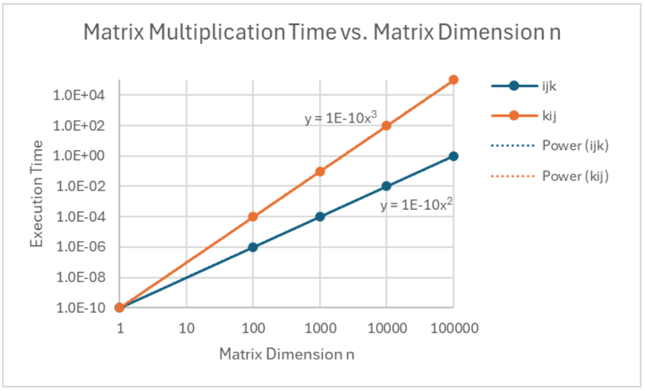
8. Your submission must clearly identify the specifications (i.e., clocking frequency, number of cores, and L1, L2, and L3 cache capacities) of the CPU that the experiment was carried out on.

In addition to your C program, you must also submit short report (i.e., no more than 2 pages long) that shows the time versus size plot and provides an explanation of the results. This report must meet the following requirements:

1. It must include a plot of the time it takes for each function version to perform the matrix multiplication given a specific value for matrix size n.

2. Both the x- and y-axes of this plot must have logarithmic scale (not linear).

3. Each curve on your plot must include a linear fit in the following form: $y = ax^b$, where both a and b are constants. Your plot should look similar to the following hypothetical example (but yours would obviously show six curves instead of two because there are six different versions of the same function):



4. Your report must relate the results shown on the plot to the qualitative miss rate for each version of the function.

All work submitted for evaluation must consist of the student's work only and may not be copied in whole or in part from any print or electronic resource on the web or from any other student. All submissions must comply with Douglas College Academic Integrity Policy.