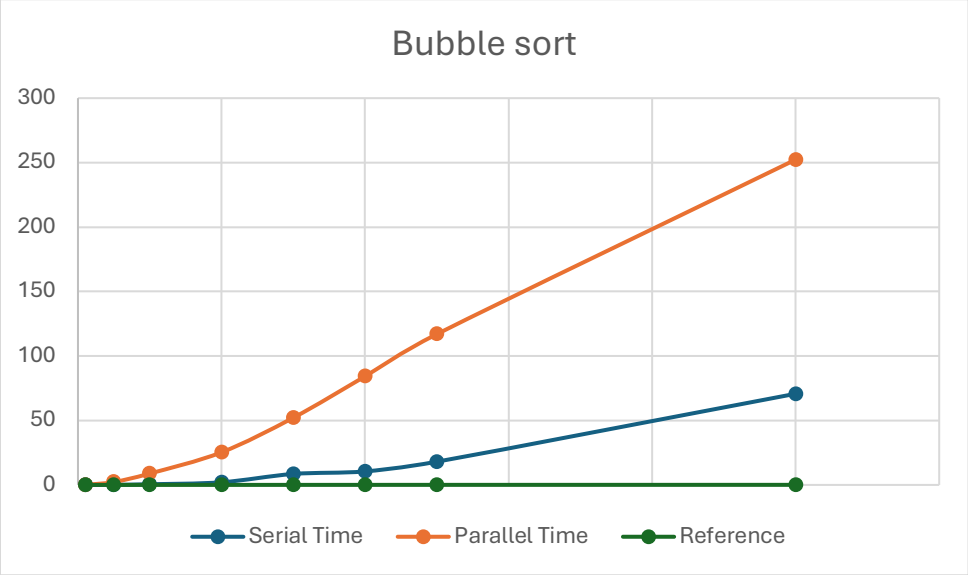
| | |
|---|---|
| Asha Shah | 800817984 |
| Prashanna Raj Pandit | 800817018 |
| Simran Basnet | 800818370 |

Results and Observation:

Bubble Sort:

| Array Size | Serial Time | Parallel Time | Reference |
|---|---|---|---|
| 1000 | 0.004 | 0.00707 | 0.003707 |
| 5000 | 0.097 | 2.275 | 0.003227 |
| 10000 | 0.531 | 8.82377 | 0.004587 |
| 20000 | 2.057 | 25.2983 | 0.009983 |
| 30000 | 8.503 | 52.0889 | 0.02889 |
| 40000 | 10.485 | 84.2099 | 0.033389 |
| 50000 | 18.01 | 116.932 | 0.043321 |
| 100000 | 70.694 | 252.343 | 0.055341 |

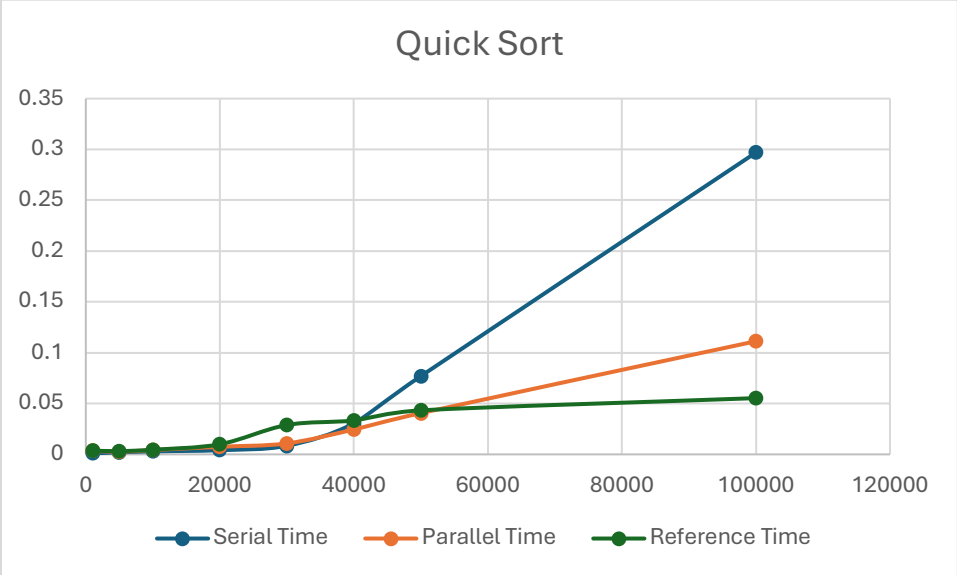


Observations:

