COMPLEX THREADING

Student name - Nimduli Demin Achchi A D

Student number - 219078029

QUICKSORT LINKS

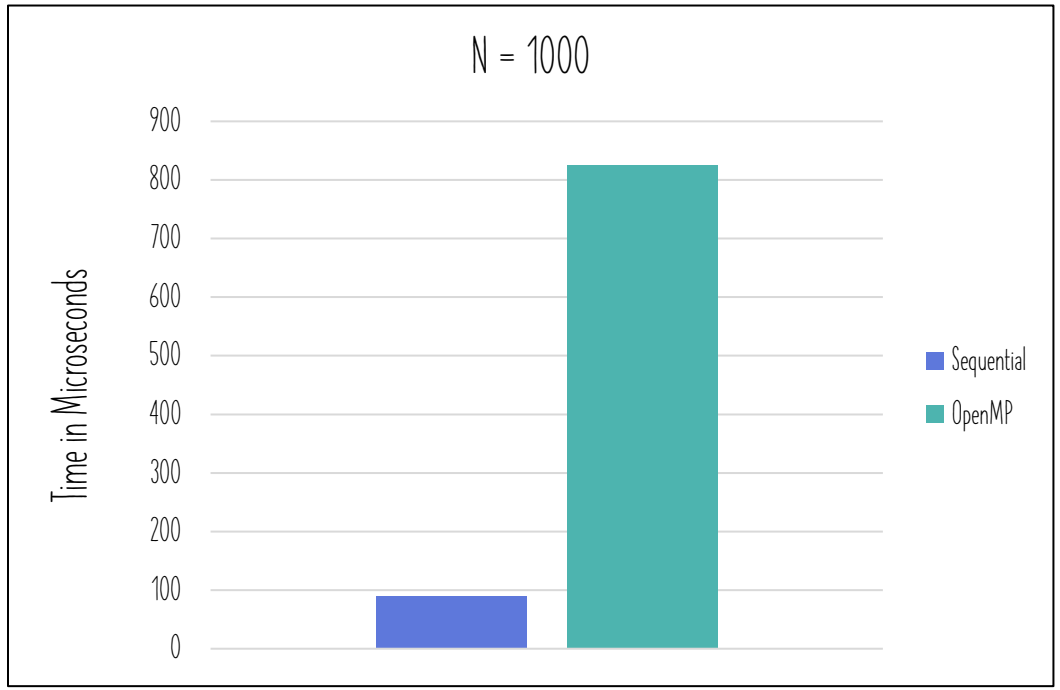
Git hub link to Sequential code -

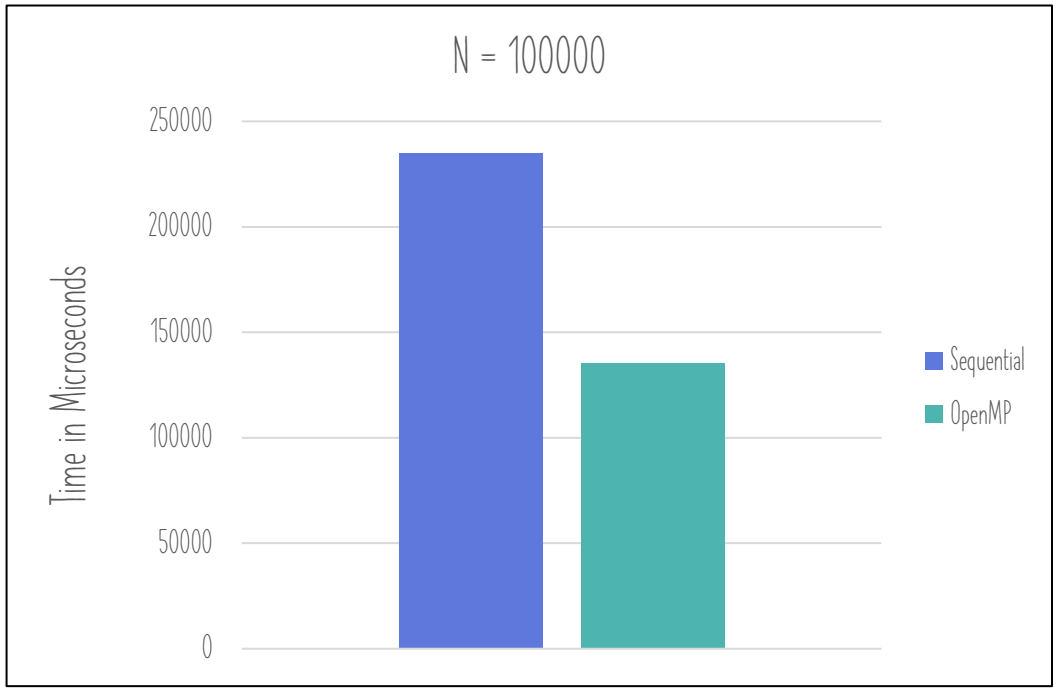
https://github.com/NimduliAthukorala/SIT315/blob/main/Module%202/T2/QuickSort_Seq.cpp

