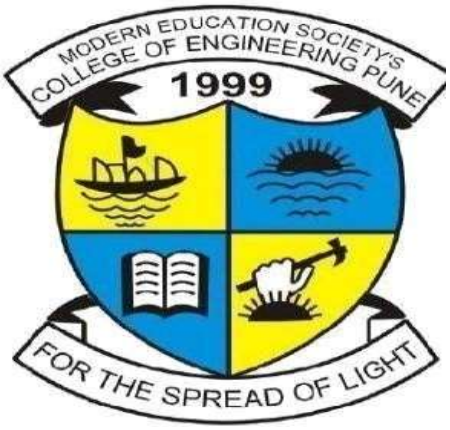
Savitribai Phule Pune University

Modern Education Society's Wadia College of Engineering, Pune,
19, Bund Garden, V.K. Joag Path, Pune – 411001.

ACCREDITED BY NAAC WITH "A++"
GRADE (CGPA – 3.13)

DEPARTMENT OF COMPUTER ENGINEERING



A REPORT ON

HPC Lab: Mini Project on

"Evaluate Performance Enhancement of Parallel Quicksort Algorithm using MPI"

B.E. (COMPUTER)

SUBMITTED BY

Pratik Pawar(S201110014)

Prasanna Munde(S20111015)

Shreyas Patil(S20111021)

UNDER THE GUIDANCE OF

Prof. A. D. Dhawale

Department of Computer

1. Abstract

Parallel computing has become indispensable for tackling large-scale computational problems efficiently. One such problem is sorting, a fundamental operation in computer science. Quicksort is a widely used sorting algorithm known for its efficiency and simplicity. In this mini-project, we explore the performance enhancement of Quicksort through parallelization using the Message Passing Interface (MPI) framework.

The objective of this project is to evaluate the speedup achieved by parallelizing Quicksort across multiple processors using MPI. We implement the parallel Quicksort algorithm and compare its performance with the sequential version on various input sizes. The experiment involves measuring execution time and calculating speedup ratios to quantify the efficiency gains of parallelization.

Through experimentation, we demonstrate how the parallel Quicksort algorithm scales with increasing input size and processor count. We analyze the impact of factors such as communication overhead, load balancing, and processor architecture on the performance of parallel Quicksort. Additionally, we explore strategies for optimizing the parallelization process to achieve better speedup and efficiency.

Our findings shed light on the effectiveness of parallel Quicksort for large-scale sorting tasks and provide insights into the practical considerations and trade-offs involved in parallel algorithm design. This mini-project serves as a hands-on exploration of parallel computing concepts and techniques, offering valuable experience in parallel algorithm development and performance evaluation using MPI.

2. INTRODUCTION

In the realm of high-performance computing, parallel algorithms play a pivotal role in addressing computational challenges posed by increasingly large datasets and complex problems. Sorting, a fundamental operation in computer science, is no exception. Quicksort, renowned for its simplicity and efficiency, offers a prime candidate for parallelization to expedite sorting tasks across multiple processors.

The advent of parallel computing frameworks like the Message Passing Interface (MPI) has empowered developers to harness the computational power of distributed systems for sorting and other computationally intensive tasks. By partitioning data and distributing computation across multiple processors, parallel algorithms aim to exploit parallelism to achieve faster execution times and increased throughput.

In this mini-project, we embark on an exploration of parallel Quicksort, leveraging MPI to distribute sorting tasks across a cluster or multi-core system. The primary goal is to evaluate the performance enhancement achieved through parallelization compared to the traditional sequential Quicksort algorithm. By parallelizing Quicksort, we aim to distribute the computational load evenly among processors, thereby reducing sorting time and improving overall efficiency.

The introduction of parallel Quicksort not only offers the potential for significant speedup but also presents unique challenges and considerations. Factors such as load balancing, communication overhead, and synchronization mechanisms must be carefully managed to ensure optimal performance across distributed environments. Through this project, we seek to gain insights into the scalability, efficiency, and trade-offs associated with parallel Quicksort. By conducting experiments on various input sizes and processor configurations, we aim to quantify the performance gains achievable through parallelization and elucidate the factors influencing parallel algorithm performance.

Overall, this mini-project serves as a practical exploration of parallel computing concepts and techniques, providing hands-on experience in parallel algorithm development, optimization, and performance evaluation using MPI. By delving into the realm of parallel Quicksort, we aim to deepen our understanding of parallel computing paradigms and their applications in addressing real-world computational challenges.

