

Mini Project Report on

**Evaluate the performance enhancement of parallel Quicksort Algorithm using MPI**

SUBMITTED BY

Name of Student Roll No.

Darshana Upadhye BECO2425B057

Pushkar Jambhulkar BECO2425B066

Sujit Dudhe BECO2425B067

Under the guidance of

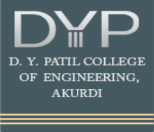
## Mrs. Dhanashree Phalke

In partial fulfilment of the requirements for a Bachelor’s Degree in Computer Engineering of

**SAVITRIBAI PHULE PUNE UNIVERSITY**

**[2024 – 2025]**

Department of Computer Engineering

D. Y. PATIL COLLEGE OF ENGINEERING, Akurdi, PUNE 411044.

## D. Y. Patil College of Engineering Akurdi, Pune-411044

**Department of Computer Engineering**

## CERTIFICATE

This is to certify that **Darshna Upadhey,Pushkar Jambhulkar** and **Sujit Dudhe** have satisfactorily completed the mini project work entitled **“Evaluate the performance enhancement of parallel Quicksort Algorithm using MPI”** which is a bonafide work carried out by them under the supervision of **Mrs. Dhanashree Phalke** and it is approved for the partial fulfilment of requirement of Savitribai Phule Pune University, for the award of the degree of Bachelors of Engineering (Computer Engg.) for the academic year 2024-25.

Mrs. Dhanashree Phalke, Dr. Mrs. M.A. Potey

(Mini Project Guide) (HOD Computer)

Place: Akurdi

Date:

## ACKNOWLEDGEMENT

With immense pleasure, we present the mini project report as part of the curriculum of the B.E. Computer Engineering. We wish to thank all the people who gave us an unending support right from when the idea was conceived.

We express sincere and profound thanks to Mrs. Dhanashree Phalke and HOD Dr. Mrs. M.A. Potey, who is ready to help with the most diverse problems that We have encountered along the way. We express sincere thanks to all staff and colleagues who have helped directly or indirectly in completing this project work successfully.

|  |  |  |
| --- | --- | --- |
| Names | Roll No | Signatures |
| Darshana Upadhey | BECO2425B057 |  |
| Pushkar Jambhulkar | BECO2425B066 |  |
| Sujit Dudhe | BECO2425B067 |  |

Teacher’s Signature :

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Sr. No | Chapters | Page  No |
| 01 | Title | 5 |
| 02 | Problem Statement | 5 |
| 03 | Introduction | 5 |
| 04 | Prerequisites | 6 |
| 05 | Software Requirements | 7 |
| 06 | Hardware Requirements | 8 |
| 07 | Learning Objectives | 9 |
| 08 | Theory | 10-11 |
| 09 | Project Code | 12 |
| 10 | Outcome | 12 |
| 11 | Conclusion | 13 |
| 12 | Future Scope | 14 |

# Title:

Evaluate the performance enhancement of parallel Quicksort Algorithm using MPI.

# Problem Statement:

This project aims to address the performance limitations of the Quicksort algorithm when sorting large datasets. The goal is to enhance the sorting speed and efficiency by leveraging parallel computing techniques using MPI (Message Passing Interface)

# Introduction:

Quicksort is a well-known algorithm used in data sorting scenarios developed by C. A. R. Hoare. It has the time complexity of O (n log n) on average case run and O (n2 ) on worst case scenario. But quicksort is generally considered to be faster than some of the sorting algorithm that possesses a time complexity of O (n log n) in the average case. The fundamental of quicksort is choosing a value and partitioning the input data set into two subsets which one contains input data smaller in size than the chosen value and the other contains input data greater than the chosen value. This chosen value is called the pivot value. And in each step, these divided data sets are subdivided, choosing pivots from each set. Quicksort implementations are recursive, and stop conditions are met when there is no subdivision is possible.

