Programming Assignment 2: pthreads producer/consumer

(+10% score) Early Due Date: Fri., 2/26, 11:59pm
Due Date: Mon, 3/1, 11:59pm
(-30% score) Final Due Date: Fri., 3/5, 11:59pm

Please read these instructions carefully. Your grade is based upon meeting all requirements stated in this assignment.

Background and Problem Statement

We now have a working serial producer/consumer program. Congratulations!

For lab 2:

- Using your existing serial program as a base, create a pthreads parallel version of your producer/consumer program.
- Using the owens cluster, engineer the optimal number of producer and consumer threads for your implementation.
- Submit both your serial and parallel programs to the owens cluster for performance benchmarking.

Program Requirements

All relevant requirements from lab 1, unless otherwise noted, apply to this program.

For this lab:

- Create one or more "producer" threads which will perform the initial transforms and populate the work queue as your producer function did in lab1.
- Read all input from stdin (as in lab 1) by either your producer thread(s) or by one or more dedicated input threads.
- Create one or more "consumer" threads, which will take work from the work queue and perform the final transforms as your consumer function did in lab 1.
- Write all of your results output to stdout (as in lab 1) by either your consumer thread(s) or by one or more dedicated output threads.
- You final results output should be identical to your serial program – printed on stdout in the same format as lab 1, order of output the same as the order of the input file.
- As in lab 1, use an array of struct work_entry as your main work queue.
  - Your work queue should be declared inside of your main( ) routine (i.e. it will be thread-global but not program-global).
  - You may adjust the size and organization (i.e. linear, circular, etc.) of this "queue" as you feel appropriate for your program design.
- The master thread should perform neither producer- nor consumer-specific functions and should perform only master thread functions.
  - No file I/O should be performed by your master thread.
- Do not use any program-global variables for this assignment.
- As with lab 1, your program should execute without any input parameters.

Testing and Instrumentation

- Use the same input data files as lab 1.
- Use analogous methods for instrumenting your parallel program as were used in lab 1, i.e. time(1) (the unix command) and time(2) (unix system call).
  - Report the "real" user time provided by time(1), which is the wall-clock elapsed time of your program execution.
  - Report the total execution time of all of your producer threads using time(2). Note that this provides the total "core-seconds" of time used.
  - Report the total execution time of all of your consumer threads using time(2). Note that this provides the total "core-seconds" of time used.
- Test your program using a variety of options for number of threads. Engineer the optimal number for the best performance of your program. You may or may not use the same number of producer and consumer threads.
  - for ease in development and testing, it is suggested that you provide support for a program parameter(s) specifying the number threads.
  - if you do so, you must provide a default value for this parameter (presumably resulting from your testing) so that your program runs with your preferred number of threads without any input parameter(s) provided
- Your parallel program must correctly complete all test files individually using no more elapsed time (time(1)) than your serial program.
- Your parallel program must correctly complete test file PC_data_t10000 in no more than 30 minutes.
- The top 10 fastest programs will be subjected to a stress benchmark. The top 5 from that pool will receive a score bonus.

Report Requirements

Run your parallel program against all of the test files provided. Provide a short (~2 page) report wchich includes the following:

- Report your final runtimes of your serial and parallel programs (using your optimal number of threads in the latter case).
- Provide a chart or graph which illustrates the scalability (or lack thereof) of your parallel solution. (Scalability charts the effectiveness of adding additional threads, typically a two-thread solution does not achieve a 50% reduction in time, but some percentage of that.) Note that you will need data from multiple runs of your parallel program with different numbers of threads to provide this chart/graph.
- Summarize your results and describe how they agreed with or contradicted your intuition.

Submission

- Submit your program files from an Owens login node using the OSC submit system as was done in lab1.
  - In your home directory, create a sub-directory named "cse5441_lab2" and place all of your program files, makefile and report for submission in that sub-directory. Submit that sub-directory.
  - Submit your source code (.c, .h). Do not submit binary files (.o), test data, or results of your program runs.
  - Provide a makefile for your program that runs without parameters (i.e. typing "make" alone will successfully build your program.
  - Name your executable file "lab2_<osc id>_<osu id>."
  - Use the Intel compiler ("icc") only. Your program should compile without warnings.
  - Include your report in .pdf format.
  - submit this assignment to "lab2"