Project #2 – Threading Introduction

Due: Midnight, 10/28/16

For this project, you will need to write multi-threaded C code to calculate the dot product of two matrices. There are a few specific requirements. First, you must use matrices of at least 2,000 x 2,000 integers in size. You must initialize each element of each matrix either statically or using a random number generator. You must allocate sufficient space for the result matrix as well, although you may choose to do so either statically or dynamically.

Requirement #1

You must write a fully functional, single-threaded version of matrix multiplication in C to ensure the results calculated are correctly. This implementation should include timing immediately before the matrix multiplication step as well as another timing call after completing the multiplication and print the time difference. This will also give you a baseline from which you can measure the performance impact of your threaded version.

Requirement #2

Once you have a single-threaded version completed, you must add support for threaded execution of matrix multiplication. For this part of the project, you will include and use the Posix thread library, otherwise known as pthreads in your application. Keep in mind that this is an embarrassingly parallel problem, so the locking we've discussed to date is not required. As we discussed in class, there are multiple decomposition techniques that may be used to parallelize different algorithms. For this project, you may choose how to decompose based on the number of threads are used. Each thread will handle a subset of the total number of rows in the first matrix to calculate the values for the final result matrix. You may choose to do this in several ways. One way would be to assign consecutive rows to different threads, interleaving row assignment to threads progressively. Another would be to assign blocks of rows to each thread.

Your implementation must support a user-defined number of threads via command line argument. In other words, if a user specifies that your code should use four threads on the command line, you must use four threads. If 8 threads are specified, your implementation must use 8 threads. Your implementation must scale with the number of threads specified on the command line.

You also must keep track of timing similar to the single-threaded version as that will be required for Requirement #3.

Requirement #3

Once you've completed both single and multi-threaded version of matrix multiplication, you must complete some simple scaling experiments to measure the performance impact of multi-threading. The improvement is directly coupled with the specification of machine you are running the code on,

so please include how many logical CPUs your test machine has. This should become obvious with the results, but please include it in your final deliverables. You will measure the total runtime for the matrix multiplication step for 1, 2, 4, 8, and 16 threads and normalize the results against the single-threaded case (e.g. 1-thread) to show how your implementation scales as more threads are used to increase parallelization.

Final Deliverables:

- C-based matrix multiplication (single threaded and pthread-based multi-threaded). A single, unified version for both is fine (and actually preferred).
- Makefile to build your submission
- Specification of machine on which you tested your code, including how many CPUs you are running on - Are those CPUs standard cores or SMT/Hyper-threaded cores?
- Performance scaling results for 1, 2, 4, 8, 16 threads. Your speedup should be relative to the single-threaded case. Please include this in the form of a chart.
- Summary of challenges
- Answer the following questions:
    - How might you further improve your implementation to achieve even higher performance?
    - Why do we not need any locking for this project?

Group dynamic report (if needed).