Lab #14
CSCI 201

Title
Parallel vs Multi-Threaded vs Single-Threaded Sequential Search

Lecture Topics Emphasized
Multi-Threading
Parallel Computing

Introduction
When given an array of random numbers and asked to determine if a number exists, a sequential search is typically performed. With n elements in the array, the running time is O(n). Although we can't improve the algorithmic running time, we can hopefully improve the actual running time through parallel programming.

Description
Create an array with 100,000 random integers in the range from 0 to 10,000,000. Duplicate values are acceptable, so no need to do any extra processing when creating the array. Generate one more random integer number between 0 and 10,000,000 to try to find in the random array. You will perform a linear search on this array three times – once in a single thread, once with multiple threads, and once with parallel threads. So that we can compare the actual runtimes, make sure you are using the same array – in other words, do not regenerate the array before searching in each of the steps.

For the single-threaded search, you will perform a basic linear search. When you find the value, you should print out the location in the array and then the amount of time that elapsed.

For the multi-threaded search, you need to split the array into smaller sub-arrays. You shouldn't actually create new arrays of smaller size, but instead you should pass the indexes into your thread's constructor and only search the sub-array in that thread. You should also pass the start time into the constructor, and of course the random number for which you are searching. Start off with 4 threads with 25,000 elements searched in each. When you find the value, print out the location in the array and the amount of time that elapsed. Note that your other threads will probably still be running, but we don't care about the time for all of the threads to complete – we only care about the time it takes to find the random number.

For the parallel search, you will split the array into smaller sub-arrays similarly to how you did it with the multi-threaded search. Each parallel thread will also need the start index, end index, random number, and start time. Start off again with 4 threads with 25,000 elements to be searched. When you find the value, print out the location in the array and the amount of time that elapsed. Again, we don't need all of the threads to complete before printing the time, so print out the time as soon as you find the number.

Our hypothesis here is that the multi-threaded search should be the slowest, followed by the single-threaded search, and the parallel search should be the fastest. This would be true in the

worst case, which would occur when the number for which we are searching is not in the array. However, if we are looking at the average case, our hypothesis may not hold true.

After you get the program working for all three searches, modify the number of threads that are used in the multi-threaded search and the parallel search to see if you can get the code to execute faster.

Also, put the entire program into a loop that iterates 10 times. On each iteration, generate a new array of 100,000 random integers and a new number for which we are trying to find. Get the average of the actual runtime for each of the searches and see if those numbers are more in-line with our hypothesis.

Grading Criteria

Labs are not graded based on any given criteria but are instead graded on effort and attendance. If you arrived to lab within the first 10 minutes and worked on it the for the entire duration of the lab, you will receive full credit regardless of whether you completed it. TAs will not grade labs until after at least half the lab period has elapsed. Use the lab time as an opportunity to more fully understand the course material and ask your TA questions.