3. METHODOLOGY

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:

• Divide: Divide the input data set S into disjoint subsets S1, S2, S3…Sk.
• Recursion: Solve the sub-problems associated with S1, S2, S3…Sk.
• Conquer: Combine the solutions for S1, S2, S3…Sk. into a solution for S.
• Base case: The base case for the recursion is generally subproblems of size 0 or 1. Many studies have revealed that to sort N items; it will take Quick Sort an average running time of O(NlogN).

The worst-case running time for Quick Sort will occur when the pivot is a unique minimum or maximum element, and as stated, the worst-case running time for Quick Sort on N items is O(N2). These different running times can be influenced by the input distribution (uniform, sorted or semi-sorted, unsorted, duplicates) and the choice of the pivot element. We have made use of Open MPI as the backbone library for parallelizing the QuickSort algorithm.

Learning message passing interface (MPI) allows us to strengthen our fundamental 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 to perform a given task. An illustration can be communication and coordination by a master process which splits a huge task into chunks and shares 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.

As elaborated in, MPI has two types of communication routines: point-to-point communication routines and collective communication routines. Collective routines as explained in the implementation section have been used in this study.

Algorithm :

- swap() function: This function is used to swap two elements in an array.

- partition() function: This function selects a pivot element from the array and partitions the array into two sub-arrays such that elements less than the pivot are on the left side, and elements greater than the pivot are on the right side. It returns the index of the pivot element.

- quickSortSerial() function: This function recursively sorts the array in a serial manner. It selects a pivot, partitions the array, and recursively calls itself on the left and right sub-arrays.

- quickSortParallel() function: This function is similar to quickSortSerial, but it uses OpenMP directives to parallelize the sorting process. It splits the sorting of the left and right sub-arrays into separate OpenMP sections, allowing multiple threads to execute these sections concurrently.

- Main function:
- It initializes two arrays, arrSerial and arrParallel, of size SIZE with random integers.
- It measures the time taken by the serial QuickSort algorithm to sort arrSerial.
- It measures the time taken by the parallel QuickSort algorithm to sort arrParallel.
- Finally, it prints the time taken by both serial and parallel versions of QuickSort.

