Name: Pranay Chavhan **Roll No:** 41411

Mini Project: Evaluate Performance Enhancement of Parallel Quicksort Algorithm using MPI

Title:

Evaluate Performance Enhancement of Parallel Quicksort Algorithm using MPI

Problem Statement:

Sorting large datasets is a common task in computing, but traditional serial sorting algorithms become inefficient as data size increases. The quicksort algorithm is efficient but inherently recursive and difficult to parallelize. This project aims to explore the performance gain achieved by implementing quicksort in parallel using MPI (Message Passing Interface).

Objective:

- To implement the Quicksort algorithm using MPI for parallel execution.
- To evaluate the performance enhancement of parallel quicksort compared to the serial version.
- To analyze scalability and efficiency on multiple processors.

Expected Outcome:

- A working implementation of Parallel Quicksort using MPI.
- Comparison of execution time between serial and parallel versions.
- A performance graph showing speedup with increasing processors.

Theory:

Quicksort is a divide-and-conquer sorting algorithm. It works by:
1. Choosing a pivot element.
2. Partitioning the array into elements less than and greater than the pivot.
3. Recursively applying the same process on sub-arrays.

MPI-based Parallel Quicksort enhances performance by:
- Distributing subarrays to different processors.
- Each process sorts its subarray.
- Results are gathered and merged in the root process.

Pseudocode:

```
Parallel_Quicksort(A, n, rank, size)
begin
  if (rank == 0) then
    Divide array A into equal chunks for each process
```

```
        Send each chunk to corresponding process

    Receive local chunk on each process
    Perform serial quicksort on local chunk

    Gather all sorted chunks at root process

    if (rank == 0) then
        Merge all sorted chunks into final sorted array
end
```

Test Case:

Input:
Array: [34, 7, 23, 32, 5, 62, 3, 8]
Processors: 4

Expected Output:
Sorted Array: [3, 5, 7, 8, 23, 32, 34, 62]

Observation:
Compare execution time of:
- Serial quicksort
- MPI-based parallel quicksort

Conclusion:

The MPI-based parallel implementation of quicksort significantly improves performance for large datasets. By distributing workload across multiple processors, we reduce overall computation time. This project demonstrates the advantage of parallel programming in sorting algorithms and highlights MPI's capability in high-performance computing.