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 “mm.c” from the /home/phil/public/csci347 directory on the lab machines and modify it to use Pthreads so it can run faster.

In this program, there are two primary routines for which you need to add thread processing. “MatMul” multiplies two matrices producing a third. “MatSquare” multiplies a matrix by itself squaring the matrix multiple times as requested by the arguments. The MatMul uses the standard $O(n^3)$ algorithm. Currently, “MatSquare” uses “MatMul” for the actual matrix multiplies. All actual computation will be moved out of those two functions and those two functions just start and manage the threads that do the actual computation. In the threaded version, “MatSquare” will not call “MatMul” because the threads in “MatSquare” need to live across matrix multiplies and “MatSquare” needs to provide proper thread synchronization for correct squaring.

Notice that mm.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:

- (5 points) Create a “mm” directory under your csci347.s20 gitlab project and copy mm.c unmodified into this directory. Commit it and push it.

- (90 points) Copy mm.c to a new file named pt-mm.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-mm.c file. DO NOT #INCLUDE C FILES. Now, 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 both MatSquare and MatMul using threads. Again, each function will not do the computation, but create threads
to do the computation. To correctly complete this assignment you need to deal with the following issues:

- MatSquare and MatMul prototypes must remain the same except adding a “threads” parameter. No global variables may be used for any code in or called by MatSquare or MatMul. All data for the thread main must be passed via the “void *” parameter to thread main.

- How to equally divide up the work between the threads. For this program, the “minimum work element” may be a single inner product of a row times a column. So, for $n$ by $m$ matrix result using $t$ threads, each thread would compute $n \times m / t$ values. In the case that $n \times m$ is not evenly divisible by $t$, the first $(n \times m) \% t$ threads need to compute one more result than the last threads.

- In MatSquare, all threads have to be working on the same matrix multiply. This requires thread synchronization to block the early thread finishers until all threads have completed the current matrix multiply before moving to the next matrix multiply.

- Both MatMul and MatSquare must start the threads and join the threads before returning.

- Change both mm.c and pt-mm.c to take a new flag, “-T”, that asks for the program to time the computation. This would be the time taken by main function’s call to “MatMul” or “MatSquare”. In the case of “MatSquare”, multiple matrix multiples should be timed. It should report both clock time and CPU time. The only changes to main() should be for added timings. main() should not create any threads. (hint: man clock, man gettimeofday) The time should be reported for just the computation calls. Do not include the time for the initialization step.

- Let $T_s$ be the 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 reached by using labs.cs.wwu.edu or linux.cs.wwu.edu. 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 cf162-01 to cf162-24, cf164-01 to cf164-24, cf165-01 to cf165-26 and cf167-01 to cf167-24. To do this, first ssh to linux.cs.wwu.edu and then ssh from there to the host you choose.

To get times large enough to check out speed ups, you will have to do large matrix multiplies. For matrix multiply, a size of 2000 by 2000 by 2000 will take around 45 seconds of time on the Linux machines in the cf1XX labs for the serial version. For squaring, squaring 6 times and a matrix size of 1500 x 1500 takes about 125 seconds.

(45 points) Write a report of no more than 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 what ever format you use) as well a final .pdf document and they need to be in your git repo in the mm directory.

- Turn-in will be by gitlab and canvas. Submit a cover sheet and source code listing in .pdf form to canvas as your turn-in time. Also, include on your cover sheet any explanations you think are needed to correctly grade your assignment. When you are done, branch a6. Any commits on branch a6 will be used as the final turn in time if it is later than the canvas turn-in time. This assignment must be “turned in” by 10:59pm on Tuesday June 6. 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.
Notes:

- You must create exactly num_threads threads in the function “MatSquare” or “MatMul”, which ever one is called, and they all must run for the full duration of the function’s run (and thus for the duration of the program). No actual computation of the results may be done in these two functions. “MatSquare” MAY NOT call “MatMul” because “MatMul” creates threads. Thread body code may call common code to do the computation. In fact, that is expected that the actual code that does the matrix multiply work would be called by both the MatMul thread body and the MatSquare thread body. The MatSquare thread body would need to do synchronization between threads to make sure that the first matrix square is fully complete before the next matrix square operation starts in ANY thread.

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

- In debugging, you should verify that your threaded solution calculates the same values as the non-threaded version.