ECE 4122/6122 Lab 6

(100 pts)

Section	Due Date
All Sections	Dec 5th, 2021 by 11:59 PM

Notes:

You can write, debug and test your code locally on your personal computer. However, the code you submit must compile and run correctly on the PACE-ICE server.

You need to upload any files needed to compile your code on PACE-ICE.

Grading Rubric

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	10%-90%	The code compiles but does not produce 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)

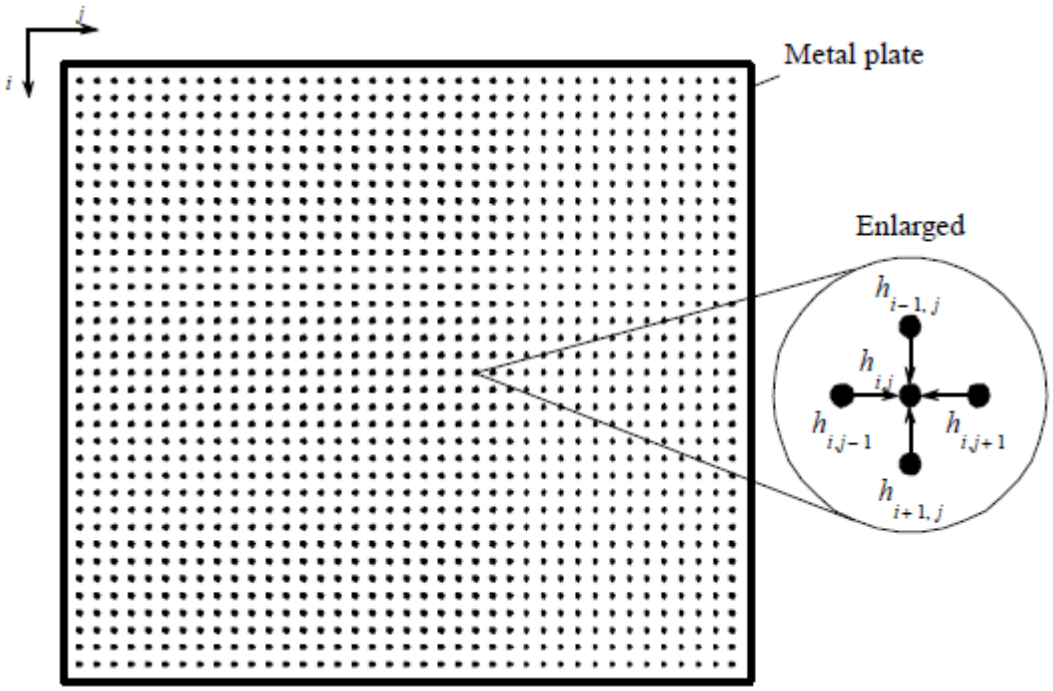
LATE POLICY

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

2D Steady State Heat Conduction in a Thin Plate

For this lab you will be writing a MPI program to determine the steady state heat distribution in a thin metal plate by submitting a job to the pace-ice system. You will be solving Laplace's equation using the finite difference method which has wide applications in science and engineering.

Consider a thin plate is perfectly insulated on the top and bottom which has known temperatures along each of its edges. The objective is to find the steady state temperature distribution inside the plate. The temperature of the interior will depend upon the temperatures around it. We can find the temperature distribution by dividing the area into a fine mesh of points, $h_{i,j}$. The temperature at an inside point can be taken to be the average of the temperatures of the four neighboring points, as illustrated below.



For this calculation, it is convenient to describe the edges by points adjacent to the interior points. The interior points of $h_{i,j}$ are where $0 < i < n$, $0 < j < n$ [(n -1) x (n - 1) interior points]. The edge points are when i = 0, i = n, j = 0, or j = n, and have fixed values corresponding to the fixed temperatures of the edges. Hence, the full range of $h_{i,j}$ is $0 \leq i \leq n,\ 0 \leq j \leq n$, and there are (n + 1) x (n + 1) points. We can compute the temperature of each point by iterating the equation:

$$h_{i,j} = \frac{h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1}}{4}$$

($0 \leq i \leq n,\ 0 \leq j \leq n$) for a fixed number of iterations or until the difference between iterations of a point is less than some very small prescribed amount. This iteration equation occurs in several other similar problems; for example, with pressure and voltage. More complex versions appear for solving important problems in science and engineering. In fact, we are solving a system of linear equations. The method is known as the finite difference method. It can be extended into three dimensions by taking the average of six neighboring points, two in each dimension. We are also solving Laplace'sequation.

