SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE

A

MINI PROJECT REPORT

ON

"Performance enhancement of parallel Quicksort Algorithm using MPI"

SUBMITTED TO THE SAVITRIBAI PHULE PUNE UNIVERSITY, PUNE IN THE FULFILLMENT OF THE REQUIREMENT

OF

Laboratory Practice VI

Fourth Year Computer Engineering

Academic Year 2024-25

BY

Name of Students:	**Exam No.:**
Abhang Pratik Sunil	B1905704201
Borase Darshana Ravasaheb	B1905704228
Darekar Omkar Balasaheb	B1905704232
Gunde Chaitanya Diliprao	B1905704256

Under the Guidance of

Mr. K. A. Shinde

Sinhgad Institutes

DEPARTMENT OF COMPUTER ENGINEERING

STES'S SINHGAD INSTITUTE OF TECHNOLOGY AND SCIENCE

NARHE, PUNE – 411041

Department of Computer Engineering
Sinhgad Institute of Technology and Science, Narhe, Pune

CERTIFICATE

This is to certify that,

Name of Students:	**Exam No.:**
Abhang Pratik Sunil	B1905704201
Borase Darshana Ravasaheb	B1905704228
Darekar Omkar Balasaheb	B1905704232
Gunde Chaitanya Diliprao	B1905704256

studying in BE Computer Engineering Course SEM-VIII has successfully completed their LP-VI Lab Mini-Project work titled **“Performance enhancement of parallel Quicksort Algorithm using MPI”** at Sinhgad Institute of Technology and Science, Narhe in the fulfillment of the bachelor’s degree in engineering of Savitribai Phule Pune University, during the academic year 2024-2025.

Mr. K. A. Shinde	Dr. G. S. Navale	Dr. S. D. Markande
Guide	**Head of Department**	**Principal**

SINHGAD INSTITUTE OF TECHNOLOGY AND SCIENCE, NARHE, PUNE– 411041

Place : Pune

Date :

ACKNOWLEDGEMENT

We take this opportunity to acknowledge each and every one who contributed towards our work. We express our sincere gratitude towards guide **Mr. K. A. Shinde**, Assistant Professor at Sinhgad Institute of Technology and Science, Narhe, Pune for his valuable inputs, guidance and support throughout the course.

We wish to express our thanks to **Dr. G. S. Navale,** Head of Computer Engineering Department, Sinhgad Institute of Technology and Science, Narhe for giving us all the help and important suggestions all over the Work.

We thank all the teaching staff members for their indispensable support and priceless suggestions. We also thank our friends and family for helping in collecting data, without their help **Performance enhancement of parallel Quicksort Algorithm using MPI** report have been completed. At the end our special thanks to **Dr. S. D. Markande,** Principal Sinhgad Institute of Technology and Science, Narhe for providing ambience in the college, which motivates us to work.

Name of Students:	**Sign:**
Abhang Pratik Sunil	
Borase Darshana Ravasaheb	
Darekar Omkar Balasaheb	
Gunde Chaitanya Diliprao	

1. Problem Statement

Evaluate performance enhancement of parallel Quicksort Algorithm using MPI.

2. Objective

To implement and analyze the performance enhancement of the Quicksort algorithm through parallelization using the Message Passing Interface (MPI), by comparing execution time, speedup, and efficiency against its sequential counterpart across varying input sizes and processor counts.

3. Pre-Requisites

- Knowledge of the Quicksort algorithm
- Understanding of parallel computing concepts
- Familiarity with MPI (Message Passing Interface)
- Programming skills in C/C++ or Python with MPI bindings

4. Theory

Quicksort is a widely used divide-and-conquer sorting algorithm known for its efficiency in sorting large datasets. It works by selecting a pivot element, partitioning the array into two sub-arrays around the pivot, and recursively sorting the sub-arrays. While efficient in its sequential form, Quicksort can be further optimized through parallelization to take advantage of modern multi-core and distributed systems.

The Message Passing Interface (MPI) enables parallel programming across multiple processes by providing a communication protocol for data exchange. By parallelizing Quicksort with MPI, different parts of the array can be sorted concurrently across multiple processors, thereby reducing execution time. This parallel implementation, however, introduces challenges such as data distribution, load balancing, and communication overhead, all of which must be carefully managed to achieve significant performance gains.

5. Process

The evaluation of performance enhancement in the parallel Quicksort algorithm using MPI involves several key steps. First, the input array is generated or read and then distributed among multiple processes using MPI communication primitives. One process acts as the root and is responsible for initiating the Quicksort by selecting a pivot and partitioning the data. Each process then recursively applies the Quicksort algorithm to its assigned portion of the array in parallel.

Communication between processes ensures that data is correctly shared and merged during recursive calls. Once all sub-arrays are sorted, the sorted segments are gathered and combined into a final sorted array. Throughout the process, performance metrics such as execution time, speedup, and efficiency are recorded for various input sizes and processor counts. This allows for a detailed comparison between the parallel implementation and its sequential counterpart, highlighting the benefits and potential limitations of parallelization with MPI.

6. Code

Input:

```
from mpi4py import MPI
import numpy as np
import time

# Initialize MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

def quicksort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[len(arr) // 2]
        left = [x for x in arr if x < pivot]
        middle = [x for x in arr if x == pivot]
        right = [x for x in arr if x > pivot]
        return quicksort(left) + middle + quicksort(right)

# Main execution
def main():
    # Problem size
    total_elements = 16

    # Root process generates random array
    if rank == 0:
        data = np.random.randint(0, 100, size=total_elements)
        print("Input Array:", data)
        chunks = np.array_split(data, size)
    else:
        chunks = None

    # Scatter chunks to all processes
    local_data = comm.scatter(chunks, root=0)

    # Start timing after scatter
    start = MPI.Wtime()

    # Local quicksort
    local_sorted = quicksort(local_data)

    # Gather sorted chunks
    gathered_data = comm.gather(local_sorted, root=0)

    # Root process merges final sorted array
    if rank == 0:
        final = []
        for chunk in gathered_data:
            final.extend(chunk)
        final = quicksort(final)
```

```
        end = MPI.Wtime()
        print("Sorted Array:", final)
        print("Execution Time:", round(end - start, 6), "seconds")

if __name__ == "__main__":
    main()
```

Output:
Input Array: [18 92 23 7 86 91 2 17 61 57 67 15 6 47 29 40]
Sorted Array: [2, 6, 7, 15, 17, 18, 23, 29, 40, 47, 57, 61, 67, 86, 91, 92]
Execution Time: 0.000XXX seconds

7. Conclusion

The parallel implementation of the Quicksort algorithm using MPI significantly improves performance by distributing the workload across multiple processes. This approach demonstrates enhanced execution speed compared to the sequential version, especially for large datasets.