1. Serial Time Increases Steadily:
    - The execution time for the serial Bubble Sort grows consistently as the array size increases, following the expected time complexity of $O(n^2)$. For example:
        - Array Size 1000: 0.004, Array Size 100,000: 70.694
2. Parallel Time Is Significantly Slower:
    - The parallel execution time for Bubble Sort is consistently slower than the serial execution time across all array sizes. For example:
        - Array Size 1000: 0.00707 (parallel) vs 0.004 (serial)
        - Array Size 100,000: 252.343 (parallel) vs 70.694 (serial)
    - The parallel overhead seems to dominate and increase sharply with larger array sizes, particularly with sizes above 5000 elements.

Quick Sort:

| Array Size | Serial Time | Parallel Time | Reference Time |
|---|---|---|---|
| 1000 | 0.000998 | 0.003502 | 0.003707 |
| 5000 | 0.001993 | 0.0027 | 0.003227 |
| 10000 | 0.002992 | 0.0043 | 0.004587 |
| 20000 | 0.004121 | 0.0075 | 0.009983 |
| 30000 | 0.008503 | 0.0108 | 0.02889 |
| 40000 | 0.030485 | 0.0245 | 0.033389 |
| 50000 | 0.07701 | 0.0406 | 0.043321 |
| 100000 | 0.296949 | 0.1113 | 0.055341 |

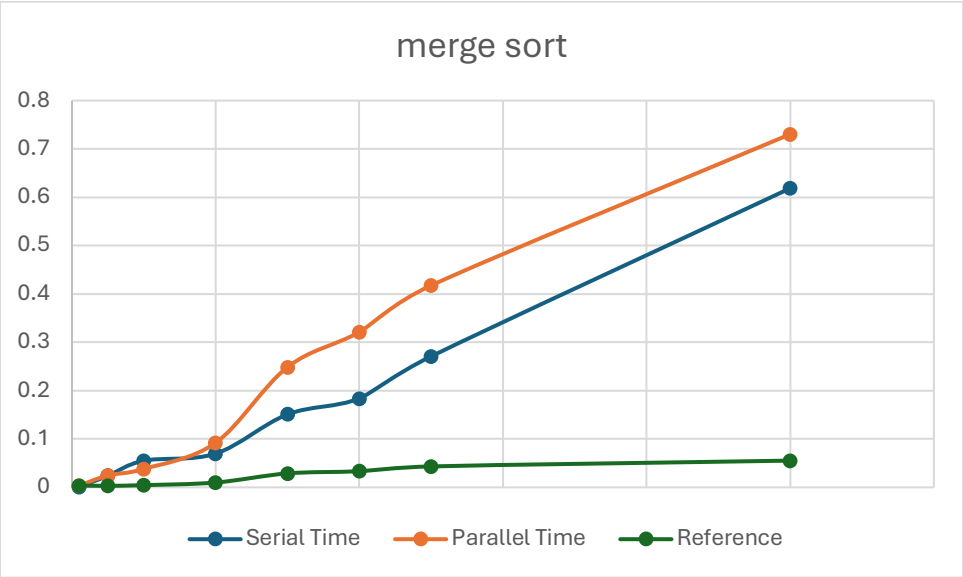



Observations:

For smaller array sizes (1000-10,000), the parallel implementation shows longer execution times than the serial version, which is likely due to the overhead of spawning multiple threads and the need for synchronization between them. The benefits of parallelism don't outweigh this overhead for smaller workloads, especially considering the operating system's thread management and scheduling costs. The parallel version outperforms the serial version as the array grows due to better CPU core utilization. However, cache coherence and memory access patterns play a role here: as the data grows larger, efficient use of the CPU cache becomes critical. In the parallel implementation, threads may compete for shared cache resources, leading to reduced speedup.

