Threads can be used to reduce “wall clock” run time on a multi-CPU system. For this assignment you are to get a copy of my program “jacobi.c” from the /home/phil/public/csci347 directory on the lab machines and modify it to use Pthreads so it can run faster.

Jacobi Iterations is a method of iterative numerical solution to Laplace’s Equation. It is used in many circumstances. Heat distribution, or electrical fields are common uses. It provides a good platform to investigate threading and speed-ups using Pthreads. The area under consideration is represented by a 2-D array of values. Some values are fixed and the other values are given an initial value. The computation recomputes all non-fixed values in the 2-D array at every iteration. A point’s value in the next iteration is the average of the four points nearest the point in the current array. The difference from the current value to the new value at point \( i, j \) is known as \( \Delta_{i,j} \). The computation quits when the maximum of the \( \Delta_{i,j} \) values for single iteration is less than some tolerance, typically a small number like 0.000001.

The program provided for you does this computation using a single thread. The edges of the 2-D array are the fixed values. The size of the array is specified in “rows” and “columns”, but the actual arrays have two more rows and columns. The extra rows and columns are the fixed values for this solution. Row 0 is some non-zero value and all other edges (the extra rows and columns) are zero. The computation computes the inner “rows by columns” array. Each iteration is computed by the “next_gen()” function. It computes the values from (1,1) to (rows,columns) and it returns the maximum \( \Delta_{i,j} \) for that iteration. Notice, this removes any conditionals checking to see if the computation is at the edge of the array because the computed indexes range from (1,1) to (rows,columns) but the arrays have indexes of (0,0) to (rows+1,columns+1) so when computing the location (i,j), adding and subtracting 1 from the index will never be outside the array. The main program allocates two arrays and computes the value alternative from one to the other. This is done by just swapping pointers to the two dimensional arrays that are allocated as a single dimensional array.

Notice that jacobi.c does command line “option” processing using the getopt(3) library function. For this assignment you will be adding a few options to this program and you are to add them to the getopt(3) processing.

The following are the tasks you need to do for this assignment:
• Create a jacobi directory under your csci347_s18 gitlab project and copy jacobi.c unmodified into this directory. Commit it and push it.

• (80 points) Copy jacobi.c to a new file named pt-jacobi.c and modify that version do the computation using Pthreads. (Do not break this program up into multiple files. My grading script will be using that name and compiling only a single file. My script will ignore any Makefile. You may add a Makefile so you need only type make, but it must be compilable using just the pt-jacobi.c file.) First, add a new flag to the getopt(3) processing for the flag “-n num_threads”. The default should be 8. Don’t forget to update the “usage()” function.

Change the program to do the computation using threads. You will need to change the “/* Compute loop */” in main(). It needs to be completely rewritten to start num_threads threads and then join all of them. The actual “compute while loop” in main() needs to be moved into the thread body. The new thread body will need to do a similar loop except the threads will have to cooperate to compute a “global $\Delta_{i,j}$”. The thread body will essentially be “in a loop, compute my part of the computation, cooperate with all other threads to compute the global $\Delta_{i,j}$, decide to continue the loop or exit the thread.” This cooperation is key to a correct solution to this problem. All threads must make the same decision to continue or quit. Also, breaking up the compute loop into each of the thread bodies will require the rewrite of the “next_gen()” function. The “next_gen()” must know which thread called it and how to do the computation for just that thread. Notice, the next_gen() function can still return the “local $\Delta_{i,j}$” for just the calling thread. Here are some issues you will need to think solve in this assignment:

− How to equally divide up the work between the threads.
− How to make sure threads are all working on the same iteration at the same time.
− How to determine if an iteration met the tolerance and the computation is done. (Or stated differently, how can all threads cooperated to compute the global $\Delta_{i,j}$.)

• Change both jacobi.c and pt-jacobi.c to take a new flag, “-T”, that asks for the program to time the computation loop. It should report both clock time and CPU time. (hint: man clock, man gettimeofday)
The time should be reported for just the computation loop. Do not include the time for the initialization step.

- Let $T_s$ be the “wall clock” time of the serial solution. Let $T_t$ be the “wall clock” time of the threaded solution. The “speed up” is defined as $T_s/T_t$. First, compare the threaded solution’s time compared to the original solution. What should you expect? Next, find the speedups, 2 threads through 16 threads. You can use the machines in CF414 and CF165 to do your timings. Make sure nobody else is using the machine before you do your timings. htop(1) can show you how much each CPU is being utilized. You can ssh into the lab using the name cf414-01 to cf414-20 and cf165-01 to cf165-26.

- (60 points) Write a report of between 750 and 1000 words, where figures do not count toward word count, describing your experiments in speed-ups. It should have a title line, an author line, and cover the following points:
  1. Your PThreads implementation. Give a high level description, don’t quote code.
  2. Your experiments and how you conducted them.
  3. Your results. Graphs are great things to include. (man gnuplot)
  4. An explanation of your results and if things were not as you expected, explain why.
  5. A final conclusion paragraph.

Your paper needs to have both source (.tex, .ods, or whatever format you use) as well a final .pdf document and they need to be in your git repo in the jacobi directory.

- Turn-in will be only by gitlab. When you are done, branch a6. The time you branch a6 will be used as the you turned in a6. Any commits on branch a6 will be used as the final turn in time. This assignment must be “turned in” by the beginning of your class section on the day due or it will be considered late. A late assignment may be turned in up until the beginning of the final. 10 points are for following directions, turn-in, and so forth.

- In debugging, you should verify that your threaded solution calculates essentially the same as the sequential version by computing a 10 by 10 size to a tolerance of 0.001 by both versions and then verifying that
the values between the sequential and threaded versions differ by at most 0.0005.

Notes:

• You must not create a thread for each iteration. You must create exactly num_threads threads and they all must run for the full duration of the program’s run.

• Do not worry about timing until you get the computation working.

• Each thread should do about equal work. For an n by m problem, each iteration should compute n*m new values. So each thread should compute about num_threads/(m+n) for that iterations. Some threads should compute one more value than the other threads.