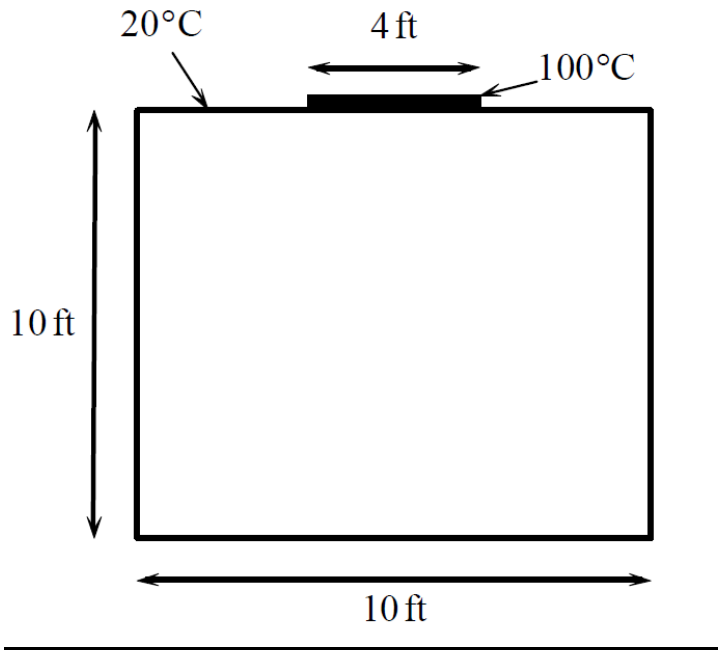
Suppose the temperature of each point is held in an array h[i][j] and the boundary points h[0][x], h[x][0], h[n][x], and h[x][n] ($0 \leq x \leq n$) have been initialized to the edge temperatures. The calculation as sequential code could be

```
for (iteration = 0; iteration < limit; iteration++) {

   for (i = 1; i < n; i++)
      for (j = 1; j < n; j++)
         g[i][j] = 0.25*(h[i-1][j]+h[i+1][j]+h[i][j-1]+h[i][j+1];

   for (i = 1; i < n; i++)// update points, Jacobi iteration
      for (j = 1; j < n; j++)
         h[i][j] = g[i][j];

}
```

using a fixed number of iterations. Notice that a second array g[][] is used to hold the newly computed values of the points from the old values. The array h[][] is updated with the new values held in g[][]. This is known as Jacobi iteration. Multiplying by 0.25 is done for computing the new value of the point rather than dividing by 4 because multiplication is usually more efficient than division. Normal methods to improve efficiency in sequential code carry over to GPU code and should be done where possible in all instances. (Of course, a good optimizing compiler would make such changes.)

Setup:

A perfectly insulted thin plate with an interior starting temperature of 20 °C and the sides held at 20 °C with a short segment on one side is held at 100 °C is shown below:



You need to write a C\C++ program using MPI to solve the steady state temperature distribution in the thin plate shown above. Your program needs to take the following command line arguments:

1. -N 256 - the number of N x N interior points
2. -I 10000 – the number of iterations

Uses type **double** for your arrays. Setup your Lab6.pbs file to request 16 nodes with 4 processors per node for a total of 64 processors. The software you develop must evenly distribute the processing of the calculations across all the processors. You are free to use any MPI functions you find appropriate.

The *main task* for your code needs to output to the console the number of milliseconds it took to calculate the solution.

The *main task* in your code needs to write out to a text file the final temperature values using a comma to separate the values in the file "finalTemperatures.csv". Each row of temperature values should be on a separate line.

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
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.

Reference:

https://webpages.uncc.edu/abw/coit-grid01.uncc.edu/ITCS4145F12/Assignments/assign5F12.pdf