CS 213 — Multiprocessor Architecture and Programming

Programming Assignment 2 - OpenMP

1 Heat Distribution

The objective is to write an OpenMP program that will model the static heat distribution of a room with a fireplace using a stencil pattern. Although a room is 3 dimensional, we will be simulating the room with 2 dimensions. The room is 10 feet wide and 10 feet long with a fireplace along one wall as depicted in Figure 1. The fireplace is 4 feet wide and is centered along one wall (it takes up 40% of the wall, with 30% of the walls on either side). The fireplace emits 100^o C of heat (although in reality a fire is much hotter). The walls are to be considered 20^o C. The boundary values (the fireplace and the walls) are considered to be fixed temperatures.

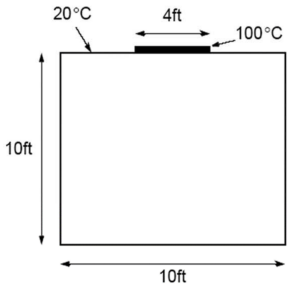

Figure 1: 10ft by 10ft room with a fireplace

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 neighbouring points, as illustrated in Figure 2:

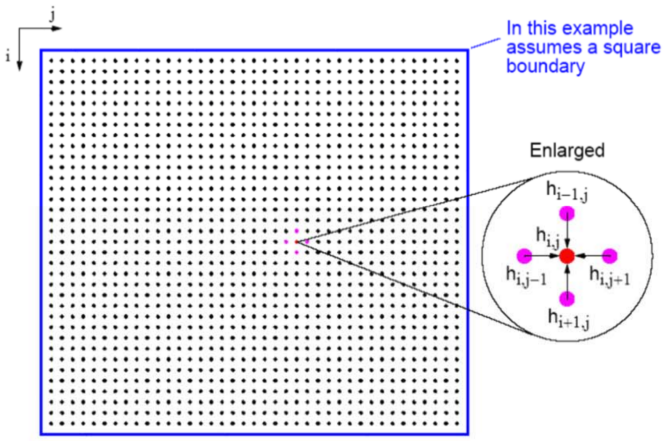

Figure 2: Determining heat distribution by a finite difference method

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 $0 < i < n$ and $0 < j < n$ with $(n-1) \times (n-1)$ 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 $(n+1) \times (n+1)$ points. We can compute the temperature of each point by iterating the foolowing equation ($0 < i < n$, $0 < j < n$) for a fixed number of iterations or until the difference between iterations of a point is less than some very small prescribed amount.

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

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's equation.

Sequential Code: 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 as following using a fixed number of iterations.

```
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 */
    for (j = 1; j < n; j++)
      h[i][j] = g[i][j];
}
```

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 parallel code and should be done where possible in all instances. (Of course, a good optimizing compiler would make such changes.)

1.1 Task 1: Sequential Program

The above code can be improved to avoid actually copying the array into another array by using a 3-dimensional array of size $N \times N \times 2$, say A[i][j][x] and every iteration, alternate between $x = 0$ to $x = 1$. Re-write the code given to use a 3-dimensional array avoiding the update loop

and complete as a C program to model the heat distribution of room given in Figure 1. Instrument the code so that the elapsed time is displayed. Have $N \times N$ points and T iterations where N and T are user inputs. Output on the console the values of every $N/8 \times N/8$ points i.e. 8×8 points irrespective of the value of N. Test your program on your own computer, with different values for N and T (e.g., N = 1000 and T = 5000).

Command to execute your program: ./heatseq N T

Note: you code will be tested with this exact command. If your code does not run with this command, you do not get the implementation credit.

1.2 Task 2: OpenMP Program

Modify the sequential program in Task 1 to be an OpenMP program. Try to improve the performance as much as possible. Code to ensure both sequential and parallel versions of heat distribution calculation produce the same correct results.

Command to execute your program: ./heatpar N T

Note: you code will be tested with this exact command. If your code does not run with this command, you do not get the implementation credit.

1.3 Task 3: Graphical Output

Make your sequential heat distribution program produce the temperature contour (heat distribution) at 10^0 C intervals in color. Repeat for the OpenMP program.

1.4 What to Submit

You should submit a single Zip file (rename your file firstName-LastName) which includes the sequential and OpenMP implementation files (each 20% of the grade) and a single PDF for your report (60%).

Your PDF report should include:

1. Screenshot of compiling the sequential program (5%)
2. Screenshot of command to execute the sequential program and program output (5%)
3. Screenshot of compiling the OpenMP program (5%)
4. Screenshot of command to execute the OpenMP program and program output (5%)
5. A short paragraph explaining your findings on what worked best to get performance for your OpenMP implementation (10%)

6. A graph to show the speed up of OpenMP version over the sequential version for N = 1000, 10,000, 100,000 and T = 5,000, 50,000, 500,000. (10%)

7. Screenshot of the graphical output (heat map) of the sequential program (10%)

8. Screenshot of the graphical output (heat map) of the OpenMP program (10%)