SIT315 M2.T2C: Complex Threading With OpenMP

Codey Funston | S222250824

Decomposition and Performance Comparison

I decided to implement a quicksort algorithm that used recursion as that seemed to work better conceptually with parallelising the program. The function runs its body and then has two recursive calls, splitting the data in two based on a random pivot element. I used OpenMP tasks to assign threads to the recursive calls since their dynamic scheduling is very good for this sort of task, especially for load balancing when quicksort() calls are done with unbalanced data partitions.

The runtime of the sequential program isn’t that bad given that the time complexity for quicksort is . For very small data input the sequential version runs faster, for example with a vector of 2000 integers, over 100 tests for each program the average for the sequential version was 102.27 microseconds, compared to the OMP version’s average of 179.88 microseconds. This is because the threading overhead is much too high. To help reduce the overhead I added a condition based on testing output to only assign a thread to a recursive call if the section of data being worked on consisted of more than 1000 integers.

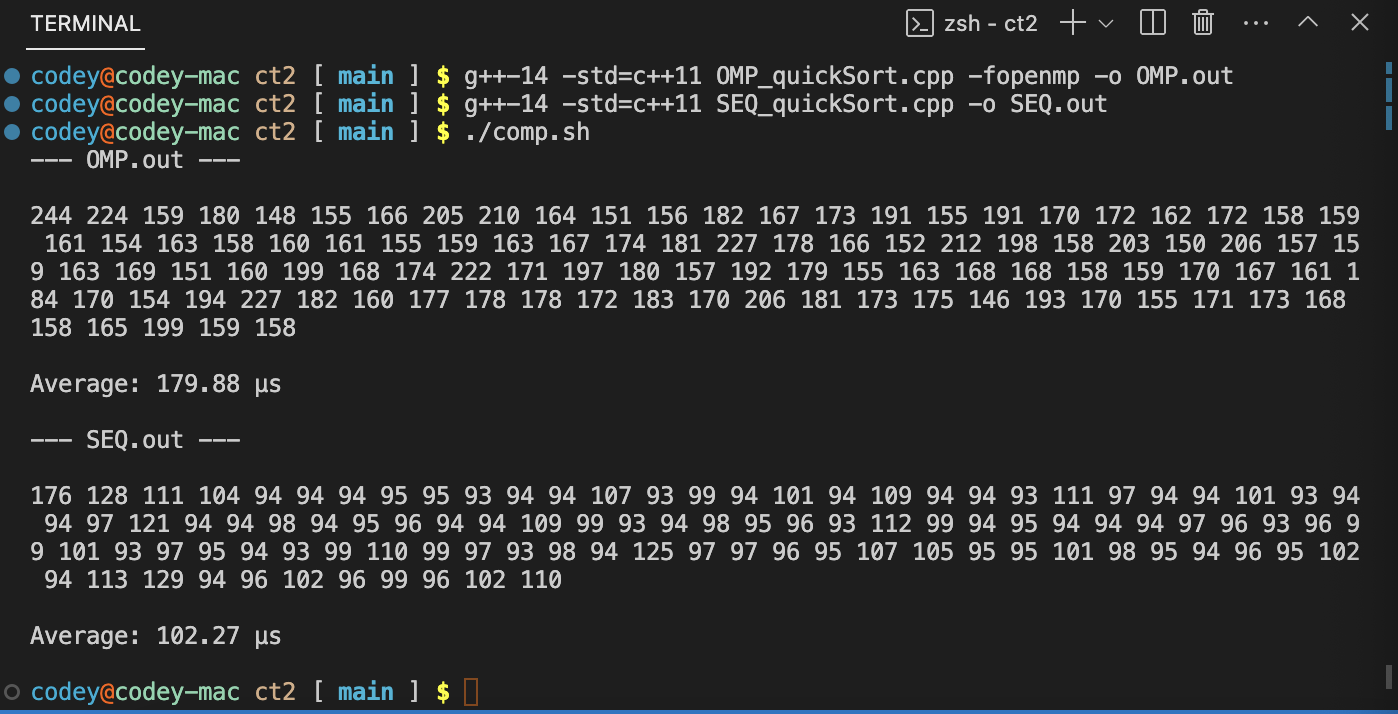
The core distribution was nice and even throughout each test, usually looking similar to this clipping from a test of the OpenMP version with 20 million integers:

