Programming Assignment 2 - PThreads

Due Date: Thurs. 10/19

Assignment

Using your implementation of a model scientific computing problem from programming assignment 1, investigate threading this application with posix threads.

Part One - Disposable Threads

Implement a pthread parallel version of your sequential program using a disposable threads model which meets the following requirements:

- create / destroy your threads within the convergence loop, creating new threads for each loop iteration.
- implement the number of threads as the third program parameter.

  Your program should now execute similar to: disposable <affect_rate> <epsilon> <num_threads> < *infile*
- Name this program: <lastname>_<firstname>_disposable.c (for example: jones_jeffrey_disposable.c).
- In other respects, conform to the requirements for assignment 1.

Part Two - Persistent Threads

Implement a pthread parallel version of your sequential program using a persistent threads model which meets the following requirements:

- Move thread creation outside of the convergence loop, and add a barrier or barriers as necessary within your program to properly synchronize the threads.
- Destroy all threads only one time.
- Name this program: <lastname>_<firstname>_persistent.c (for example: jones_jeffrey_persistent.c).
- Your program should now execute similar to: persistent <affect_rate> <epsilon> <num_threads> < *infile*
- In other respects, conform to the requirements for part one above.

Testing and Submission Instructions

- Provide a single make file which will build both program versions, naming the executables *"disposable"* and *"persistent"*, as appropriate. Your makefile should execute with the command "**make**," with no parameters.
- Compile your programs, as with assignment 1, with optimizer level3 (-O3) and (-lrt) <small LRT> option. (Note: the -pthread option may be used in lieu of or in addition to -lrt on some systems.)

- Instrument your programs as with assignment 1.
- Collect timing results for both versions of your program using 2, 8, 16, and 32 threads and test input file testgrid_400_12206, using the values for epsilon and affect _rate you developed in lab 1.
- All timing tests should be done on the stdlinux accounts. Use the “top” command to help select a time when system use is low.
- Other than execution times, your parallel program should achieve the same results (final convergence values, number of iterations required) as your serial program.
- Ensure the program can be compiled with "make", before submitting.
- Create a directory "cse5441_lab2". Within this directory, place:
  - all program files (.c or .cc files);
  - makefile
  - report in .pdf format
- If you are in **12:45 section** use command: "submit **c5441aa** lab2 cse5441_lab2" to submit
- If you are in **2:20 section** use command: "submit **c5441ab** lab2 cse5441_lab2" to submit
- do not include other files (executables, etc), and do not create any sub-directories.
- Ensure that your submission files all have group read permissions.

Report Requirements

- Present your program output and timing results for the testgrid_400_12206 test file from your sequential program as well as both versions of your pthreads parallel program (for all requested numbers of threads).
- Summarize timing results, being sure to answer the following questions (at a minimum).
  - Did this program perform better sequentially or in parallel?
  - Which number of threads was most effective?
  - Which parallel version (disposable or persistent) was most effective?
  - How did your results match or conflict with your expectations?
  - Were there any unexpected anomalies in the timing information collected?
  - Which timing methods seem best for parallel programs? How does this compare with your expectations?
- If your parallel program produces results which differ from your serial version, point this out and explain.
- Submit all reports in .pdf format.