

**PROJECT REPORT**

**Comparative Analysis of Sorting Algorithms Using OpenMP, MPI, and Serial Implementations**

**GROUP #10**

**Bisma Abid (21k-3269)**

**Wara Batool (21k-3214)**

**Syed Hadi Raza (21k-3420)**

**INTRODUCTION**

Sorting algorithms play a crucial role in computer science, optimizing the arrangement of data elements. Among these, Quick Sort and Merge Sort stand out for their efficiency and widespread use. This report delves into a comparative analysis of these two algorithms, exploring their intricacies, performance characteristics, and suitability for various data scenarios.

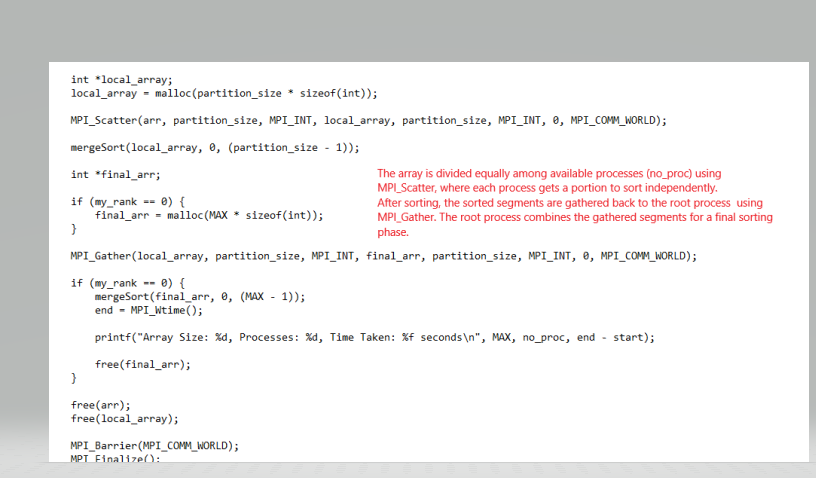
**IMPLEMENTATION**

**OpenMP Quick Sort Implementation:**

Identify parallelizable segments, like the partitioning step. Utilize #pragma omp parallel for to distribute partitioning work among threads for concurrent handling, ensuring load-balanced execution.

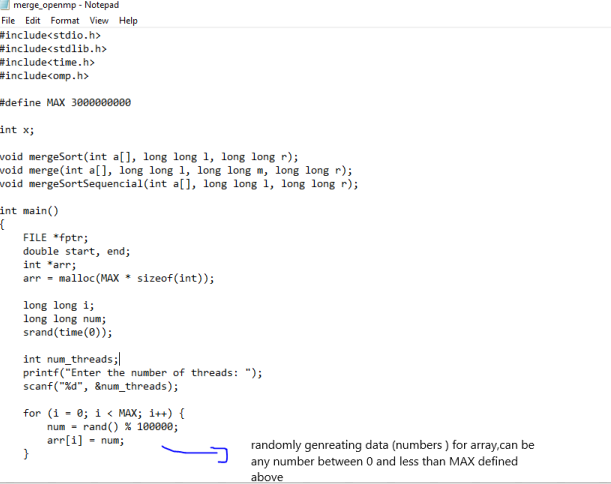
**MPI Merge Sort Implementation:**

Divide the array into segments and use Scatter operations to distribute segments among different MPI processes. Each process independently sorts its allocated segment. Utilize MPI communication primitives like MPI send and MPI Recv to exchange data between processes for merging and combining the sorted segments.



**OpenMP Merge Sort Implementation:**

Parallelize the recursive splitting and merging phases using #pragma omp parallel directives in the recursive calls. Use #pragma omp barrier directives to ensure thread synchronization, particularly during merging stages, allowing multiple threads to execute parts of the algorithm concurrently.



A screenshot of a computer code

Description automatically generated

**COMPARISON**

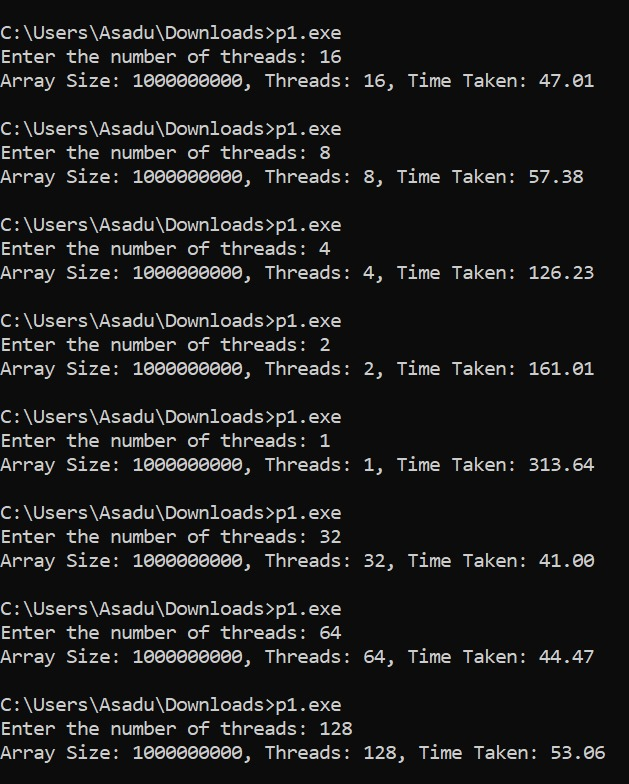
**MERGE SORT (OPEN MP)**

The table below summarizes the comparison in openmp when same data set is run on different number of threads.

