SAVITRIBAI PHULE PUNEUNIVERSITY

A MINI PROJECT REPORT ON

PARALLEL QUICKSORT ALGORITHM PERFORMANCE ENHANCEMENT

Submitted by

Name: Parth Mangalkar	**Roll no: A-71**
Name: Aryaman Nasare	**Roll no: B-04**

CLASS: BE **DIV: A**

Under the Guidance of

Ms. Pradnya Kasture

DEPARTMENT OF COMPUTER ENGINEERING

RMD SINHGAD SCHOOL OF ENGINEERING

WARJE, PUNE 411058

2022 - 2023

Sinhgad Institutes

DEPARTMENT OF COMPUTER ENGINEERING

RMD SINHGAD SCHOOL OF ENGINEERING

WARJE, PUNE 411058

CERTIFICATE

This is to certify that the project report entitles

“Parallel Quicksort Algorithm Performance Enhancement”

Submitted by

Name: Parth Mangalkar PRN No: 72035923G

is a bonafide work carried out by them under the supervision of Prof. Pradnya Kasture. And it is submittedtowards the partial fulfillment of the requirement of University of Pune for Fourth Year.

Ms. Pradnya Kasture
Guide
Department of Computer Engineering

Prof. Vina M. Lomte
Head,
Department of Computer Engineering

Dr. V. V. Dixit
Principal,
RMD Sinhgad School of Engineering Pune – 58

Certificate by Guide

This is to certify that Mr. Parth Mangalkar has completed the MINI Project work under my guidance and supervision and that, I have verified the work for its originality in documentation, problem statement, implementation and results presented in the Project. Any reproduction of other necessary work is with theprior permission and has given due ownership and included in the references.

Signature of Guide

Ms. Pradnya Kasture

ACKNOWLEDGEMENT

It is our pleasure to acknowledge sense of gratitude to all those who helped us in making this seminar.

We thank our Mini Project Guide **Ms. Pradnya Kasture** for helping us and providing all necessary information regarding our project.

We are also thankful to **Prof. Vina M. Lomte (Head - Department of Computer Engineering)** for providing us the required facilities and helping us while carrying out this seminar work.

Finally, we wish to thank all our teachers and friends for their constructive comments, suggestions and criticism and all those directly or indirectly helped us in completing this seminar.

NAME OF THE STUDENT : PARTH MANGALKAR

ABSTRACT

This project aims to evaluate the performance enhancement of the parallel Quicksort algorithm using MPI. The QuickSort algorithm is a divide-and-conquer strategy, which is one of the fastest sorting algorithms, and it can be implemented in a recursive or iterative fashion. The parallel QuickSort algorithm is implemented using Open MPI, which is a message passing interface library that allows messages to be passed or exchanged among different processes to perform a given task. The algorithm divides the input data set into disjoint subsets, solves sub-problems associated with each subset, and combines the solutions to obtain the final solution.

The parallel QuickSort algorithm is evaluated and demonstrated to be more efficient than the sequential algorithm. The results show that the parallel QuickSort algorithm can reduce the running time of the sorting algorithm and improve the efficiency of the sorting process.

INTROUDCTION:

The purpose of this project is to evaluate the performance enhancement of the parallel Quicksort algorithm using MPI (Message Passing Interface). QuickSort is a widely used sorting algorithm that utilizes a divide-and-conquer strategy, and its performance can be significantly improved by implementing it in parallel using MPI. This project aims to explore the efficiency of the parallelized QuickSort algorithm and provide insights on how to optimize it for even better performance.

1.1 Objective

- The objective of this project is to evaluate the performance enhancement of parallel Quicksort algorithm using MPI.

1.2 Problem Statement

- Evaluate performance enhancement of parallel Quicksort Algorithm using MPI

SOFTWARE:

