ECE 4122/6122 Lab 2: Using OpenMP to Calculate the Electric Field Produced by Array of Point Charges

(100 pts)

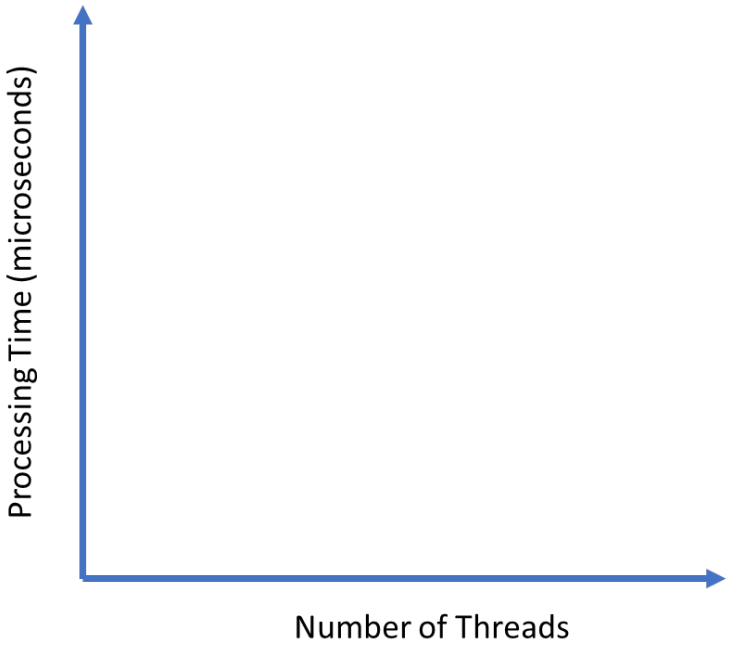
Category: Custom Classes and Multithreading
Due: Saturday October 7th, 2023 by 11:59 PM

Objective:

To understand and apply the principles of **OpenMP** for parallel calculations in a computationally-intensive problem related to the Electric Field calculations.

Description:

- Redo Lab1 but use OpenMP to create a multithreaded application.
- You do not have to use the classes from Lab1.
- Create a short one page report (pdf) describing your results with a graph showing the processing time results of a 1000 x 1000 grid for
  - 1 thread
  - 2 threads
  - 4 threads
  - 8 threads
  - 16 threads



OverView:

Create an application that uses OpenMP for multithreading to calculate the electric field at a point in space.

- **Your program should query the user for how many threads to run concurrently when doing the calculation.**
- Your program needs to query the user for the size of the **N x M** array and the charge **q**.
- Your program should continuously prompt the user if a new calculation is wanted.
- Your program should use OpenMP to reduce the processing time taken.
- After calculating the resultant electric field your program should output the Ex, Ey, Ez values and the resultant electric field strength.
- Your program should also output how long it takes in microseconds from the time the user inputs the xyz location to just before the results are output to the screen.
- Spaces separate the values when entering multiple inputs.
- Make sure you check for valid inputs
    - N and M should be natural numbers.
    - Separation distances should be > 0.0 and be valid numerical values
    - Charge should be a valid numerical value.
    - Point location should be made up of valid numerical values.
- Make sure that your code checks for a location that may be the same as a point charge location.

Sample Program Flow:

- Please enter the number of concurrent threads to use: 16
- Please enter the number of rows and columns in the N x M array: 100 100
- Please enter the x and y separation distances in meters: 0.01 0.03
- Please enter the common charge on the points in micro C: 0.02
-
- Please enter the location in space to determine the electric field (x y z) in meters: 1.0 2.0 3.0
- The electric field at (1.0, 2.0, 3.0) in V/m is
- Ex = x.xxxx * 10^y
- Ey = x.xxxx * 10^y
- Ez = x.xxxx * 10^y
- |E| = x.xxxx * 10^y
- The calculation took x.xxxx microsec!
- Do you want to enter a new location (Y/N)? Y
-
- Please enter the location in space to determine the electric field (x y z) in meters: 2.0 2.0 2.0
- The electric field at (2.0, 2.0, 2.0) in V/m is
- Ex = x.xxxx * 10^y
- Ey = x.xxxx * 10^y
- Ez = x.xxxx * 10^y
- |E| = x.xxxx * 10^y
- The calculation took x.xxxx microsec!
- Do you want to enter a new location (Y/N)? N
- Bye!
-

Turn-In Instructions

Place your files in a zip file called **Lab1.zip** and upload this zip file on the assignment section of Canvas.

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 look through your code for other elements needed to meet the lab requirements. The table below shows typical deductions that could occur.

AUTOMATIC GRADING POINT DEDUCTIONS PER PROBLEM:

Element	Percentage Deduction	Details
Does Not Compile	40%	Code does not compile on PACE-ICE!
Does Not Match Output	Up to 90%	The code compiles but does not produce correct outputs.
Runtime and efficiency of code setup	Up to 20%	The code generates the correct output but runs slower than expected.
Clear Self-Documenting Coding Styles	Up to 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

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 represents. 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.