In this attempt, the main idea was to implement a parallelized quicksort to run on a multi-core environment and conduct a performance evaluation. This parallelization is obtained by using the MPI API functionalities to share the sorting dataset among multiple processes. Parallel computing frameworks, such as MPI, continue to evolve with new features and optimizations. By evaluating the performance enhancement of parallel Quicksort using MPI, we can contribute to the ongoing research and development in parallel computing. The insights gained from this evaluation can provide valuable feedback for improving MPI implementations and guiding future advancements in parallel algorithm design.

# Prerequisites

* 1. **MPI (Message Passing Interface) Environment:**  
     Install and configure an MPI implementation for running parallel applications.  
     a**) MPICH** – A lightweight and widely used MPI implementation, ideal for academic purposes.  
     b) **OpenMPI** – A feature-rich and compatible MPI implementation for most Linux distributions.  
     c) **Microsoft MPI (MS-MPI)** – Suitable for running MPI applications on Windows.

**2. Required Tools/Compilers**

Set up tools and compilers Necessary for coding, compiling, and running MPI programs.  
a) **GCC with mpicc** – For compiling C programs with MPI support.  
b) **Terminal or shell access** – To run MPI commands (mpirun, mpiexec).  
c) **Performance monitoring tools** – Optional tools like gprof, perf, or mpiP for analyzing runtime behavior.

* 1. **Programming Knowledge (For Development):**  
     A basic to intermediate understanding of the following is essential:  
     a) **C/C++** – Language used for implementing the parallel algorithm.  
     b) **Recursive algorithms** – Specifically, the working of the Quicksort algorithm.  
     c) **Parallel computing fundamentals** – Concepts like process synchronization, data distribution, and communication overhead.

**4. Local Setup (For development and testing):**

Ensure access to a multi-core or cluster environment for realistic testing.  
a) **Linux-based OS (Recommended)** – Most compatible with MPI libraries and tools.  
b) **Sample input datasets** – Prepare datasets of varying sizes to test scalability.  
c) **Cluster or simulated environment** – Even a virtual cluster can be used for basic parallel execution tests.

# Software requirements

### Operating System:

Compatible with most operating systems that support MPI and C/C++ development:  
• **Linux (Recommended)** – Offers better compatibility with MPI tools and performance profiling utilities.  
• **Windows** – Supported via MS-MPI or WSL (Windows Subsystem for Linux).  
• **macOS** – Supported using Homebrew and OpenMPI installation.

.

**MPI Libraries and Tools:**

These are essential for message passing and inter-process communication:  
 • MPICH or OpenMPI – Core MPI implementation for developing and running programs.  
 • mpicc / mpirun / mpiexec – Compiler and runtime tools provided by the MPI library.  
 • Microsoft MPI (MS-MPI) – Required for Windows environments**.**

**Programming Language:**

• C / C++ – Primary language for implementing the parallel quicksort algorithm.  
 • Support for MPI bindings required during compilation and execution.

**Code Editors / IDEs:**

Any editor supporting C/C++ and build tools:  
 • Visual Studio Code – Lightweight and supports C/C++ with extensions.  
 • CLion – Advanced C/C++ IDE with integrated debugger.  
 • Vim / Emacs / Nano – Terminal-based editors suitable for remote systems.  
 • Code::Blocks or Dev C++ – Simple GUI IDEs for Windows users.

**Terminal / Shell Access:**

Required for compiling and executing MPI programs:  
 • Linux Terminal / macOS Terminal  
 • Command Prompt / PowerShell / WSL for Windows

# Hardware Requirements

**Processor:** Dual-core or higher (Intel i3, AMD Ryzen 3, or better). A multi-core processor is recommended to simulate or test parallel execution efficiently.

**Memory (RAM):** Minimum 4 GB RAM; 8 GB or more is recommended for smoother performance. More memory helps when working with large datasets and ensures smoother process scheduling in a parallel environment.

