Final Documentation for MPI-Based QuickSort Project

1. Project Title

Parallel QuickSort Using MPI

2. Project Scope

Objective: The aim of this project is to implement a parallel version of the QuickSort algorithm using MPI (Message Passing Interface) and evaluate its performance against a sequential version. The project seeks to demonstrate the efficiency gains achieved through parallel processing.

Scope: This project encompasses:

- The design and implementation of a sequential QuickSort algorithm.
- The parallelization of the QuickSort algorithm using MPI.
- Performance evaluation of both implementations through empirical testing.

3. Project Design

System Architecture: The project consists of two main implementations:

- **Sequential QuickSort**: A straightforward implementation that sorts an array in a single-threaded manner.
- **Parallel QuickSort**: An implementation that uses MPI to distribute the sorting task across multiple processes.

Key Components:

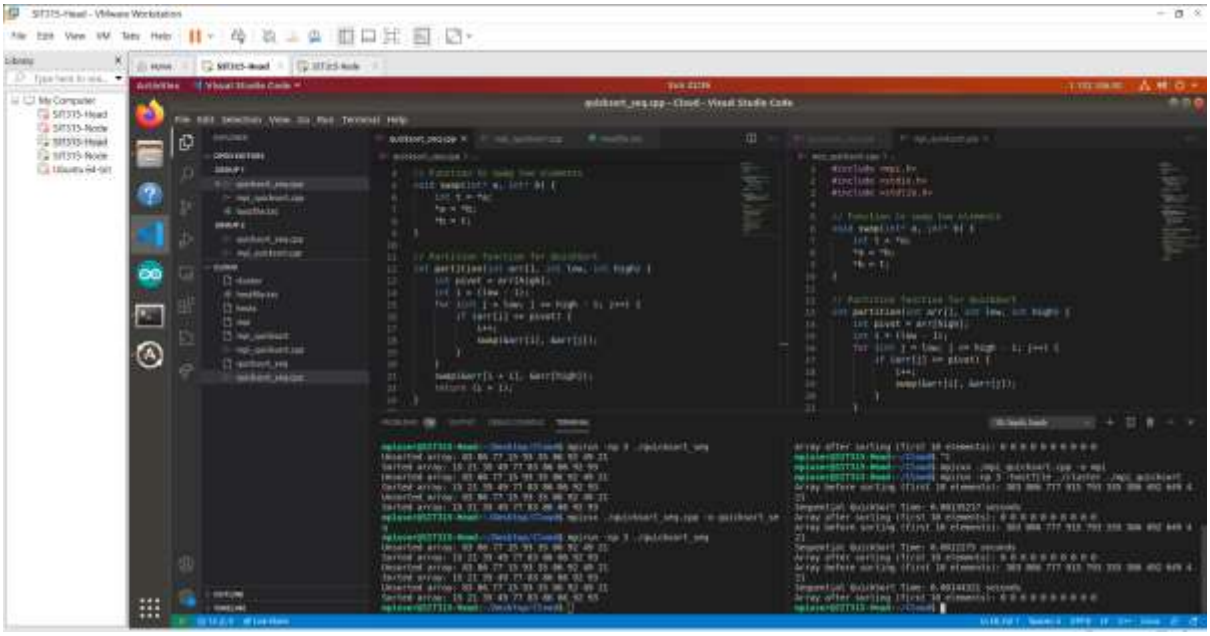
- **Master Process**: Responsible for initializing the array, partitioning it, distributing the data to worker processes, and gathering sorted results.
- **Worker Processes**: Each worker sorts its designated portion of the array independently.

Data Flow:

1. **Initialization**: The master process creates an array with random integers.
2. **Distribution**: The master process partitions the array and sends these partitions to worker processes.
3. **Parallel Sorting**: Each worker process sorts its assigned partition using QuickSort.
4. **Collection and Merging**: The master process gathers the sorted subarrays and combines them into a final sorted array.

4. Implementation

The implementation is conducted in C++ with the use of MPI for parallel execution.



5. Evaluation

Performance Testing: The project was tested with various array sizes to measure the performance of both implementations. Execution times were recorded, demonstrating that the parallel implementation significantly reduces sorting time for larger datasets compared to the sequential version.

Challenges and Limitations:

- The main challenge was ensuring proper data distribution among processes to maximize efficiency.
- The performance gain diminishes with smaller datasets due to overhead from process communication.