Performance Analysis

Sequential QuickSort

The sequential version of QuickSort recursively sorts the sub-arrays on a single thread. Its performance relies purely on the processing power of a single CPU core.

Parallel QuickSort

The parallel version aims to improve performance by running two recursive calls (left and right sub-arrays) in parallel using multiple threads. However, there are overheads involved in managing threads and recursion.

Results

The following results were observed when sorting an array of 10,000,000 random integers

```
[Running] cd "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.T2C\" && g++ -std=c++17 sequential_quicksort.cpp -o sequential_quicksort && "f:\Deakin
Year3 Tasks and Assigments\SIT315\M2.T2C\"sequential_quicksort
顺序版 QuickSort 运行时间: 5023 毫秒

[Done] exited with code=0 in 5.843 seconds

[Running] cd "f:\Deakin Year3 Tasks and Assigments\SIT315\M2.T2C\" && g++ -std=c++17 parallel_quicksort.cpp -o parallel_quicksort && "f:\Deakin
Year3 Tasks and Assigments\SIT315\M2.T2C\"parallel_quicksort
并行版 QuickSort 运行时间: 4876 毫秒

[Done] exited with code=0 in 5.669 seconds
```

Program Version	Execution Time (ms)
Sequential	5023
Parallel Version	4876

The parallel version of QuickSort achieved only a marginal improvement over the sequential version. This indicates that while the algorithm is well-suited for parallel execution, the gains are heavily dependent on thread management, recursion depth, and hardware resources.

Github Link
https://github.com/Lonely-DM/SIT315/tree/main/M2.T2C