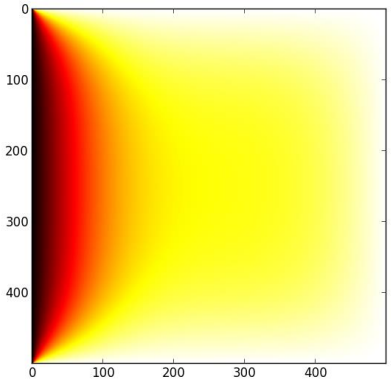
CS160 – Winter 2018 – Programming Assignment #4

Due 1159pm, Feb 25, 2018 (SUNDAY, one day extension)

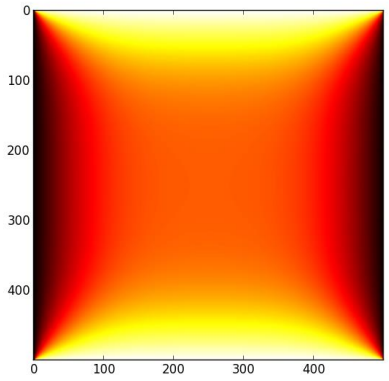
The goal of this assignment is to parallelize the serial code similar to the one used in PR2 using pthreads instead of MPI. It is simple iteration to solve the final temperature distribution on a plate using so-called Dirichlet (fixed temperature) boundary conditions. You will run your final code on TSCC, the Triton Shared Computing Cluster at SDSC, but you can develop your code in the computer labs. We'll also be 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 file is a little different that the one in PR2 but *should* generate identical results. 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 0 0.001 heatleft.out`**, *Visualization:* **`heatmap.py heatleft.out`**



Computation: **`heat2d 500 500 100 100 0 0 0.0001 heateven.out`** *Visualization:* **`heatmap.py heateven.out`**

heat2d.c

Heat2d has the following usage:

```
usage: heat2d M N Tl Tr Tt Tb 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`
- `Tt = Temperature at the top boundary`
- `Tb = Temperature at the bottom boundary`
- `eps = threshold to stop iterations`
- `File = output file, final Temperature distribution`

Parallelizing heat2d (A single pthreads implementation)

You will parallelize heat2d using pthreads. You may choose a row or column decomposition of the matrix.

Your final program will called heat2d and it will have one additional (optional) parameter so that it is invoked via

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

if the final argument is present, it specifies the number of threads to use to parallelize heat2d. if nthreads > 1, then that number of threads should cooperatively solve the heat equation via iteration.

You will need to determine and implement the appropriate synchronization among the threads. If that argument does not appear, it is equivalent to specifying nthreads=1.

Requirements

- You only need to check # of arguments. We will not test with erroneous or badly-formed arguments. Your program should handle any reasonable combination of the input arguments.
- M/N might not be evenly divisible by nthreads
- Parallelization must be accomplished via pthreads
- If M,N, Tl, Tr, Tt,Tb,eps are fixed, the number of threads should NOT have any effect on the accuracy or total number of iterations required. That is, the single threaded code and multi-threaded must give identical numerical results.
- Your code should have no memory leaks. We likely will use valgrind to check this for your code (valgrind is available on ieng6. It's not available on TSCC).
- You must not change the output file format (already coded for you).
- `$ make heat2d` should properly build your executable
- `$ make clean` should remove all object files and executables
- `$ make` should be the equivalent of "`make heat2d`"

Hints/Notes

- You may or may not see any real speed up (slowdown is possible). The goal of this particular program is correct computation. Synchronization penalties might erase any benefit of using multiple processors.
- Work with small matrices (100x100, 200x200) when developing, you will appreciate that the computation is quicker in these small test cases.
- Verify that your program works properly at reasonable thread counts, e.g., 1,2,4,8,16,32. Check odd numbers of threads, too.
- Your program should not freeze or hang for any set of reasonable inputs.
- Think about "shared variables" and how "synchronized" your threads need to be to insure identical numerical results to the single threaded code
- You might want to use pthread condition variables to implement a logical barrier. You may use pthread_barrier, but be sure to test it when a large number of iterations are required.
- Running on TSCC – Section will cover this or you can go to the page http://www.sdsc.edu/support/user_guides/tscc-quick-start.html. Please do NOT use tscc-login.sdsc.edu to login. Instead use tscc-login4.sdsc.edu (this is dedicated to our class)
- Matplotlib is not on TSCC, if you want to generate pretty pictures, please do this on ieng6
- `$ qsub -I -l nodes=1:ppn=4` will give you an interactive session on TSCC, this is a good test environment.

What to turn in

- Makefile
- All of your *.c and *.h files needed to compile your code. NO object files or executable files should be included

- Output file from heat2d running on 4 threads at size 2000 x 2000. This file should be called **heat2d2K.log**

Turning your program

Just like PR2 and PR3. Don't forget to push your final changes in Github.

What you will be graded on

- Correctness at various sizes and thread counts. We'll test non-evenly divisible M/N with respect to thread count, odd numbers of threads, etc.
- Did you use "global" variables instead of defining these variables as parameters to your threads?
- 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.
- 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 once.
- Absence of memory leaks. (Validated with valgrind)

Example of running valgrind on ieng6

```
$ valgrind ./heat2d 200 200 100 0 0 50 0.001 out1.txt 4
==883== Memcheck, a memory error detector
==883== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==883== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright
info
==883== Command: ./heat2d 200 200 100 0 0 50 0.001 out1.txt 4
==883==
HEAT2D
  C version
  A program to solve for the steady state temperature distribution
  over a rectangular plate.
  Spatial grid of 200 by 200 points.

  The iteration will be repeated until the change is <= 0.001
  Boundary Temperatures  left: 100  right: 0
  The steady state solution will be written to 'out1.txt'.
Iteration  Change
         1  18.781407
         2  8.605528
         4  3.715845
         8  1.951484
        16  0.974135
```

```
        32  0.483341
        64  0.240750
       128  0.120421
       256  0.060082
       512  0.030065
      1024  0.015037
      2048  0.007512
      4096  0.003470
      8027  0.001000

  Error tolerance achieved.
  CPU time = 49.910000

  Solution written to the output file 'out1.txt'

HEAT2D:
  Normal end of execution.
==883==
==883== HEAP SUMMARY:
==883==     in use at exit: 0 bytes in 0 blocks
==883==   total heap usage: 218 allocs, 218 frees, 343,608 bytes
allocated
==883==
==883== All heap blocks were freed -- no leaks are possible
==883==
==883== For counts of detected and suppressed errors, rerun with: -v
==883== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```