A picture containing symbol, logo, emblem, font

Description automatically generated

**Parallel Programming**

**And**

**Comparison of Sorting Algorithms**

**Group Members:**

Muhammad Mubin Farid (21K-4827)

Ahmed Mustafa (21K-4516)

Shaharyar Shahid (21K-3443)

**Course Instructor : Sir Muhammad Danish Khan**

**Introduction:**

Since the invention of microprocessors back in the 1970s, they have come a long way from having single cores to multiple cores now. A significant increase has been made in the clock speed alongside the number of transistors in a chip. However, we can’t utilize the proper potential of a CPU without great software, this is where Parallel Programming comes into play. Parallel programming is a technique that involves dividing a program into smaller parts and executing them simultaneously. The goal of parallel programming is to reduce the overall execution time of a program by running it on multiple processors or computers at the same time. In this report, we will discuss the three most popular sorting algorithms, namely Quick Sort, Bubble Sort, and Merge Sort, and compare them using Serial and two types of parallel implementations, namely OpenMP, and PThread. We have used C++ language for the driver code and Python to plot our results using a csv file to store the execution times of the algorithms. We have provided large arrays of random unsorted data set to each individual implementation of the sorting algorithms and shared the time complexities of serial, OpenMP and pthread implementations in terms of sorting the data under the description of each algorithm.

**Serial Implementation:**

The simplest type of implementation, where the whole program is executed on a single processor or computer. In serial implementation, a single process execution time is equal to the sum of the execution times of each task. In other words, it is like writing a simple code and just compiling and running it.

**OpenMP Implementation:**

It is a shared-memory parallelization API that is used to parallelize loops and other portions of code. In this implementation, multiple threads are created, each executes a portion of the program. The threads share a common memory space, which allows for efficient communication between them. The code is implemented by including <omp.h> library. “-fopenmp “ enables support for OpenMP directives during compilation, while. Note: Recursion in openMP takes more time than pthread because of task creation again and again.

**Pthread (POSIX Thread) Implementation:**

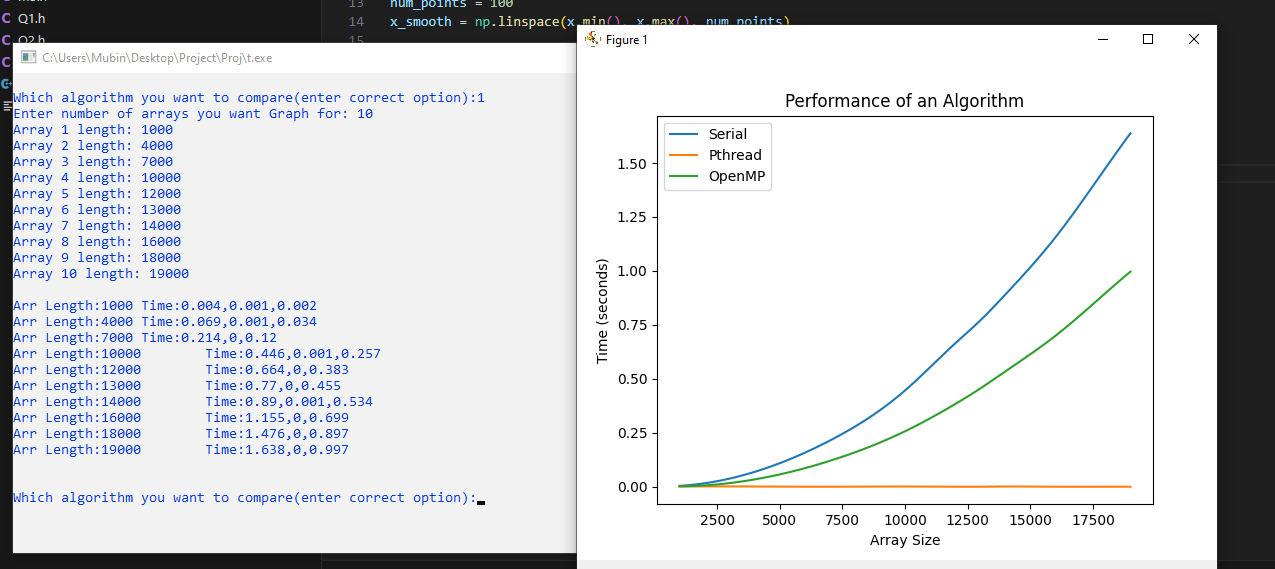
The library <pthread.h> is included that allows for the creation of multiple threads within a program and each thread executes a portion of the program. Unlike OpenMP, programming with pthread requires explicit thread creation, management, and synchronization.”-lpthread” links the program with the pthread library during the linking phase.

**Bubble Sort:**

The simplest and slowest working algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted. Average Case sorting time complexity is O( n2 ).