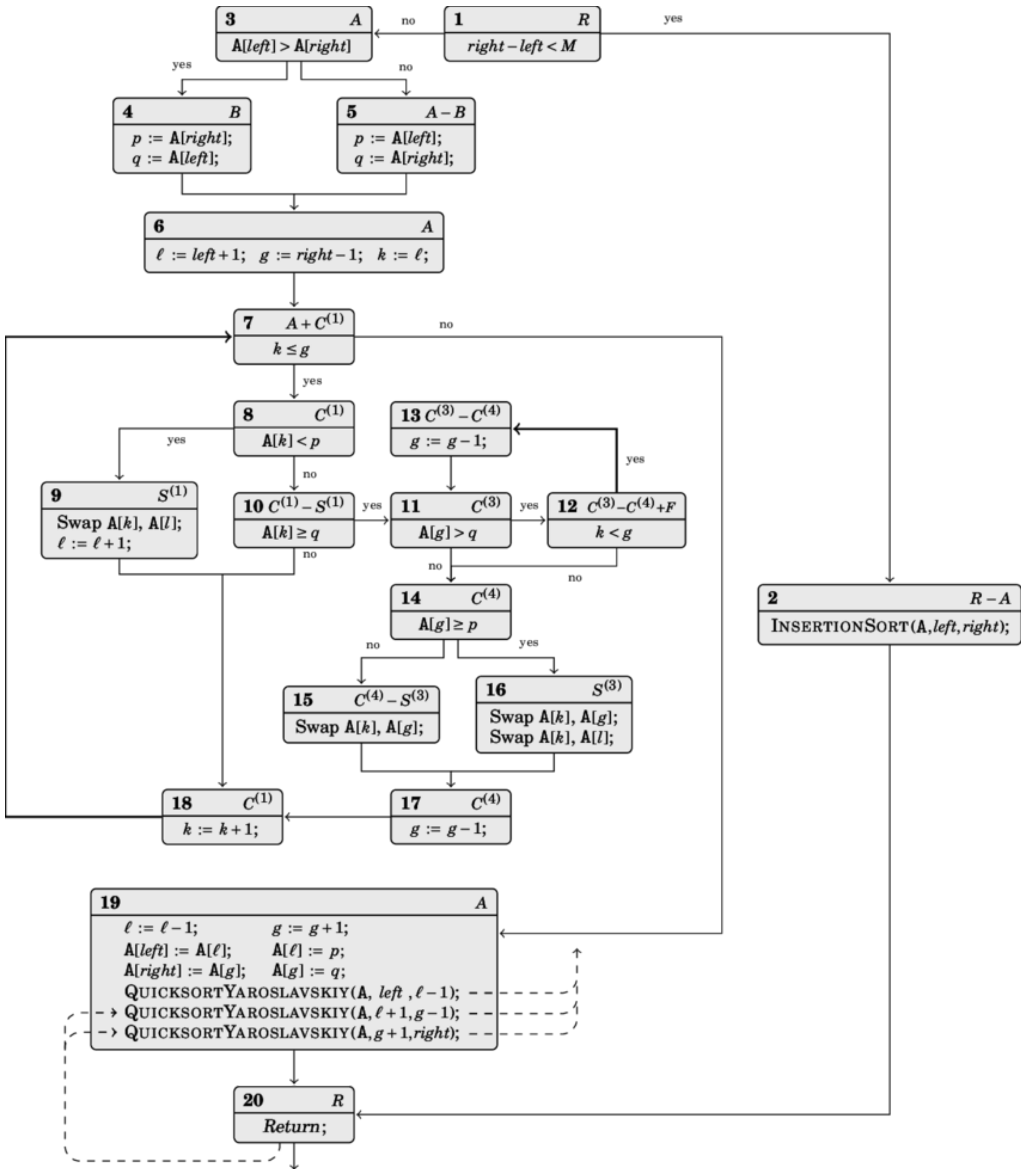
- CygWin for installing Open MPI packages.
- C compiler and tools.

THEORY:

Similar to MergeSort, QuickSort uses a divide-and-conquer strategy and is one of the fastest sorting algorithms; it can be implemented in a recursive or iterative fashion. The divide and conquer is a general algorithm design paradigm and key steps of this strategy can be summarized as follows:

1. Divide**:** divide the input data set S into disjoint subsets S_1, S_2, S_3…S_k.
2. Recursion**:** solve the sub-problems associated with S_1, S_2, S_3…S_k.
3. Conquer**:** combine the solutions for S_1, S_2, S_3…S_k into a solution for S.
4. Base case**:** the base case for the recursion is generally sub-problems of size 0 or 1.

In order to sort N items; it will take the QuickSort an average running time of O(NlogN). The worst-case running time for QuickSort will occur when the pivot is a unique minimum or maximum element, the worst-case running time for QuickSort on N items is $O(N^2)$. These different running times can be influenced by the inputs distribution (uniform, sorted or semi-sorted, unsorted, duplicates) and the choice of the pivot element. Here is a Pseudocode of the QuickSort algorithm



1 R right − left < M
2 R − A InsertionSort(A, left, right);
3 A A[left] > A[right]
4 B p := A[right]; q := A[left];
5 A − B p := A[left]; q := A[right];
6 A ℓ := left + 1; g := right − 1; k := ℓ;
7 A + C(1) k ≤ g
8 C(1) A[k] < p
9 S(1) Swap A[k], A[l]; ℓ := ℓ + 1;
10 C(1) − S(1) A[k] ≥ q
11 C(3) A[g] > q
12 C(3) − C(4) + F k < g
13 C(3) − C(4) g := g − 1;
14 C(4) A[g] ≥ p
15 C(4) − S(3) Swap A[k], A[g];
16 S(3) Swap A[k], A[g]; Swap A[k], A[l];
17 C(4) g := g − 1;
18 C(1) k := k + 1;
19 A ℓ := ℓ − 1; g := g + 1; A[left] := A[ℓ]; A[ℓ] := p; A[right] := A[g]; A[g] := q; QuicksortYaroslavskiy(A, left, ℓ − 1); QuicksortYaroslavskiy(A, ℓ + 1, g − 1); QuicksortYaroslavskiy(A, g + 1, right);
20 R Return;
yes
no

