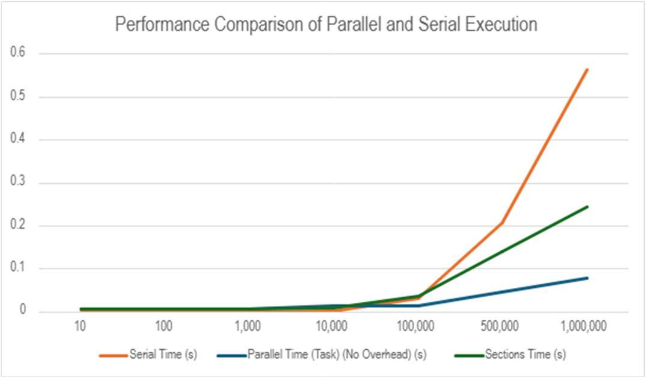
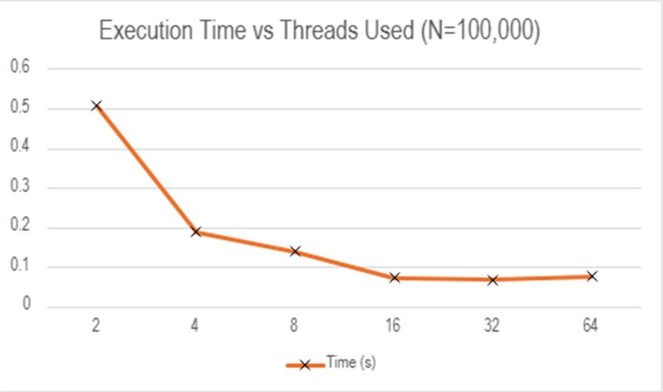
Q1) Quick Sort Implemented on Cluster and Analysed by Subham Gaurav





***Note - Without Overhead meaning :** Only creates parallel tasks if the subarray size is greater than 1000. This avoids the extra thread management and synchronisation.
****Note –** Please refer to the code footer for the actual time measurements.

A. Small Array Sizes (10 – 1,000)

- **Serial time is very low**, close to microseconds. **Parallel Time (No Overhead)** is much higher than serial time. **Overhead dominates** the execution time; e.g., At size 10: **Speedup < 1**: This means parallelization is **not beneficial** for small sizes.

Conclusion: For small tasks, the overhead cost outweighs parallel gains. **Parallelism is inefficient here**.

B. Medium Sizes (10,000 – 100,000)

- **Serial time increases gradually**, but **parallel (no overhead)** time doesn't grow proportionally — a good sign.

Conclusion: The algorithm parallel logic scales well (speedup), but **overhead becomes problematic**. Indicates **scalability is logical** when implemented well

e.g., At 1000 Array Length Serial = 0.030481s **Parallel with no overhead** (0.014640 s ⇒ **Speedup ≈ 2.08**) while **Parallel With Overhead** = 3.702799 s ⇒ **Heavy overhead (~3.69 s)**

C. Large Sizes (500,000 – 1,000,000)

- Parallel time (no overhead) increases **slowly**, which is a good sign of **scalability**.
- **Conclusion**: The algorithm can be **very fast with parallel execution (good speedup).** But too much **extra work (overhead) makes it very slow**. This may be due **to poor thread handling, bad memory use,** or **delays in coordination.**

  e.g., At 1,000,000 elements: **Serial** = 0.563983 s **Parallel (No Overhead)** = 0.078146 s ⇒ **Speedup ≈ 7.2 With Overhead** = 55.205389 s ⇒ **Overhead** = **~55.12 s**

For Different Threads

Assuming serial time ≈ **0.508326 s (with 2 threads)**, we observe:

- **4 threads**: speedup ≈ 2.7× **8 threads**: speedup ≈ 3.6× **16 threads**: speedup ≈ 7.0×

  32 threads: speedup ≈ 7.4× **64 threads**: speedup ≈ 6.61× speedup slightly decreases due to overhead

The program scales **well up to 32 threads**, showing steady improvement.

The parallel algorithm works well and **gets faster as we add more threads, up to 32 threads**, when processing an **array of size 100,000**. After that, using **more threads doesn’t help much because** of **extra work and memory sharing problems**. This means the **best number of threads** for this task is **probably between 16 and 32.**