

Assignment 2

Second OpenMP Programming Assignment

B. Wilkinson: Modification date February 2, 2016

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.

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.

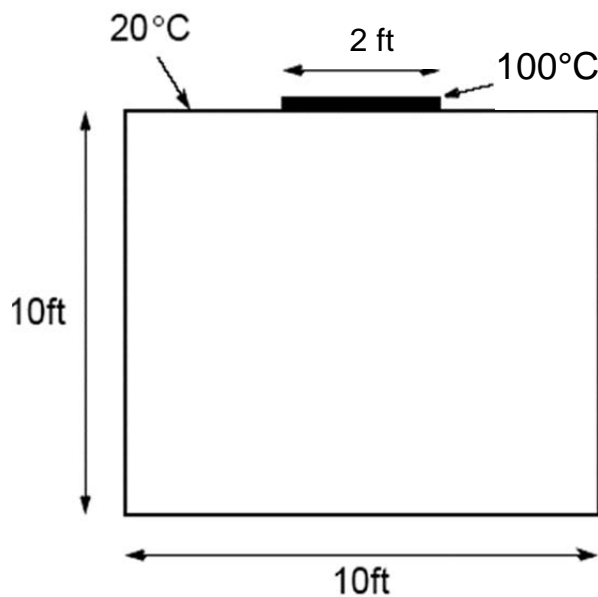


Figure 1: 10 x 10 Room with a Fireplace

The fireplace is 2 feet wide and is centered along one wall (it takes up 20% of the wall, with 40% of the walls on either side). The fireplace emits 100° C of heat (although in reality a fire is much

hotter). The walls are considered to be at 20° 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:

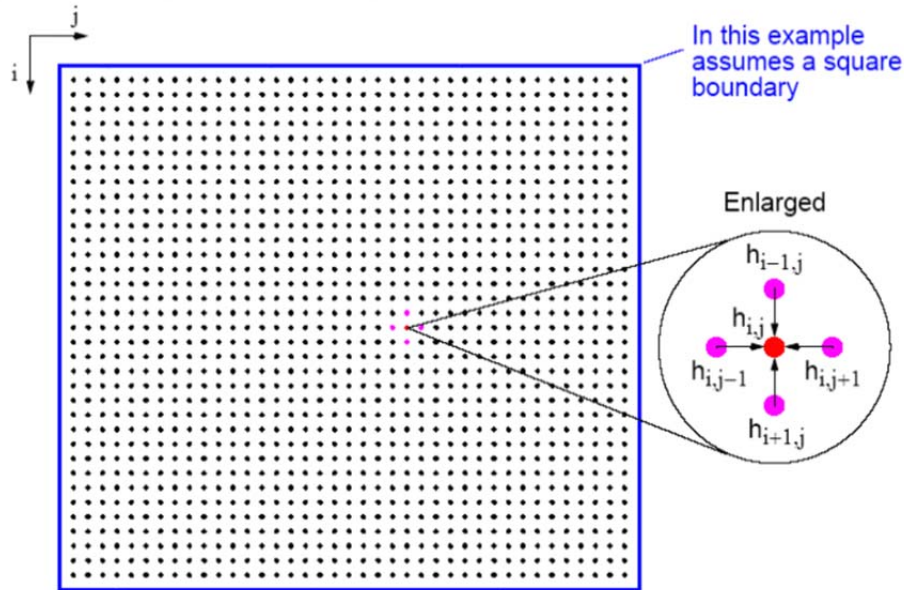


Figure 2: Determining Heat Distribution by a Finite Difference Method

For this calculation, it is convenient to include the edges by points in the array so that with an $N \times N$ 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;      // swap values of current and next
  next = 1 - current;
}
```

The final result is in $h[\text{current}][][]$. 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.

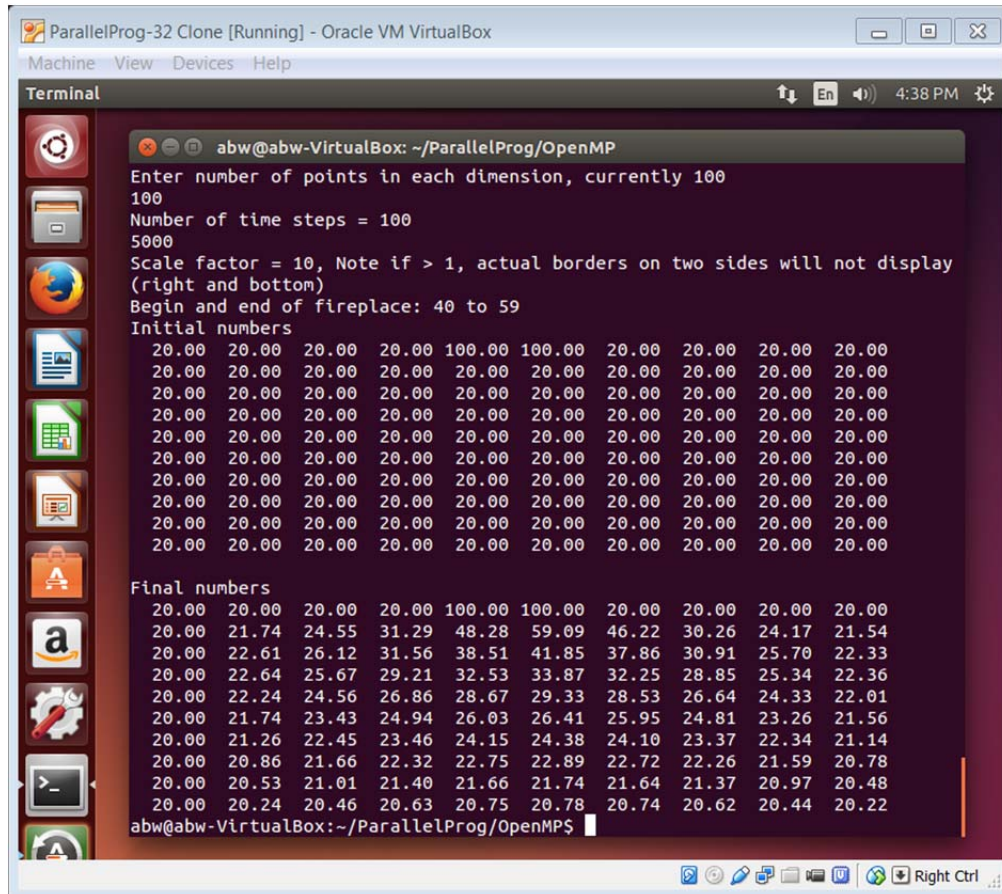
Task 1 Sequential program (20%)

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). Have $N \times N$ points and T iterations where N and T are set by keyboard input during program execution.

Feb 2, 2016: It is not required to use dynamically allocated arrays. You may use variable length arrays, i.e. arrays declared with variables as their indices. (However, variable length arrays will limit the size of N you can use. See FAQ, C programming.)

Output on the console the values of every $N/10^{\text{th}}$ point. For example, if $N = 100$, print out every 10^{th} point. Hence, there will be 10 rows of numbers, each row having 10 numbers, irrespective of the value of N .

Test your C program on your own computer with $N = 100$ and $T = 5000$, and two other larger values of N and T . Determine the largest value of N that will work on your platform (does not need to be the exactly the maximum value). Explain. Below is a sample from my program (using a faster sequential algorithm than given here so your output will be different for the same number of iterations).



