Assignment #2

Heat Distribution Simulation

Problem Description

The temperature of the interior of an area will depend on the temperatures around it. Consider a square area that has known temperatures along each of its edges, and a single heat source at one location. The scientific question that we would like to answer is, "What is the distribution of heat in a region over time given a fixed heat source?"

The approach to solving this problem is divide the area into a fine mesh of points, h[i][j]. Temperature at an inside point are taken to be the average of temperatures of four neighboring points. Figure 1 shows the model of the area as a square mesh of points. One of the points is enlarged.

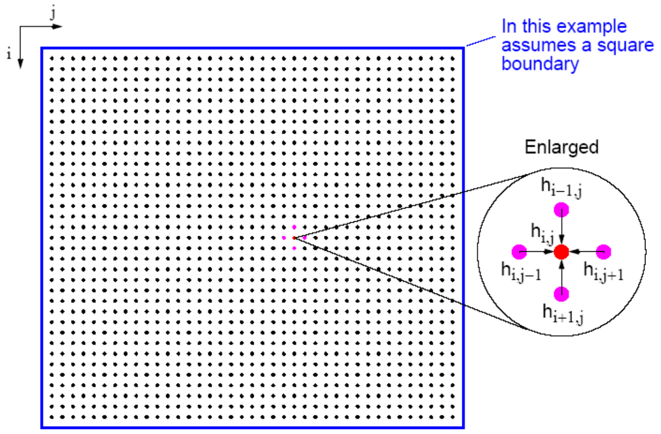

Figure 1: Mathematical Model of Area (Source: Wilkinson, 2004)

Assignment

You are to write a program that uses MPI and a data parallel technique to solve the heat distribution problem.

Part 1:

You should start by implementing the sequential version of this program. This program should declare a mesh of an appropriate size, which may depend on the graphical output technique that you use. If you use a bitmap format, a size of 1000 by 1000 is a good size to start with. The temperature of the edges should be fixed at 20 degrees Celsius. You should also initialize the interior of the mesh to 20 degrees Celsius, but these will change as the program executes.

The program should define a fireplace that is about 40% of the room width and placed at the top of the room. The temperature of the fireplace should be set to 300 degrees Celsius. Your program should run iteratively to calculate the temperature of each coordinate in the mesh. It should produce graphical output of the heat distribution that is at least as descriptive as the example in Figure 2.

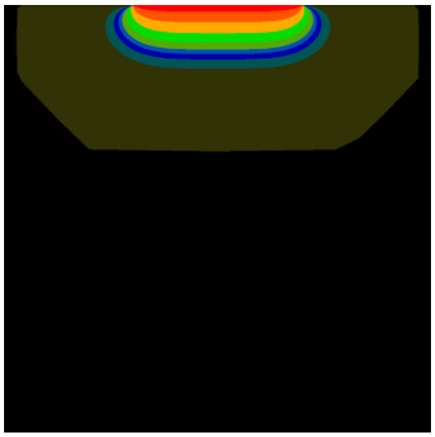
Figure 2: Example output of the heat distribution program

The following sequential code can help you get started. In the main program:

```
for each iteration {
   CopyNewToOld(new, old);
   CalculateNew(new, old, xsource, ysource);
   Optionally, PrintGridtoFile(new, xsource, ysource);
}
```

You should implement each of the three steps inside this loop in a function. You may wish to use a function prototype for each of these such as these, but you may modify them if prefer:

```
void CopyNewToOld(float new[][COLS], float old[][COLS]);
void CalculateNew(float new[][COLS], float old[][COLS], ... );
void PrintGrid(float grid[][COLS], int xsource, int ysource);
```

The location and temperature of the heat source will need to stay fixed at all times. The CalculateNew function will have code that is similar to:

```
for (i = 1; i < n-1; i++)
    for (j = 1; j < n-1; j++)
       new[i][j] = 0.25*(old[i-1][j]+old[i+1][j]+old[i][j-1]+old[i][j+1]);
```

While you are testing, you should print output after each iteration to see how the array changes. After you have finished testing you should run the program without printing in each iteration for a large number of iterations, perhaps 50,000, to see the results of the converged calculation.

Your program should use graphical output. You may use any technique that you like, but one technique using an ascii bitmap file is demonstrated in the sample code: printcolors.c. The output of this code is in .pnm format, and you can convert them to .gif or .jpeg using the command "convert [input file] [output file]" on Palmetto. The following range of colors may be useful to you, although you can change them if you like:

If the temperature is above 250, print in RED
If the temperature is between 180 and 250 print in ORANGE

If the temperature is between 120 and 180 print in YELLOW
If the temperature is between 80 and 120 print in LTGREEN
If the temperature is between 60 and 80 print in GREEN
If the temperature is between 50 and 60 print in LTBLUE
If the temperature is between 40 and 50 print in BLUE
If the temperature is between 30 and 40 print in DARKTEAL
If the temperature is between 20 and 30 print in BROWN
If the temperature is 20 or less print in BLACK

Part 2:
Parallelize your program using a data parallel technique in MPI. It is recommended that you divide the area into strips of equal size, but you can use any partitioning technique that you like. You should use temporary data arrays to hold data points that are used by an MPI process as input to the calculation but are not modified in that process. These temporary arrays must be communicated in each iteration along with the set of data calculated by each process. The MPI algorithm is similar to the sequential algorithm:

```
for each iteration {
   send/receive ghost points from neighbor process
   CopyNewToOld(new, old);
   CalculateNew(new, old, xsource, ysource);
   Optionally, PrintGridtoFile(new, xsource, ysource);
}
```

Teamwork
You may form a team with up to 2 students (including yourself) in the class to work on this assignment together. Select a team lead to represent your team.

Submission
The team lead is required to submit the source code developed by the team for both sequential and parallel programs, any compiling instructions/assumptions necessary, as well as a readme.txt file describing the roles each team member takes in this assignment on Canvas by the due date. One unified grade will be given to each member of the team in this assignment.

The grade you receive will depend on the correctness of the program as well as the programming design. Some parts of this assignment may be left unspecified. Wherever you make assumptions or design decisions you should state these explicitly for full credit. For full credit your program must compile without errors and must execute correctly with a variety of number of processors. It should have adequate comments and use good software engineering style and techniques. Partial credit will still be granted even if your code is incomplete but some functions are implemented.