Git hub link to Parallel code -

https://github.com/NimduliAthukorala/SIT315/blob/main/Module%202/T2/QuickSort_OMP.cpp

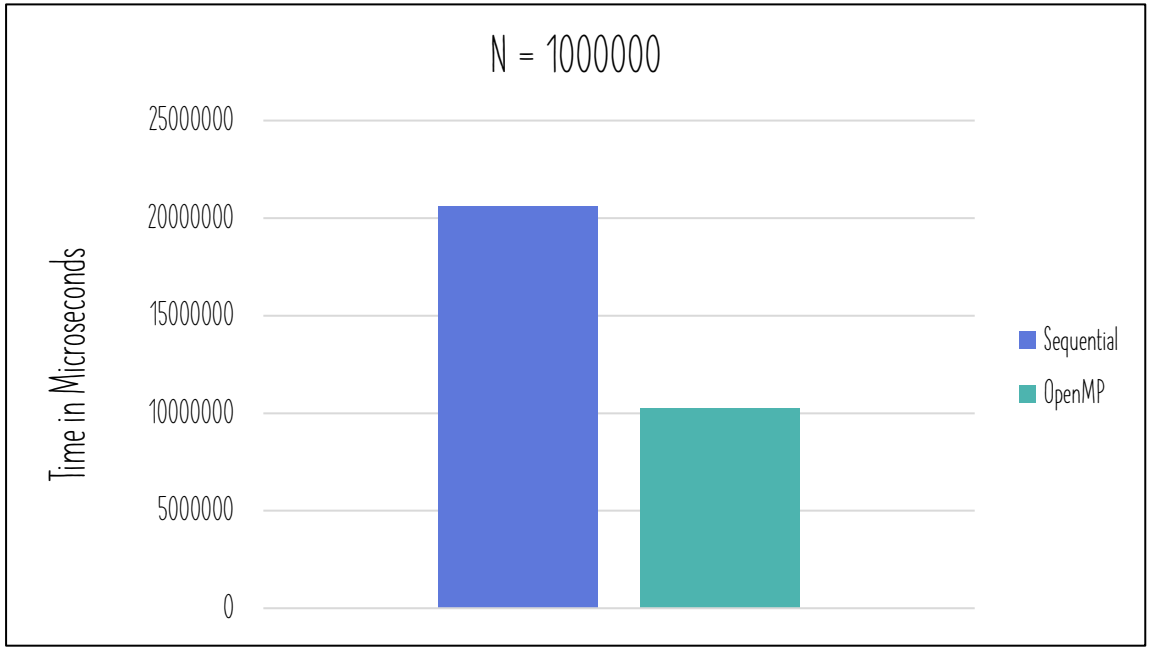
COMPARE THE PERFORMANCE OF SEQUENTIAL AND OPENMP





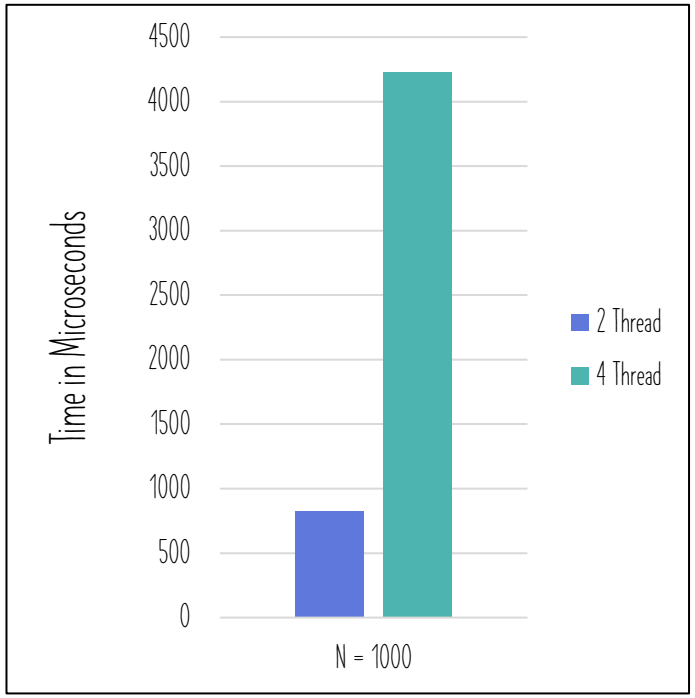
As shown above the two graphs compare the performance of Sequential and Parallel programs using OpenMP for different values of N. Here N is the size of the vector. As we can see the performance of Sequential is better for very small vectors compared to a larger vector where OpenMP is faster. This is using 2 threads.

