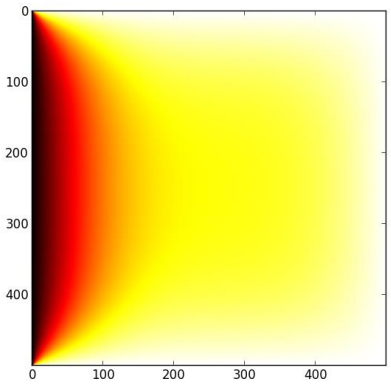
CS160 – Winter 2018 – Programming Assignment #1

Due 1159pm, Jan 27, 2018

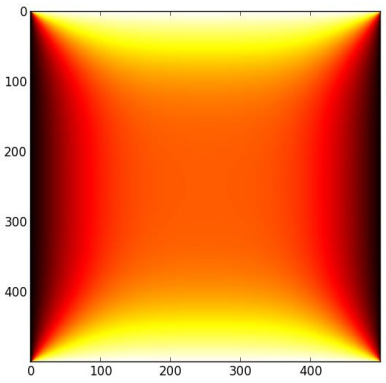
The goal of this assignment is to parallelize a given serial code that uses a simple iteration to solve the final temperature distribution on a plate using so-called Dirichlet (fixed temperature) boundary conditions. You will run your final code on TSCC, the Triton Shared Computing Cluster at SDSC, but you can develop your code in the computer labs. We'll also be (attempting to use) using Github classroom for turn in your final codes so that you have flexibility in designing your final code. We want you to run on TSCC so that timing (speed) improvements are more likely.

heat2d.c

You are given a serial code, called heat2d (and a very simple python program that will give you png of your output file). The following two figures were generated by running heat2d and then using heatmap.py to create the png graphics file .



Computation: **`heat2d 500 500 100 0 0.001 heatleft.out`**, *Visualization:* **`heatmap.py heatleft.out`**



Computation: **`heat2d 500 500 100 100 0.0001 heateven.out`** *Visualization:* **`heatmap.py heateven.out`**

Before you start

You will be using GitHub to turn in your code (details coming) and using TSCC. We need to assign you an account on TSCC and you need to tell us your GitHub account.

1. If don't have one already, create an account on GitHub. They are free.
2. Fill out the form at https://goo.gl/forms/QfdXR5MN715DWah53. We need your UCSD email address. **YOU MUST COMPLETE THIS FORM NO LATER THAN WED 24 JAN 2018**. It will take the TAs some time to add your ssh key to a predefined TSCC username and then give you your account id.

heat2d.c

Heat2d has the following usage:

```
usage: heat2d M N Tl Tr eps file
```

- `M = # of rows of the matrix`
- `N = # of columns of the matrix`
- `Tl = Temperature at the left boundary`
- `Tr = Temperature at the right boundary`
- `Eps = threshold to stop iterations`
- `File = output file, final Temperature distribution`

Your first set of tasks is to modify heat2d to do the following

1. Read heat2d so you understand what it does.
2. The usage should be updated to be

```
usage: heat2d M N Tl Tr Tt Tb eps file
```

```
1. M = # of rows of the matrix
2. N = # of columns of the matrix
3. Tl = Temperature at the left boundary
4. Tr = Temperature at the right boundary
5. Tt = Temperature at the top boundary  (row index = 0)
6. Tb = Temperature at the left boundary (row index = M-1)
7. Eps = threshold to stop iterations
8. File = output file, final Temperature distribution
```

The code should then be modified so that Tt and Tb are utilized in the code instead of being set to 0 as they are in the original.

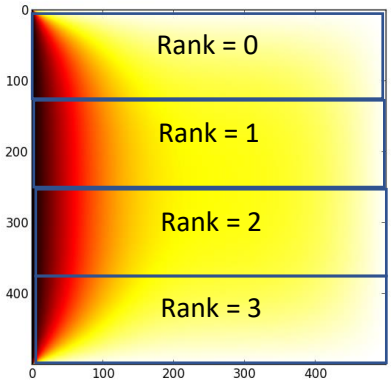
3. All command line arguments should be validated (as in PR1).
    a. M,N must be positive integers >= 10
    b. Tl, Tr,Tt,Tb must be valid doubles (we won't test with large are very small values)
    c. Eps is a valid positive float with value less than 1.0 (0.001 is a good threshold)
4. Create a Makefile that will properly build (and rebuild) heat2d when heat2d.c is updated
    a. We will test your program build with "make heat2d"
    b. Have a clean target that will remove all *.o (object files) and executables.

Parallelizing heat2d (two different parallel implementations)

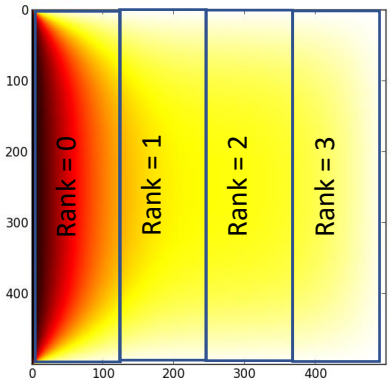
You will parallelize heat2d into two different versions: heat2dcol.c and heat2drow.c. The difference is a column-wise distribution vs. a row-wise distribution of data. (heat2dcol.c should be column decomposition, heat2drow.c should be a row decomposition)

Each processor in your MPI parallel implementation must only allocate the memory it needs to perform its part of the computation. It will not be acceptable code for each MPI process allocate an M X N double array (though an intermediate development version of your code might do this for simplicity).

Example of 4-way parallel data distribution.



Row Decomposition (heat2drow.c)



Column Decomposition of Data

Testing performance

You should test the parallel performance of your codes, please create a table of timings, run on TSCC nodes that give the performance of heat2drow and heat2dcol running each running on

- 1,2,4,8 processors
- Problem sizes of 500, 1000,2000

- You may use whatever boundary conditions you want, but be consistent among all the test cases

Hints/Notes

- Do not time how long it takes to gather final state of the matrix from the parallel processors and print to a file. We are only interested in how long the parallel computation takes
- The row decomposition is easier in C than the column decomposition, why ?
- Work with small matrices (100x100, 200x200) when developing, you will appreciate that the computation is quicker in these small test cases.
- You should be able to reuse a great deal of code between the row version and column version. Structure your solution to take advantage of that. We frown on wholesale code copying between the two versions (e.g. argument handling, etc).
- You can use MPI built in collectives to determine if all processes have converged

What to turn in

- Makefile
- Job submission script used to test 4 processors, problem size 1000, heat2drow
- All of your *.c and *.h files needed to compile your code
- A txt file that is a write up of your performance testing. If you notice any performance differences between the row distribution and column distribution, please provide a brief explanation.

Turning your program

Details provided later in the week, No later than Friday prior to the program due date. Likely earlier than this. Please bear with us, we are learning how to use GitHub classroom, too.

What you will be graded on

- Correctness in both error and non-error cases
- Indentation – Your code must be indented. Choose a style and stick with it.
- Comments – You must comment your code in a reasonable way. At a minimum, your name, ucsd email address, student ID must appear in comments. All your routines must be commented with a brief description of what the routine does, what it returns and a description of the parameters. `svalidate.c` is sufficiently commented.
- We will run your code for testing and compare your output files to known good ones in various cases.

- You may have your code print whatever you like to standard output, standard error. When usage is printed, it should only be printed by one process
- Your file output should be written by process 0.