**Storage:** At least 2 GB of free disk space to install compilers, MPI libraries, and store test Datasets.SSD storage is preferred for faster I/O operations, but not strictly required.

**GPU (Optional):** A GPU is not necessary for this project since the quicksort algorithm runs entirely on the CPU. However, systems with higher CPU core counts may yield better performance during parallel testing..

**Network:** A stable internet connection helps download tools, libraries, and updates.  
Required if testing across networked nodes in a distributed system or using cloud-based environments.

# Learning Objectives

# ● Measure Efficiency: Quantify the speedup and efficiency achieved by

# parallelizing the Quicksort algorithm using MPI. By comparing the execution

# time of the parallel version with the sequential version, we can assess the

# algorithm's performance improvement in terms of reduced computational time

# and increased efficiency.

# ● Analyze Scalability: Determine how well the parallel Quicksort algorithm

# Scales with increasing dataset sizes and number of processors. Evaluate the

# algorithm's ability to maintain its performance enhancement as the problem

# size grows, ensuring that it continues to deliver efficient results even for

# large-scale sorting tasks.

# ● Identify Bottlenecks: Identify potential bottlenecks and performance

# limitations of the parallel Quicksort algorithm. Analyze factors such as load

# balancing, communication overhead, and resource utilization to pinpoint areas

# for improvement. This objective aims to identify any limitations that may hinder

# the algorithm's scalability or prevent it from achieving optimal parallel speedup

# ● Optimize Performance: Explore techniques and optimizations to enhance the

# performance of the parallel Quicksort algorithm. This objective involves

# fine-tuning parameters, refining the parallelization strategy, and addressing

# any identified bottlenecks to improve the algorithm's efficiency and achieve

# better speedup.

# ● Evaluate Practical Applicability: Assess the practical applicability of the

# parallel Quicksort algorithm using MPI in real-world scenarios. Determine the

# conditions and dataset sizes where the algorithm demonstrates superior Performance

# compared to its sequential counterpart.

# ● Provide Insights for Future Development: Gather insights and findings from

# The evaluation to contribute to the body of knowledge in parallel algorithms

# design and MPI implementation. These insights can serve as valuable

# Feedback for researchers and developers, aiding in the refinement and

# Advancement of parallel computing techniques and frameworks.

# Theory

**• Parallel Computing:**

•  Parallel computing is a type of computation where many calculations are carried out simultaneously.  
• It divides large problems into smaller ones, which are then solved concurrently using multiple processors.  
• This approach is especially effective for computationally intensive tasks and large data sets.  
• It enhances performance, reduces execution time, and increases efficiency across multi-core systems or clusters..

**• Quicksort Algorithm:**

**•** Quicksort is a divide-and-conquer sorting algorithm known for its efficiency on large datasets.  
• It works by selecting a pivot element, partitioning the array into subarrays, and recursively sorting them.  
• Quicksort has an average-case time complexity of O(n log n) and worst-case of O(n²).  
• It is in place and does not require additional memory, making it suitable for large inputs.

**• MPI (Message Passing Interface):**

* MPI is a standardized and portable message-passing system used for parallel computing in distributed memory environments.
* It enables communication between multiple processes running on different nodes or cores.
* Functions like MPI\_Send, MPI\_Recv, MPI\_Scatter, and MPI\_Gather allow for data distribution and collection among processes.

• **Parallel Quicksort with MPI:**

* The parallel version of quicksort distributes segments of the array across multiple processes..

