ECE 4122/6122 Lab 2: OpenMP & Matrix Multiplication

(100 pts)

Due: Monday Oct 10th, 2022 by 11:59 PM

Problem 1: Matrix Multiplication (90 pts)

Write a C++ application that can read in two matrices (A & B) from a text file using a command line argument to specify the file path and name. The input file format is shown below. A C++ program is included to generate random sample input data files.

4 2
1.35477 8.35009
9.68868 2.21034
3.08167 5.47221
1.88382 9.92881
2 3
9.96461 9.67695 7.25839
9.8111 1.09862 7.98106

Your application must work correctly when compiled in both OpenMP mode and non-OpenMP mode. Your application needs to multiple the two matrices together and output the multiplied matrix to a text file called **MatrixOut.txt** using the same format as the input file. Use the standard matrix multiplication technique in your application.

(https://en.wikipedia.org/wiki/Matrix_multiplication)

If $\mathbf{A}$ is an $m \times n$ matrix and $\mathbf{B}$ is an $n \times p$ matrix,

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

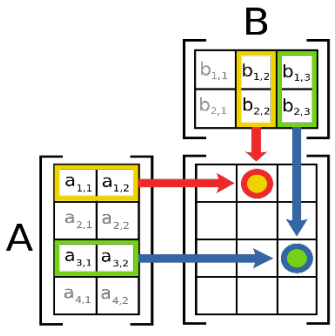
the *matrix product* $\mathbf{C} = \mathbf{AB}$ (denoted without multiplication signs or dots) is defined to be the $m \times p$ matrix

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix}$$

such that

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^{n} a_{ik}b_{kj},$$

for $i = 1, \ldots, m$ and $j = 1, \ldots, p$.



***** Your application will also be graded on speed when compiled using OpenMP flag.***

Problem 1 Turn-In Instructions:

Place all the files (*.h, *.cpp) used in your solution into a zip file called Lab2Problem1.zip and upload to the Canvas assignment.

Problem 2: OpenMP Analysis Report (10 pts)

Upload a short essay in a text file (Lab2.Prob2.txt) describing and comparing the performance of your application when compiled using OpenMP flag verses not using the flag. Run multiple tests using larger and larger matrix sizes and indicate when OpenMP performed better than not using OpenMP. What was the largest test size and what were the relative run times?

Grading Rubric

If a student's program runs correctly and produces the desired output, the student has the potential to get a 100 on his or her homework; however, TA's will **randomly** look through this set of "perfect-output" programs to look for other elements of meeting the lab requirements. The table below shows typical deductions that could occur.

⇨ ***Execution Time is important for this lab.***

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Files named incorrectly	10%	Per problem.
Execution Time	Up to 8%	0 pts deducted < 10x shortest time pts deducted = (2/10) (your time/shortest time) - 2 (rounded up to whole point value, with 4 pts maximum deduction for each problem)
Does Not Compile	30%	Code does not compile on PACE-ICE!
Does Not Match Output	10%-90%	The code compiles but does not produce the correct outputs.
Clear Self-Documenting Coding Styles	10%-25%	This can include incorrect indentation, using unclear variable names, unclear/missing comments, or compiling with warnings. (See Appendix A)

LATE POLICY

Element	Percentage Deduction	Details
Late Deduction Function	score – 0.5 * H	H = number of hours (ceiling function) passed deadline

***You are free to post solution times on pizza so that other students can gauge their run times.

Appendix A: Coding Standards

Indentation:

When using *if/for/while* statements, make sure you indent 4 spaces for the content inside those. Also make sure that you use spaces to make the code more readable.
For example:

```
for (int i; i < 10; i++)
{
    j = j + i;
}
```

If you have nested statements, you should use multiple indentions. Each { should be on its own line (like the *for* loop) If you have *else* or *else if* statements after your *if* statement, they should be on their own line.

```
for (int i; i < 10; i++)
{
   if (i < 5)
   {
       counter++;
       k -= i;
   }
   else
   {
       k +=1;
   }
   j += i;
}
```

Camel Case:

This naming convention has the first letter of the variable be lower case, and the first letter in each new word be capitalized (e.g. firstSecondThird). This applies for functions and member functions as well! The main exception to this is class names, where the first letter should also be capitalized.

Variable and Function Names:

Your variable and function names should be clear about what that variable or function is. Do not use one letter variables, but use abbreviations when it is appropriate (for example: “imag" instead of “imaginary”). The more descriptive your variable and function names are, the more readable your code will be. This is the idea behind self-documenting code.

File Headers:
Every file should have the following header at the top

```
/*
Author: your name
Class: ECE4122 or ECE6122 (section)
Last Date Modified: date

Description:

What is the purpose of this file?

*/
```

Code Comments:

1. Every function must have a comment section describing the purpose of the function, the input and output parameters, the return value (if any).
2. Every class must have a comment section to describe the purpose of the class.
3. Comments need to be placed inside of functions/loops to assist in the understanding of the flow of the code.