```
ParallelProg-32 Clone [Running] - Oracle VM VirtualBox
Machine View Devices Help
Terminal 4:38 PM
abw@abw-VirtualBox: ~/ParallelProg/OpenMP
Enter number of points in each dimension, currently 100
100
Number of time steps = 100
5000
Scale factor = 10, Note if > 1, actual borders on two sides will not display
(right and bottom)
Begin and end of fireplace: 40 to 59
Initial numbers
 20.00 20.00 20.00 20.00 100.00 100.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00 20.00
Final numbers
 20.00 20.00 20.00 20.00 100.00 100.00 20.00 20.00 20.00 20.00
 20.00 21.74 24.55 31.29 48.28 59.09 46.22 30.26 24.17 21.54
 20.00 22.61 26.12 31.56 38.51 41.85 37.86 30.91 25.70 22.33
 20.00 22.64 25.67 29.21 32.53 33.87 32.25 28.85 25.34 22.36
 20.00 22.24 24.56 26.86 28.67 29.33 28.53 26.64 24.33 22.01
 20.00 21.74 23.43 24.94 26.03 26.41 25.95 24.81 23.26 21.56
 20.00 21.26 22.45 23.46 24.15 24.38 24.10 23.37 22.34 21.14
 20.00 20.86 21.66 22.32 22.75 22.89 22.72 22.26 21.59 20.78
 20.00 20.53 21.01 21.40 21.66 21.74 21.64 21.37 20.97 20.48
 20.00 20.24 20.46 20.63 20.75 20.78 20.74 20.62 20.44 20.22
abw@abw-VirtualBox:~/ParallelProg/OpenMP$
```

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
 - Results of using different values of N and T .

Task 2 - OpenMP Program (25%)

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 within rounding errors (as in Assignment 1).
- Statements to time the execution of both sequential and parallel versions. In both cases use `omp_get_time()` routines to record time stamps as in Assignment 1.
- Compute the speed-up factor (sequential time/parallel time) 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 with $N = 100$ and $T = 5000$.
 - A graph show in the speedup you obtain with different values of N and T . On your computer you may not get any speedup.

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

Task 4 Using remote server (10%)

“*Creating graphical output using X-11 graphics*” 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 3c on **cci-gridgw.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-gridgw.uncc.edu**
- Screenshot of command to execute the program on **cci-gridgw.uncc.edu**
- Screenshot of the graphical output of the program forwarded to on your computer
- Conclusions

Task 5 Changing the termination condition (15%)

So far we have used a Jacobi iteration that terminates when a given number of iterations have been done. Re-write the OpenMP program so that iterations terminate when all computed values do not change from one iteration to the next iteration by more a value ϵ , where ϵ is entered at the keyboard.

What to submit for Task 5:

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

- A full description of the method
- 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

Assignment Report Preparation and Grading

Produce a report that shows that you successfully followed the instructions and performs all tasks by taking screenshots and include these screenshots in the document. Note it is easy to obtain screenshots in Word as now it has that option in the **Insert > Screenshot** menu.

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 “What to include ...” section at the end of each task/part. **Include all code, not as screen shots of the listings but complete properly documented code listing.** The programs are often too long to see in a single screenshot and also not easy to read if small. Using multiple screenshots is not option for code. Copy/paste the code into the Word document or whatever you are using to create the report.

Assignment Submission

Convert your report into a *single pdf* document and submit the pdf file on Moodle by the due date as described on the course home page. It is extremely important you submit only a pdf file. *Do not submit in any other format (i.e. Word, OpenOffice, zip, ...).*

If you do not submit the report as a pdf file, you will automatically lose 10 point (10%).