LEBANESE AMERICAN UNIVERSITY
SCHOOL OF ARTS AND SCIENCES
DEPARTMENT OF COMPUTER SCIENCE & MATHEMATICS
BYBLOS

CSC 447: Parallel Programming for Multicore and Cluster Systems
Instructor: Haidar M. Harmanani
Spring 2022

Lab 5
Parallel Quick Sort

Due: March 31, 2022

Iterative Quicksort

The Quicksort algorithm works by repeatedly dividing unsorted sub-arrays into two pieces: one piece containing the smaller elements and the other piece containing the larger elements. The splitter element, used to subdivide the unsorted sub-array, ends up in its sorted location. By repeating this process on smaller and smaller sub-arrays, the entire array gets sorted. The typical implementation of Quicksort uses recursion. The attached implementation replaces recursion with iteration by using a stack. It manages its own stack of unsorted sub-arrays. When the stack of unsorted sub-arrays is empty, the array is sorted.

Instructions

1. Download and install the QSort2 directory holding the source file Quicksort.c.
2. Compile and run the serial application to verify that the algorithm correctly sorts the elements of the data array generated within the code.
3. Add the appropriate OpenMP pragmas and clauses to implement an efficient parallel solution within the code.
4. Compile the program. Be sure to include the OpenMP compiler flag.
5. Execute the parallel program with 1 and 2 threads. Do you notice a difference in the execution time? Try different data sizes and different numbers of threads, up to the number of cores on your system.
6. Check the program outputs to verify they are the same when using multiple threads. If you are getting results that are not consistent with the serial execution, there is likely a race condition in the application. You may wish to review previous parts of the lecture series for a solution to any race conditions.

Implementation Hints

It is recommended that you attempt to parallelize the iterative Quisksort code on your own. If you run into problems, these hints may give you an idea about how to proceed or restructure your solution.

- Ensure that all concurrent accesses to the shared stack of unsorted sub-arrays are done in a mutually exclusive fashion.
- Even though a thread tests and finds something on the stack, at the point that the thread is ready to remove something from the stack, the stack may be empty. Your code should not attempt to remove an item from an empty stack and, consequently, should not attempt to partition an "empty" sub-array.
- In serial, the empty stack signals the completion of the algorithm. With multiple threads, an empty stack most likely means that threads are working to partition sub-arrays. Once the partitioning has been completed, the shared stack will hold more index pairs. Your parallel code must use some other means to determine when all elements have been moved to their final sorted position.