|  |  |
| --- | --- |
| **THREADS** | **TIME IN SECONDS** |
| 1 | 313.64 |
| 2 | 161.01 |
| 4 | 126.23 |
| 8 | 57.38 |
| 16 | 47.01 |
| **32** | **41.00** |
| 64 | 44.47 |
| 128 | 53.06 |

**>Optimal solution at 32 threads**

**At 32 threads, the tasks are just the right size—neither too big nor too small. This balance ensures that creating these tasks doesn't take more time than actually doing the sorting work, making things efficient. More than 32 threads we are spending more time in dividing the data set rather than sorting it , because of which our time is increasing**

****

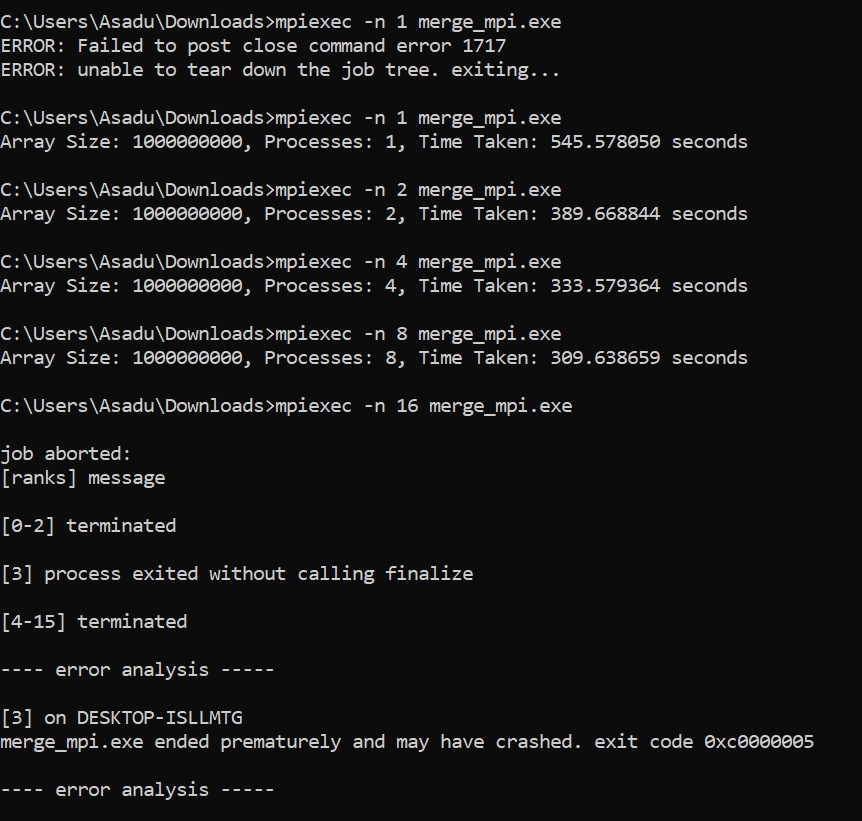
**MERGE SORT IN MPI**

The table below summarizes the comparison in mpi when same data set is run on different number of processors.

|  |  |
| --- | --- |
| **PROCESSORS** | **TIME IN SECONDS** |
| 1 | 545.578050 |
| 2 | 389.668844 |
| 4 | 333.579364 |
| **8** | **309.638659** |
| 16 | Error |

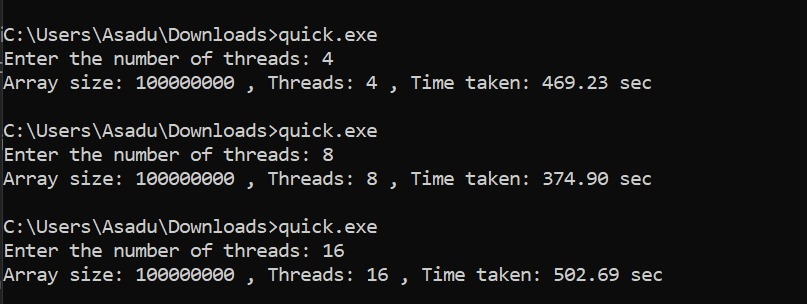
**>optimal solution at 8 processors**

**As the number of MPI processes increases, the size of the partition each process receives decreases .At 16 processors, the partition size might become too small, potentially leading to memory-related issues due to inadequate memory allocation or fragmentation**

****

**QUICK SORT (OPEN MP)**

|  |  |
| --- | --- |
| **THREADS** | **TIME IN SECONDS** |
| 4 | 469.23 |
| **8** | **374.90** |
| 16 | 502.69 |

****

**CONCLUSION**

* **MPI:** Shows good scalability initially, but the error at 16 processors indicates a limitation or bottleneck.
* **OpenMP**: Demonstrates excellent scalability up to 16 threads, showcasing efficient parallelism and significant performance improvement.

**Observations:**

* MPI's scalability might face limitations due to potential issues with workload distribution or memory constraints, evident from the error at 16 processors.
* OpenMP shows remarkable scalability up to 16 threads, maintaining consistent improvements in performance. However, the slight increase in time beyond 16 threads suggests a possible overhead limitation or resource contention.

Both MPI and OpenMP exhibit scalability in sorting performance, but OpenMP showcases more consistent and efficient scalability up to a certain point. Understanding the bottlenecks or limitations within each parallelization approach is crucial for optimizing performance.

Overall, OpenMP demonstrates more reliable and efficient scaling behavior for Merge Sort in this scenario compared to MPI, showcasing significant performance improvements with increased threads until reaching a point of diminishing returns.