EECS 338 Homework 5: Multithreading

General requirements:

- See posted due date.
- Create a typed report for discussion questions and program output.
- Upload a single, compressed file (e.g. zip) to Canvas that contains all required files (programs + report).
- All work should be your own, as explained in the Academic Integrity policy from the syllabus.
- Include either a single makefile that compiles all programs or a separate makefile for each.

1. Write a C program that uses 2 P-threads to speed up an algorithm of your choice, such as finding the Shubert function minimum. In your report, explain the algorithm you chose and whether you are using public sample code. Each of n threads should perform 1/nth of the work. Note that you may wish to pass an argument to a thread function, and the textbook's sample code for P-threads demonstrates this. Print the local result for each thread, the overall (global) result, and the run time of the parent/server. Configure the settings such that the overall run time is *at least 1 second.* You may divide the work between the parent thread and 1 child thread or between 2 child threads. **Include the output in your report.** Below is an example of output for finding the Shubert minimum:

```
Total time was 10.65 seconds.
Local min = -62.12412
Local min = -186.73064
Global min = -186.73064
```

   NOTE: This problem serves as a preparation for problem #2 below.

2. Repeat problem #1 above using 4 P-threads for the computations. **Include the output in your report.**

3. Repeat problem #2 above using 4 threads with OpenMP and the "parallel for" clause and an array for the local results (see "openmp3.c" from the lecture examples). Remember that the array will be shared in OpenMP and does not need to be declared as a global variable (see openmp3.c). Note that "parallel for" only works with integer iterator variables, so you need to use an integer variable in the for loop (0, 1, 2, ...) and compute doubles for the x1 values by multiplying the step size. For example, using $-2.0 \leq x1,x2 \leq 2.0$ and a step size of 0.5, you would compute the following x1 values: -2 + 0*0.5, -2 + 1*0.5, -2 + 2*0.5, ...). *You do not need to print the local result for each thread (this would require extra work in OpenMP).* **Include the output in your report.**

4. Repeat problem #3 above using the "critical" clause and a single, shared variable for the overall result. Remember that the purpose of the "critical" clause is to protect the execution of any statements that should not be executed simultaneously in separate threads. This may include an "if" statement or any calculations that involve the single, shared variable. Multiple statements can be contained within a block using brackets {}. **Include the output in your report.**