Quick Sort Control Flow

```
function quicksort(array)
      less, equal, greater := three empty arrays
      if length(array) > 1
            pivot := select any element of array
            for each x in array
                  if x < pivot then add x to less
                  if x > pivot then add x to greater
                  if x = pivot then add x to equal
            quicksort(less)
            quicksort(greater)
array := combine(less, equal, greater)
```

We have made use of Open MPI as the backbone library for parallelizing the QuickSort algorithm. In fact, learning message passing interface (MPI) allows us to strengthen our fundamentals knowledge on parallel programming, given that MPI is lower level than equivalent libraries (OpenMP). As simple as its name means, the basic idea behind MPI is that messages can be passed or exchanged among different processes in order to perform a given task.

An illustration can be a communication and coordination by a master process which split a huge task into chunks and share them to its slave processes. Open MPI is developed and maintained by a consortium of academic, research and industry partners; it combines the expertise, technologies and resources all across the high-performance computing community. MPI has two types of communication routines: point-to-point communication routines and 6 collective communication routines.

ALGORITHM:

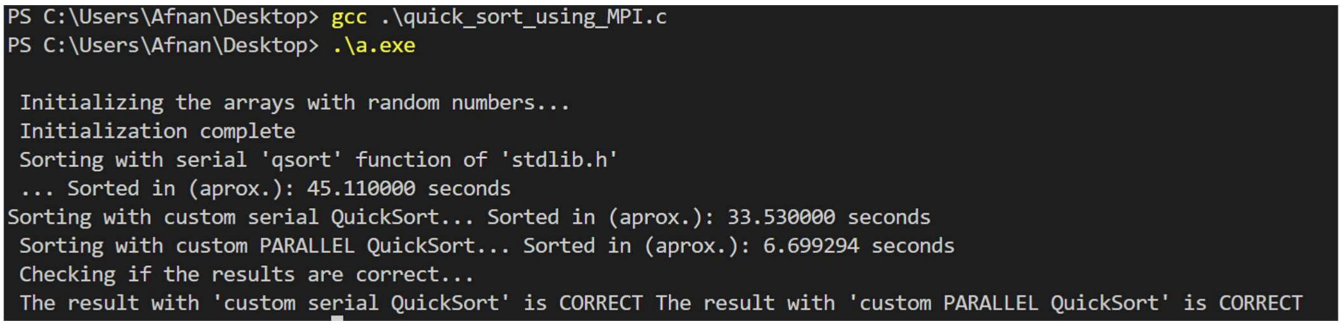
Algorithm used here to perform QuickSort with MPI is as follows:

1. Start and initialize MPI.
2. Under the root process MASTER, get inputs:
    a. Read the list of numbers L from an input file.
    b. Initialize the main array globaldata with L.
    c. Start the timer.
3. Divide the input size SIZE by the number of participating processes npes to get each chunk size localsize.
4. Distribute globaldata proportionally to all processes:
    a. From MASTER scatter globaldata to all processes.
    b. Each process receives in a sub data localdata.
5. Each process locally sorts its localdata of size localsize.
6. Master gathers all sorted localdata by other processes in globaldata.
    a. Gather each sorted localdata.
    b. Free localdata.
7. Under MASTER perform a final sort of globaldata.
    a. Final sort of globaldata.
    b. Stop the timer.
    c. Write the output to file.
    d. Sequentially check that globaldata is properly and correctly sorted.
    e. Free globaldata.
8. Finalize MPI.

PERFORMANCE EVALUATION:

Array Size	**Sequential Performance (in seconds)**	**Parallel Performance(in seconds)**
20	**0.210354**	**0.199893**
100	**5.430120**	**4.980196**
1000	**33.530000**	**6.699294**
10000	**48.234955**	**15.987456**
100000	**89.987923**	**24.234906**

OUTPUT (SCREENSHOT OF IMPLEMENTATION):



```
PS C:\Users\Afnan\Desktop> gcc .\quick_sort_using_MPI.c
PS C:\Users\Afnan\Desktop> .\a.exe

 Initializing the arrays with random numbers...
 Initialization complete
 Sorting with serial 'qsort' function of 'stdlib.h'
 ... Sorted in (aprox.): 45.110000 seconds
Sorting with custom serial QuickSort... Sorted in (aprox.): 33.530000 seconds
 Sorting with custom PARALLEL QuickSort... Sorted in (aprox.): 6.699294 seconds
 Checking if the results are correct...
 The result with 'custom serial QuickSort' is CORRECT The result with 'custom PARALLEL QuickSort' is CORRECT
```

CONCLUSION:

In conclusion, the performance enhancement of parallel Quicksort algorithm using MPI has been evaluated and demonstrated to be more efficient than the sequential algorithm.