T2,
2020

SIT315 - M2.T2C

CONCURRENT PROGRAMMING

CAMERON BOYD | STUDENT #218275923

INTRODUCTION

This document will be comparing the execution time of a sequential program and an OpenMP multithreaded concurrent program. Both programs share the exact same structure, the difference being the use of threads in the concurrent one.

Both programs will be utilising the Quicksort comparison sorting algorithm to sort a one-dimensional array.

Additionally, both the sequential and concurrent programs use the Lomuto partitioning scheme. This is known to be less efficient than the Hoare partitioning scheme but is easier to understand and implement. I chose the Lomuto partitioning scheme as it is better suited to those still learning C++.

As for testing the execution time, all times were measured in milliseconds. Moreover, there were four array sizes used for testing:

- 10,000;
- 100,000;
- 1,000,000; and
- 2,000,000.

In addition to this, each size was tested three times to gain an average of the time taken.

SEQUENTIAL VS. CONCURRENT

The following results were obtained by using two threads for the concurrent program. Note that *n* refers to the number of elements in the array. Of which, each element is a random integer between 0 and 100.

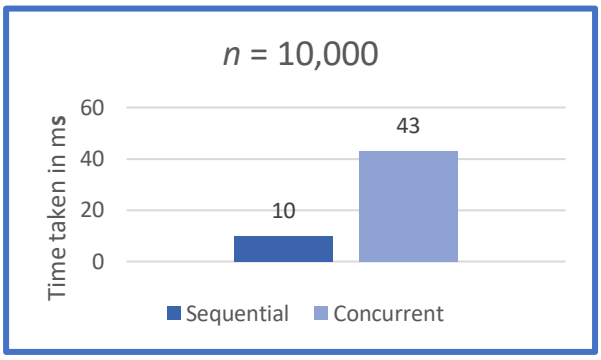

Figure 1: Array with 10,000 elements

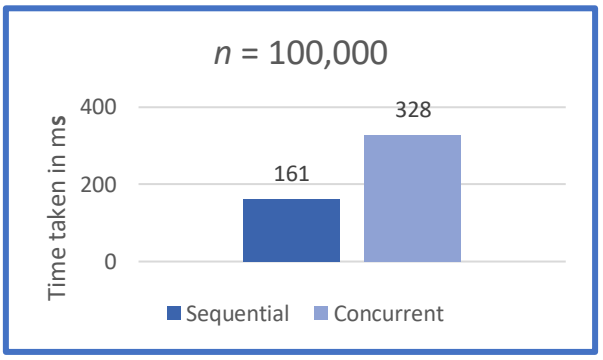

Figure 2: Array with 100,000 elements

As Figures 1 and 2 indicate, the sequential program is much faster in its execution time than the concurrent variant when the array contains 10,000 and 100,000 elements. Figure 1 shows the concurrent program taking 4.3x longer to execute than the sequential program. Figure 2 shows the concurrent program taking 2x longer than its counterpart.

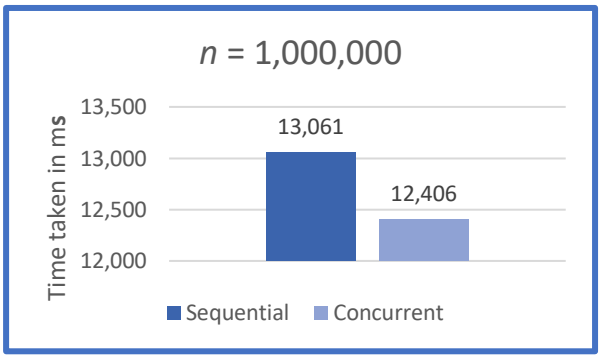

Figure 3: Array with 1,000,000 elements

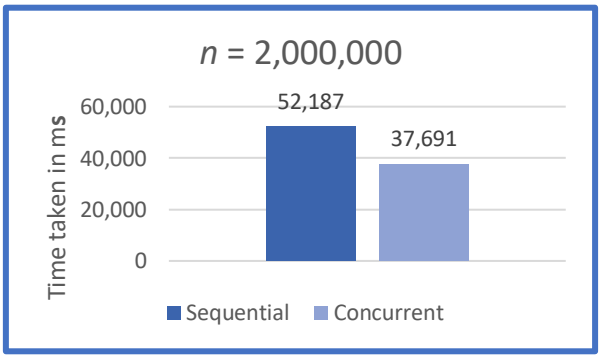

Figure 4: Array with 2,000,000 elements

Figures 3 and 4 tell a different story to 1 and 2, however. As the array grows, the concurrent program becomes increasingly faster with regards to its execution time. Figure 3 shows the concurrent program taking 0.95x as long as the sequential one. Figure 4 shows the concurrent program taking 0.72x as long as its sequential variant.

CONCLUSION

The results of this test coincide what was found in M2.T1P:

- Small tasks are better suited to a sequential program; and
- The larger the task, the more effective multithreading becomes in reducing execution time.

As was the case with M2.T1P as well, these results may be inconclusive as I am not the best C++ programmer and have very little experience with multithreading.