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
In this lab, you will write a linear search program in three different ways, namely, single-threaded, multi-threaded, and parallel. Each approach will be a separate class, and we will be writing a helper class later to run the three classes.

For the single-threaded search, you will perform a basic linear search. When you find the value, you should return the index (-1 if not found) and then print out the amount of time that elapsed.

For the multi-threaded search, you need to split the array into smaller sub-arrays. Instead of creating new arrays of smaller size, you should pass the start and end indexes into your thread’s constructor and only search the specified subarray in the thread. Start off with splitting the array into 4 subarrays, and then instantiate a new thread for each subarray where you will do a linear search. If you find the value, return the index, and then calculate the total 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 the threads to complete – we only care about the time it takes to find the random number. If you didn’t find the number, print out the total time for all the threads to complete.

For the parallel search, you will split the array into smaller subarrays similarly to what you did in the multi-threaded approach. Each parallel thread will also need the start index, end index, target number, and start time. Start off again with 4 threads. When you find the value, return the index, and then calculate the amount of time that elapsed. Again, print out the time as soon as you find the number - we don’t care if all the threads are finished. We only care about all the threads finishing if the number is not in the array.

Now let’s create the helper class. In the helper class, instantiate an array with 100,000,000 unique integers. (One way to come up with unique integers is to make the values be equivalent to the indexes initially, then shuffle the array.) Generate a random integer between 0 and 100,000,000-1, which will be your first target number. Create another integer that is NOT in the array to be your second target number. Use the three classes above to find your target numbers and check if they have all returned the same index by printing them out. Make sure you are using the same array, so that we can compare the run times. Do not modify the array since we want to compare the running times on the same array.
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.

Theoretically, 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 occurs 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.
Grading Criteria
Labs are graded based on your understanding of the course material. To receive full credit, you will need to 1) complete the lab following the instructions above AND 2) show your understanding of the lab material by answering questions upon check-off.

If there is a discrepancy between your understanding of the material and your implementation (i.e. if your code is someone else’s work), you will receive a grade of 0 for the lab. Please note, it is the professor’s discretion to report the incident to SJACS.

Instructors, to ensure consistency across all lab sections, please strictly stick to the following criteria:

1) Single-threaded Search
   a) 0.10% - Linear search returns the index and prints out the elapsed time
   b) 0.05% - Code is complete but has bugs
   c) 0% - Code is incomplete (no partial credits)

2) Multi-threaded Search
   a) 0.20% - Linear search returns the index and prints out the elapsed time upon finding the index
   b) 0.15% - Code is complete but has bugs
   c) 0.10% - Student is on the correct path, and has completed more than 50% (The skeleton code on the course website does NOT count towards the 50%)
   d) 0% - Student completes less than 50% (no partial credits)

3) Parallel Search
   a) 0.20% - Linear search returns the index and prints out the elapsed time upon finding the index
   b) 0.15% - Code is complete but has bugs
   c) 0.10% - Student is on the correct path, and has completed more than 50% (The skeleton code on the course website does NOT count towards the 50%)
   d) 0% - Student completes less than 50% (no partial credits)

4) Helper Class
   a) 0.10% - Helper class creates the specified array, generates two target numbers, calls the three classes with the two numbers, and compares the returned indexes
   b) 0.05% - Helper class creates the specified array, and generates two target numbers
   c) 0% - Otherwise (no partial credits)
5) Check-off Questions: Please randomly select one question from each section. Each question is worth 0.07%, and the check-off questions are worth 0.20% total.

Question 1

A) How do we generate unique integers?
B) How do we calculate the elapsed time? Do all threads have to finish? Why or why not?
C) How do we make sure that the number generated is in the array?

Question 2

A) Why do we search for a number that is NOT in the array?
B) In multi-threaded and parallel searches, if the target number can’t be found in the array, when do we stop the timer?
C) How does a large array (with 100,000,000 elements) help us test the hypothesis?

Question 3

A) Instead of searching for a number that is NOT in the array, what could we do instead?
B) Why do we want an array with unique integers? What could happen if the integers in the array are not unique?