Assignment 2
Second OpenMP Programming Assignment

B. Wilkinson and C Ferner: Modification date Sept 4, 2014

Overview

In this assignment, you will write and execute your own OpenMP program to model the static heat distribution of a room with a fireplace (two dimensions only) using the stencil pattern. You are also asked to generate X11 graphical output and sample X11 code is provided. First you will develop and test OpenMP programs on your own computer (VirtualBox or a native Linux installation). Later you will test the programs on the UNC-Charlotte's 4-processor (16-core) **cci-grid05.uncc.edu** system. This approach reduces issues of faulty user programs running on the UNC-C cluster that can affect the system in a deleterious manner. Users can also do local editing and testing before running on the cluster. The last task is for extra credit and asks you to modify the heat distribution program for dynamic heat distribution where the heat sources vary with time.

Heat Distribution (Static Heat Equation)

The objective is to write an OpenMP program that will model the *static* heat distribution of a room with a fireplace (where the heat source temperatures do not vary with time) using a stencil pattern. Although a room is 3-dimensional, we will be modeling the room in two dimensions. The room is 10 feet wide and 10 feet long with a fireplace along one wall as depicted in Figure 1.

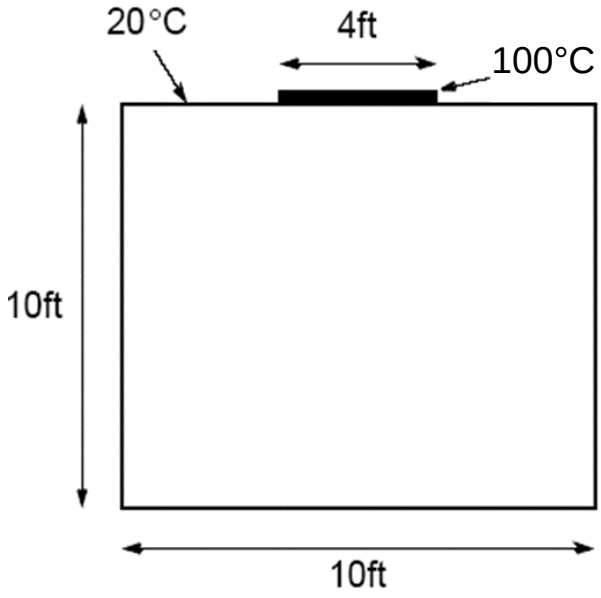

Figure 1: 10 x 10 Room with a Fireplace

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 100º C of heat (although in reality a fire is much hotter). The walls are considered to be at 20º C. The boundary values (the fireplace and the walls) are considered to be fixed temperatures.

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 in Figure 2:

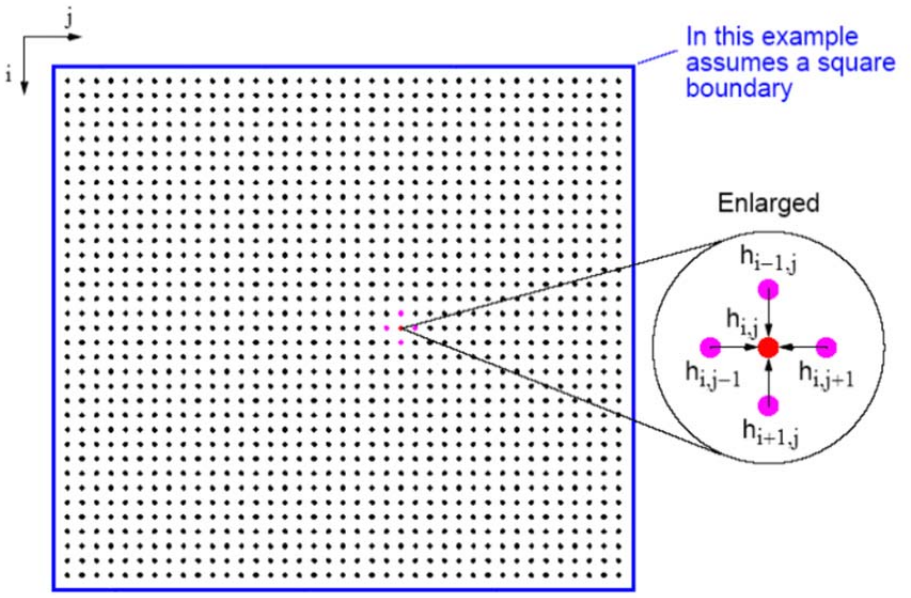

Figure 2: Determining Heat Distribution by a Finite Difference Method

For this calculation, it is convenient to include the edges by points in the array so that with an N x N array (i.e. h[N][N]), the interior points of $h_{i,j}$ are where $0 < i < N$-1, $0 < j < N$-1. The edge points are when $i = 0$, $i = N$-1, $j = 0$, or $j = N$-1, and have fixed values corresponding to the fixed temperatures of the edges. We can compute the temperature of each point by iterating the equation:

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

($0 < i < N$-1, $0 < j < N$-1) for a fixed number of iterations or until the difference between iterations of each 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's equation.