**Graph Evaluation:**

The graph for bubble sort compares the execution time of the serial, OpenMP, and pthread implementations as the input size increases. The serial implementation shows a significant increase in execution time as the array size grows larger. The OpenMP implementation also experiences an increase in execution time, but the rate of increase is slightly lower compared to the serial implementation. On the other hand, the pthread implementation exhibits almost constant execution time with a very slight decrease. This suggests that both OpenMP and pthread implementations provide improvements over the serial implementation, with pthread being the most efficient among the parallel versions.

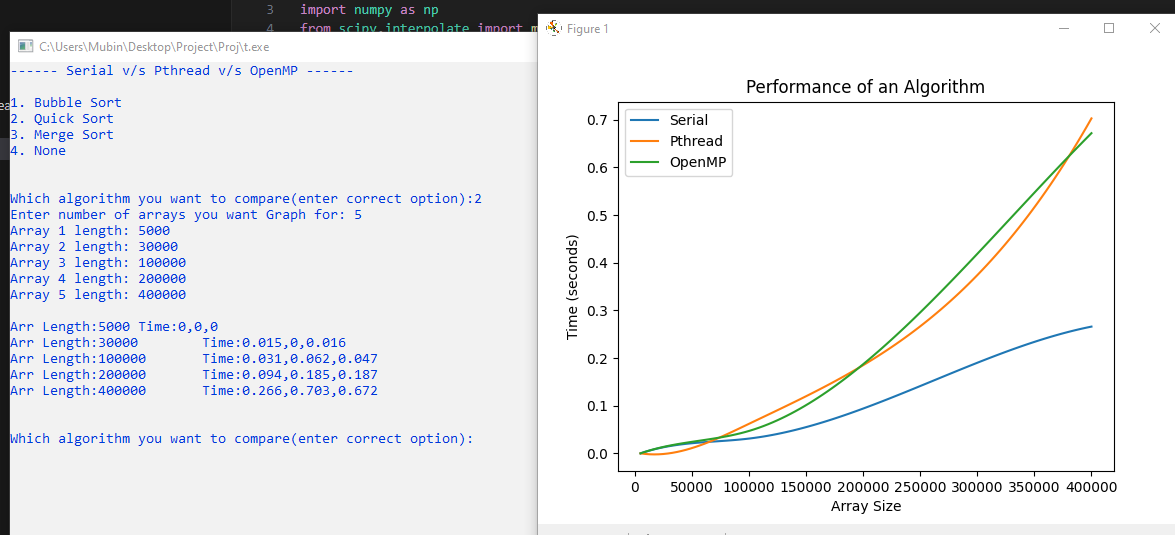


**Quick Sort:**

Quick Sort was developed by Tony Hoare in 1959. It works on a divide-and-conquer algorithm that works by selecting a "pivot" element from the array and partitioning the other elements into two sub-arrays based on whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. Average Case sorting time complexity is O(n log n).

**Graph Evaluation:**

The graph for quick sort indicates that, initially, the pthread implementation outperforms both the serial and OpenMP implementations, demonstrating faster execution times up to an array size of around 30,000. However, beyond that point, the serial implementation becomes faster than the pthread and OpenMP implementations. The performance of the pthread and OpenMP versions remains relatively similar throughout. This suggests that for smaller array sizes, pthread offers the best performance, but for larger array sizes, the serial implementation becomes more efficient.

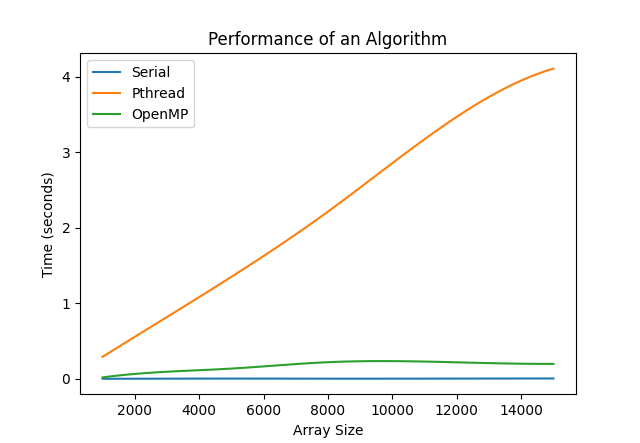


**Merge Sort:**

Merge Sort is a divide-and-conquer algorithm that works by dividing an array into two halves, sorting each half, and then merging the sorted halves back together. Average Case sorting time complexity is O(n log n).

**Graph Evaluation:**

In the graph for merge sort, the execution times for the pthread implementation progressively increase as the array size grows larger. In contrast, the serial and OpenMP implementations maintain a relatively constant execution time regardless of the array size. This indicates that pthread encounters increasing overhead as the input size increases, resulting in longer execution times compared to the other implementations. The serial and OpenMP versions, on the other hand, are able to maintain consistent performance, suggesting their efficiency in handling larger data sets.



**Conclusion:**

Overall, the graphs demonstrate the varying performance characteristics of different parallel implementations (OpenMP and pthread) compared to the serial implementation for each sorting algorithm. These differences are influenced by factors such as the inherent parallelism of the algorithm, input size, and the efficiency of the parallelization techniques used.