A white box with blue and black stripes

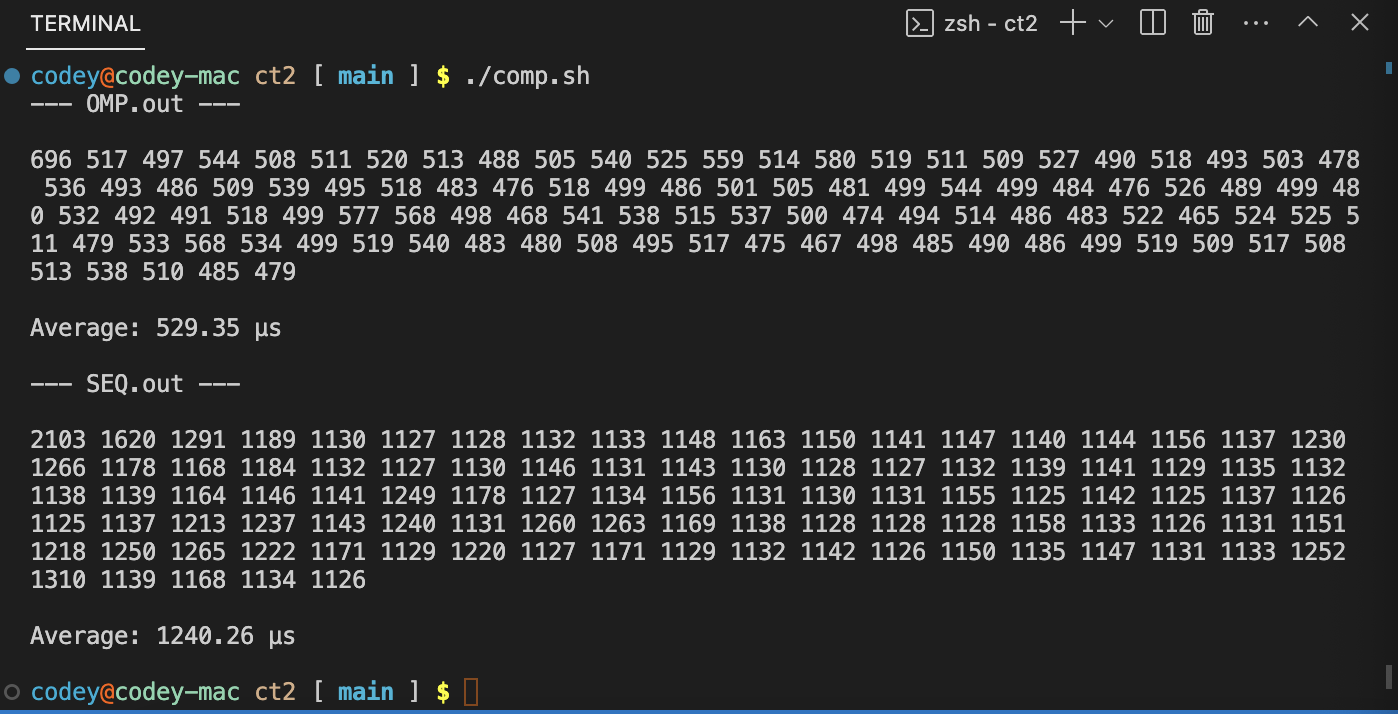
Description automatically generated

Some Results and Corresponding Input Data Size

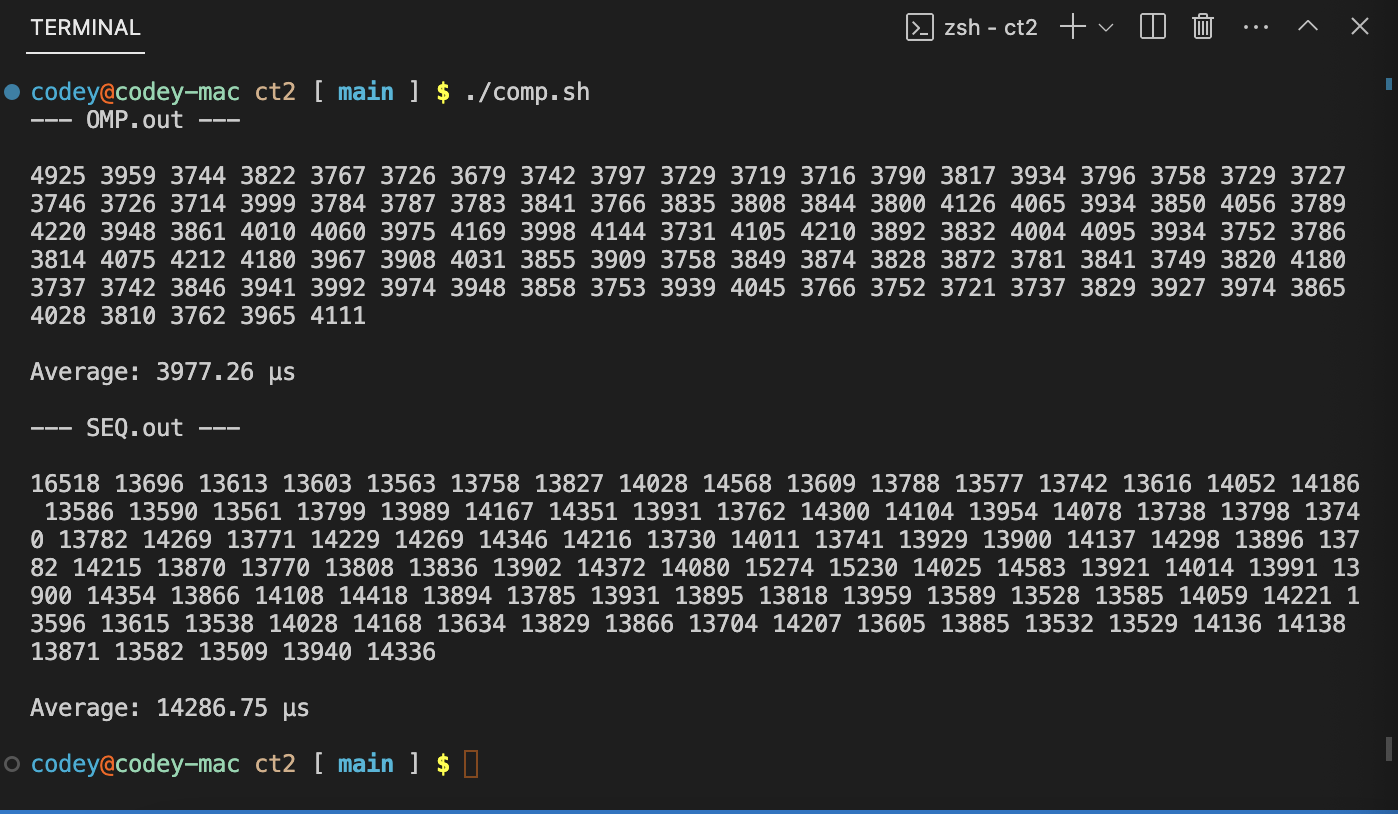
* 2,000



* 20,000 (OMP is now more than twice as fast)



* 200,000 (OMP has an incredible jump, at more than 3 times faster)



* 2000000 (another speed increase with OMP now more than 4 times faster)

A screenshot of a computer

Description automatically generated