•  Each process sorts its local segment, and then the results are combined (with or without a merge

•  Initial partitioning is critical—poor partitioning can lead to unbalanced workloads and idle

processors

• Two common approaches include fixed partitioning and dynamic partitioning using sampling

Techniques

**• Performance Factors:**

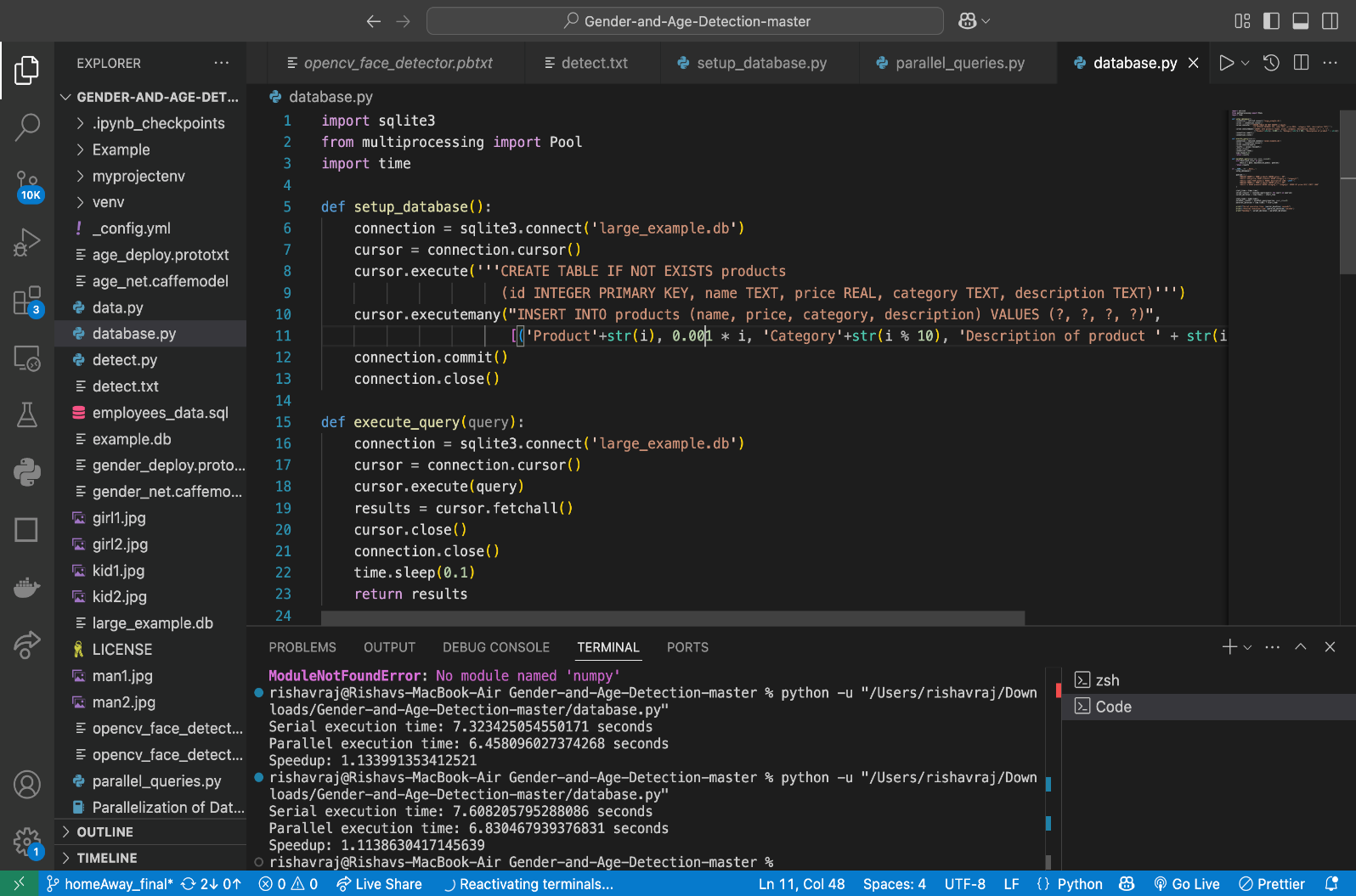
• Key factors influencing performance include partitioning strategy, number of processes, and