Sequential Code. Suppose the temperature of each point is held in an array h[i][j] and the boundary points have been initialized to the edge temperatures. The calculation as sequential code could be

```
double h[N][N],g[N][N];

for (iteration = 0; iteration < MAX_ITERATONS; iteration++) {
 for (i = 1; i < N-1; i++)
   for (j = 1; j < N-1; 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-1; i++)       // update points
   for (j = 1; j < N-1; 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 a 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.

One way to improve the code further while still keeping the Jacobi iteration method is to extend the array into three dimensions to hold the present and next iteration values and then switch between them to avoid copying arrays, i.e:

```
double h[2][N][N];

current = 0;
next = 1;
for (iteration = 0; iteration < MAX_ITERATONS; iteration++) {
 for (i = 1; i < N-1; i++)
   for (j = 1; j < N-1; j++)
     h[next][i][j] = 0.25 * (h[current][i-1][j] + h[current][i+1][j]
                          + h[current][i][j-1] + h[current][i][j+1]);
 current = next;
 next = (current + 1)% 2;
}
```

The final result is in **h[current][][]**. If **MAX_ITERATONS** is even this would be **h[0][][]**. It is probably accurate enough to take it as that location even if **MAX_ITERATONS** was odd (one iteration more).

Preliminaries

The OpenMP programs for this assignment are to be held in the directory **~/ParallelProg/OpenMP**, which is already created in the provided virtual machine, with sample programs. Cd to this directory.

Task 1 Sequential program (15%)

Write a sequential program for the heat distribution calculation based upon the code given using the three-dimensional array **`h[2][N][N]`** (where the variable **`current`** toggles between zero and one). Instrument the code so that the elapsed time is displayed. Have *N* x *N* points and *T* iterations where *N* and *T* are set by keyboard input during program execution. Output on the console the values of every *N*/8 x *N*/8 points i.e. 8 x 8 points irrespective of the value of *N*.

Test your C program on your own computer with $N = 1000$ and $T = 5000$, and other values.

What to submit for Task 1

Your submission document should include *(but is not limited to)* the following:

- Your sequential C program listing to solve the heat distribution problem
- On your own computer
    - Screenshot of compiling the program
    - Screenshot of command to execute the program and program output

Task 2 - OpenMP Program (30%)

Modify the sequential program in Task 1 to be an OpenMP program. Include in your program:

- Sequential C code to compute the heat distribution (from Task 1)
- OpenMP code to compute the heat distribution
- Code to check that the sequential and parallel versions of heat distribution calculation produce the same correct results**.**
- Statements to time the execution of both sequential and parallel versions.
- Statements to compute the speed-up factor and display.

Test the program on your computer.

What to submit for Task 2

Your submission document should include *(but is not limited to)* the following:

- Your OpenMP program listing with all the features specified to solve the heat distribution problem
- On your own computer
    - Screenshot of compiling the program
    - Screenshot of command to execute the program and program output

Task 3 Graphical output (30%)

Read "*Notes on creating graphical output using X-11 graphics Ubuntu Virtual Machine*" posted on the home page for information on X11 code and how to compile X11 code.

(a) Sample X11 code is provided in the VM called **sample.c** within the directory **~/parallelProg/X11**. Compile this program and report on its output.

(b) Using **sample.c** as a basis, make your sequential heat distribution program in Task 1 produce X11 graphics displaying the temperature contours at 10°C intervals in color. Execute your program on your computer.

(c) Repeat for the OpenMP program from Task 2, adding X11 graphics displaying the temperature contours at 10°C intervals in color to the OpenMP program. Compile and execute on your computer.

What to submit for Task 3:

Your submission document should include *(but is not limited to)* the following:

- Screenshot of **sample.c** executing on your computer (Task 3 a).
- Screenshot of executing your sequential heat distribution program with graphical output on your computer (Task 3 b)
- Listing of your OpenMP heat distribution program with X11 code to solve the heat distribution problem (Task 3 c)
- Screenshot of executing your OpenMP program with graphical output on your computer (Task 3 c)
- Conclusions

Task 4 Using remote server (25%)

Read "*Notes on creating graphical output using X-11 graphics Ubuntu Virtual Machine*" posted on the home page describes how to execute your program on a remote server and forward the graphics to your computer. Execute your OpenMP heat distribution program with graphical output from Task 3 on **cci-grid05.uncc.edu** and forward the X11 output to your computer.

What to submit for Task 4:

Your submission document should include *(but is not limited to)* the following:

- Screenshot of compiling the program on **cci-grid05.uncc.edu**
- Screenshot of command to execute the program on **cci-grid05.uncc.edu**
- Screenshot of the graphical output of the program forwarded to on your computer
- Conclusions

Task 5 Dynamic Heat Distribution (Extra credit +15%)

Previously the heat distribution assumed that the heat sources did not vary with time and we solved Laplace's equation:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0$$

With heat sources that vary, this equation becomes the general Heat equation (Poisson's equation):

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = \frac{1}{\alpha}\frac{\partial f}{\partial t}$$

where α is a constant and *f*, the temperature, varies with time.

Modify the heat distribution program for dynamic heat distribution where the heat sources vary with time. You will need to do some research on how to reformulate the difference equations. Make your own decisions on the input data (time variable heat sources). Write a sequential program and then add OpenMP directives to parallelize it. Compile and execute on your computer.

What to submit for Task 5:

Your submission document should include *(but is not limited to)* the following:

- Your OpenMP program listing with full comments
- Screenshot of compiling the program on your computer
- Screenshot of command to execute the program on your computer and results
- Conclusions

Grading

Every task and subtask specified will be allocated a score so make sure you clearly identify each part/task you did. Make sure you include everything that is specified in the "Include in your submission document" section at the end of each part. **Include all code, not as screen shots of the listings but complete properly documented code listing in the report.**

Assignment Submission

Produce a ***single pdf*** document that show that you successfully followed the instructions and performs all tasks by taking screenshots and include these screenshots in the document. Submit by the due date as described on the course home page.