Merge Sort:

| Array Size | Serial Time | Parallel Time | Reference |
|---|---|---|---|
| 1000 | 0.000996 | 0.00361 | 0.003707 |
| 5000 | 0.023989 | 0.02467 | 0.003227 |
| 10000 | 0.055125 | 0.03808 | 0.004587 |
| 20000 | 0.069778 | 0.09213 | 0.009983 |
| 30000 | 0.150944 | 0.24878 | 0.02889 |
| 40000 | 0.183934 | 0.32111 | 0.033389 |
| 50000 | 0.270917 | 0.41764 | 0.043321 |
| 100000 | 0.61865 | 0.73046 | 0.055341 |

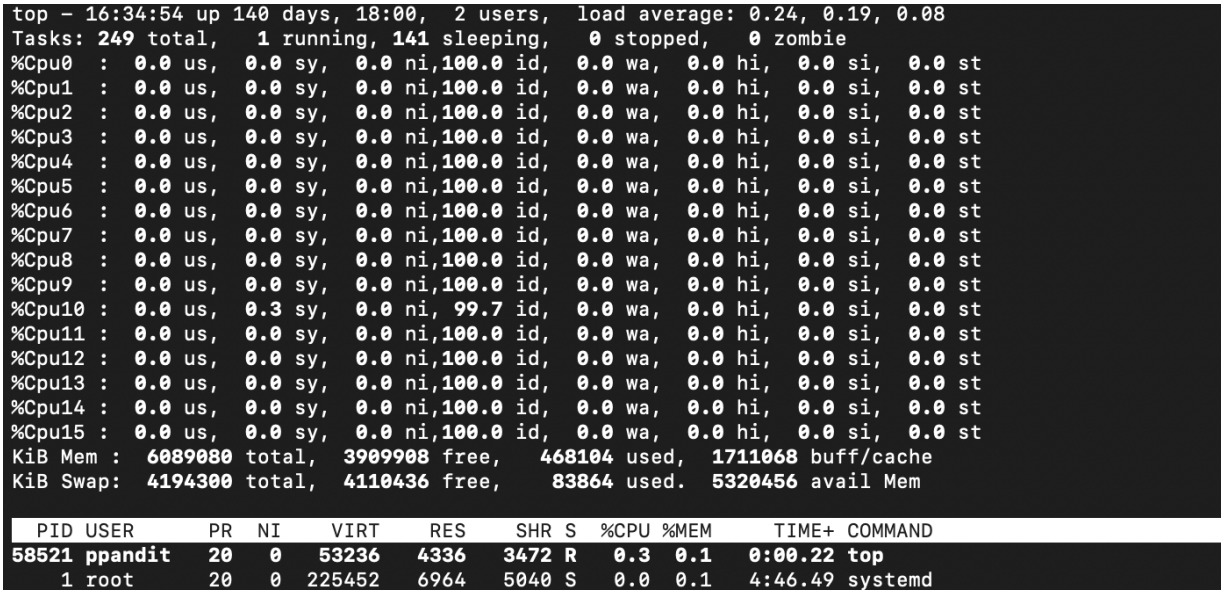



Observations:

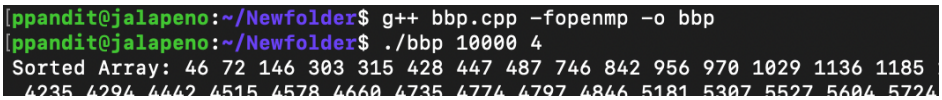
The data from merge sort output shows that for smaller array sizes (up to 10,000), the parallel version performs similarly or slightly better than the serial version, but as the array size increases, the parallel version becomes slower relative to the serial version. This is likely due to thread management overhead, where creating and managing multiple threads outweighs the benefits for smaller arrays. As the data size grows, parallelism allows better utilization of CPU cores, but issues like memory contention and cache coherence start to impact performance. The increasing gap between parallel and serial times for larger arrays suggests that the parallel version encounters bottlenecks related to memory access patterns and possibly thread synchronization.

Hyperthreading:

Turning on hyperthreading.




Running Bubble sort parallel with hyperthreading:




Output:

Execution Time: 592436 microseconds. i.e 0.592436 seconds.

Bubble sort serial: Execution Time: 1465674 microseconds

Observation:

With hyperthreading enabled on the Jalapeno system, the parallel execution of bubble sort ran much faster (0.592436 seconds) compared to without hyperthreading on local PC (8.82377 seconds). This is because Jalapeno's 16 hyperthreaded cores can better utilize idle resources, improving throughput and efficiency.

However, in serial execution, the performance with and without hyperthreading is less pronounced because hyperthreading benefits mostly parallel tasks. The results for the serial execution (1.465674 seconds on Jalapeno vs. 0.531 seconds on local PC) show that in a single-threaded context, the benefits are marginal or sometimes even detrimental due to the overhead from managing multiple logical cores without fully utilizing them.