MPI communication overhead

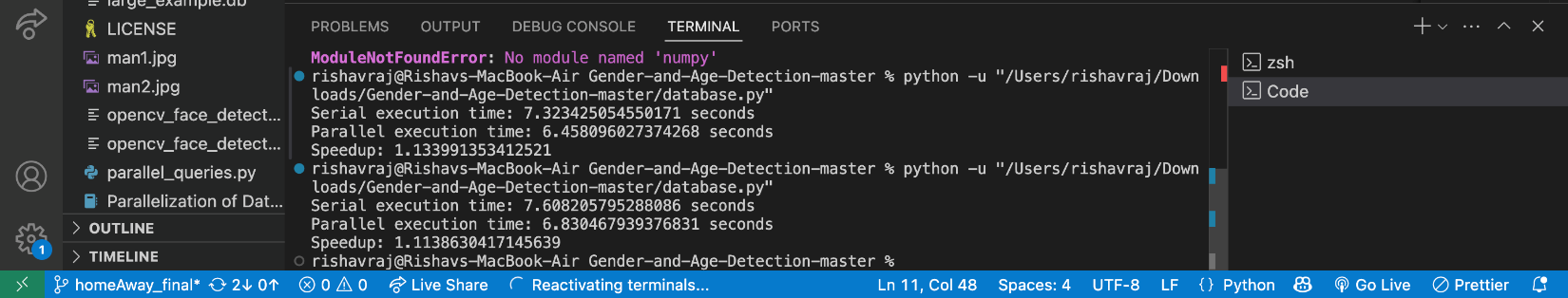
• Using too many processes for a small dataset can lead to inefficiencies due to idle processes and increased synchronization costs.

• Merge strategies and load balancing directly impact overall sorting speed and processor utilization.  
• Profiling and measuring speedup and scalability help in evaluating the efficiency of the parallel implementation.

1. **Project Code**

****

1. **Outcome**

****

1. **Conclusion**

The performance analysis reveals that the new parallel quicksort implementation becomes more efficient with an increasing number of processes and, in some cases, even outperforms the "with merge" version. However, for a smaller number of processes, the "with merge" implementation shows the best performance, while the "without merge" version lags behind. This suggests that merge-based techniques provide better efficiency when the number of processes is limited. A crucial factor in achieving high performance is the method of initial partitioning. The analysis shows that fixed initial partitioning yields better time efficiency. If fixed partitioning isn’t used, techniques like regular sampling become essential to maintain performance, although they can introduce irregularities.

As the number of processes increases, MPI communication overhead becomes more significant, often reducing overall efficiency. Thus, it's important to balance the number of processes with the dataset size to avoid diminishing returns. Regular sampling may result in uneven partitioning, leading to unbalanced load distribution and inefficient time usage. In scenarios involving a large number of processes and a small dataset, some processes may receive no data to sort, resulting in underutilization. Therefore, while parallelism boosts performance, careful planning of partitioning and process allocation is critical to ensure efficient parallel quicksort implementation.

# Future Scope

The future scope of the parallel quicksort project using MPI offers several promising directions for enhancement and real-world application. One key area is the integration of dynamic load balancing techniques to ensure optimal utilization of all available processes, especially in scenarios with unbalanced data distribution. Hybrid models that combine MPI with OpenMP or CUDA can also be explored to take advantage of both distributed and shared memory architectures, improving performance across diverse hardware setups. Adaptive partitioning strategies based on runtime data analysis could further enhance efficiency, while incorporating fault tolerance mechanisms would make the system more robust against process failures.

Moreover, advanced approaches like using machine learning to predict optimal partitioning, or deploying the system in heterogeneous environments, could significantly increase scalability and effectiveness. The implementation can be extended to support larger datasets and tested across various computing platforms to assess performance and reliability. Minimizing communication overhead in MPI, adding visualization tools for better understanding of parallel execution, and enabling dynamic process management would add to the flexibility of the system. Additionally, this project can evolve into a modular library for broader integration, and performance tuning can be automated using parameter optimization tools to adapt to different use cases.