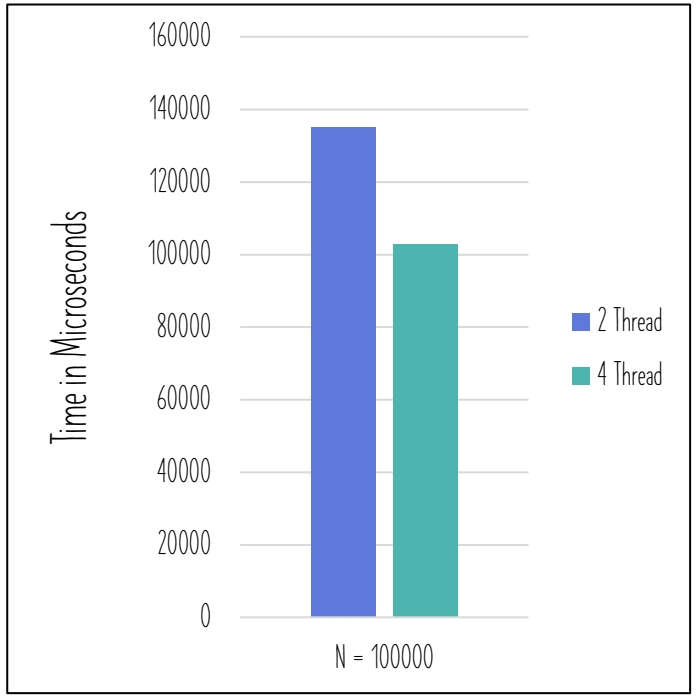
COMPARE THE PERFORMANCE OF SEQUENTIAL AND OPENMP

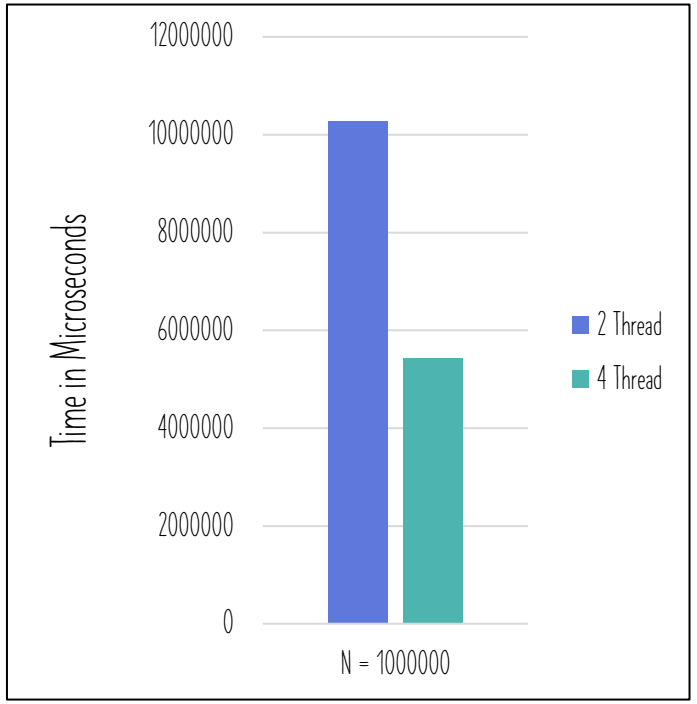


As shown above the two graphs compare the performance of Sequential and Parallel programs using OpenMP for different values of N. Here N is the size of the vector. For an extremely large vector Parallel using OpenMP is up to 2 times faster.

COMPARE THE PERFORMANCE FOR DIFFERENT THREADS IN OPENMP







As shown above the graphs compare the performance of Parallel programs using OpenMP for different values of N. Here N is the size of the vector. This also looks deeper into the performance and thread number. Up till 1000 using 2 threads is better compared to 4. Thus, indicating that it is faster to use 4 threads for large vectors.

CONCLUSION

- After testing several methods in OpenMp I identified that using the 'single' directive with 'nowait' clause made the program faster.
- This is because the use of the nowait clause ensures that other threads will not wait instead enter the next recursion immediately.
- In conclusion for small vectors of approximately less than 10000 elements it will be faster to use sequential.
- For larger vectors it is best to use parallel using OpenMP as provided in the code