4. CODE

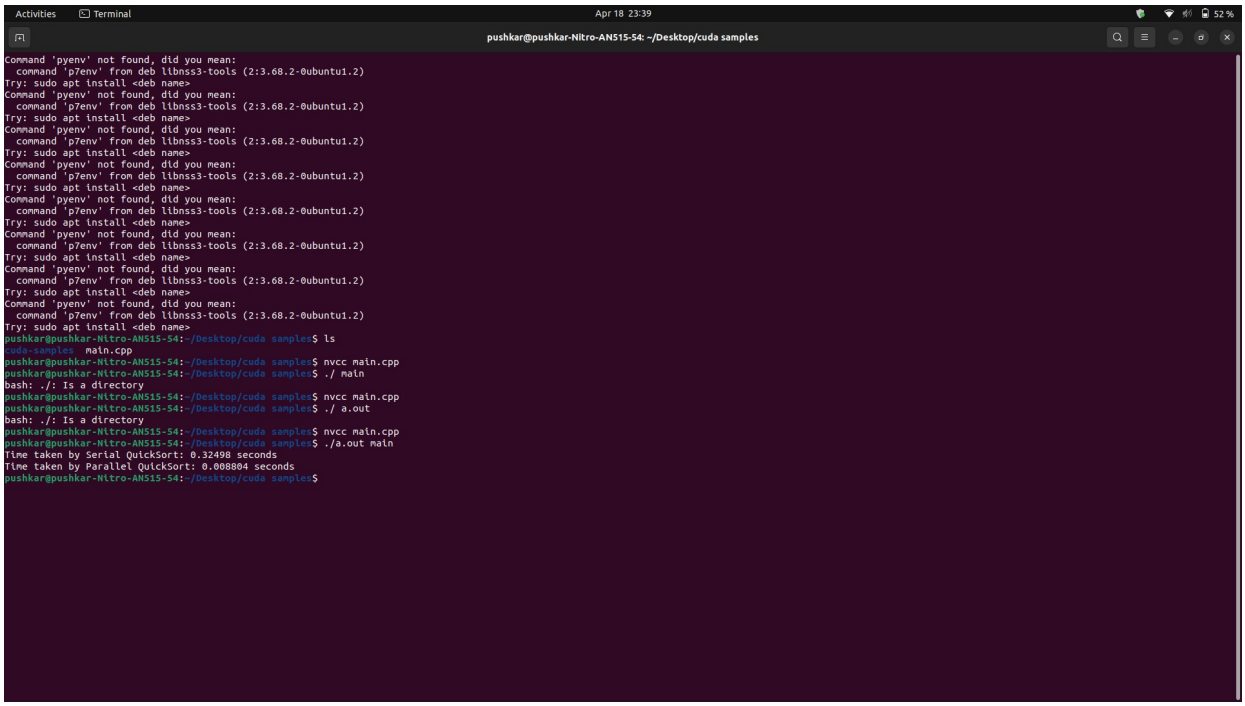
```
#include <iostream>
#include <vector>
#include <ctime>
#include <omp.h>
#include <cstdlib>

void swap(int *a, int *b)
{
   int t = *a;
   *a = *b;
   *b = t;
}

int partition(std::vector<int> &arr, int low, int high)
{
   int pivot = arr[high];
   int i = (low - 1);
   for (int j = low; j <= high - 1; j++)
   {
      if (arr[j] < pivot)
      {
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i + 1], &arr[high]);
   return (i + 1);
}
void quickSortSerial(std::vector<int> &arr, int low, int high)
{
   if (low < high)
   {
      int pi = partition(arr, low, high);
      quickSortSerial(arr, low, pi - 1);
      quickSortSerial(arr,
                  pi + 1, high);
   }
}
void quickSortParallel(std::vector<int> &arr, int low, int high)
{
   if (low < high)
   {
      int pi = partition(arr, low, high);
#pragma omp parallel sections
      {
```

```
#pragma omp section quickSortParallel(arr, low, pi - 1);
#pragma omp section quickSortParallel(arr, pi + 1, high);
        }
    }
}
int main()
{
    const int SIZE = 1000000;
    std::vector<int> arrSerial(SIZE), arrParallel(SIZE);
    for (int i = 0; i < SIZE; i++)
    {
        arrSerial[i] = arrParallel[i] = rand() % SIZE;
    }
    clock_t startSerial = clock();
    quickSortSerial(arrSerial, 0, SIZE - 1);
    clock_t endSerial = clock();
    clock_t startParallel = clock();
    quickSortParallel(arrParallel, 0, SIZE - 1);
    clock_t endParallel = clock();
    std::cout << "Time taken by Serial QuickSort: " << (double)(endSerial - startSerial) /
CLOCKS_PER_SEC << " seconds" << std::endl;
    std::cout << "Time taken by Parallel QuickSort: " << (double)(endParallel - startParallel) /
CLOCKS_PER_SEC << " seconds" << std::endl;
    return 0;
}
```

Output


Activities
Terminal
Apr 18 23:39
52 %
pushkar@pushkar-Nitro-AN515-54: ~/Desktop/cuda samples
Command 'pyenv' not found, did you mean:
command 'p7env' from deb libnss3-tools (2:3.68.2-0ubuntu1.2)
Try: sudo apt install <deb name>
pushkar@pushkar-Nitro-AN515-54:~/Desktop/cuda samples$ ls
cuda-samples main.cpp
pushkar@pushkar-Nitro-AN515-54:~/Desktop/cuda samples$ nvcc main.cpp
pushkar@pushkar-Nitro-AN515-54:~/Desktop/cuda samples$ ./ main
bash: ./: Is a directory
pushkar@pushkar-Nitro-AN515-54:~/Desktop/cuda samples$ nvcc main.cpp
pushkar@pushkar-Nitro-AN515-54:~/Desktop/cuda samples$ ./ a.out
bash: ./: Is a directory
pushkar@pushkar-Nitro-AN515-54:~/Desktop/cuda samples$ nvcc main.cpp
pushkar@pushkar-Nitro-AN515-54:~/Desktop/cuda samples$ ./a.out main
Time taken by Serial QuickSort: 0.32498 seconds
Time taken by Parallel QuickSort: 0.000804 seconds
pushkar@pushkar-Nitro-AN515-54:~/Desktop/cuda samples$