Fall 2021

**Course Instructors: Sir Hassan Jamil**

Parallel and Distributed Computing

**Group Members:**

* Anusha Saad – 19K-0281 – Section 4H
* Saman Khan – 19K-0354 – Section 4H

PROJECT REPORT

Comparison of Sorting Algorithms Using: Serial, Pthreads, OpenMP

# **Introduction**

Parallel Programming is still a challenging task in our days. Although there are many powerful parallel programming systems, most of them operate on a relatively low abstraction level (e.g. POSIX threads, MPI). The specification of OpenMP promised advances in this regard, and has provided a relatively smooth way to incrementally parallelize existing programs. Nevertheless, OpenMP is not without problems and rough edges.

Sorting data has always been one of the key problems of computer science. Many sequential algorithms have been suggested. Our project aims to find out which programming method is best suited for each of these. Serial, Pthreads, and as well as OpenMP have their own pros and cons. While some may be best suited, others take very long and reduce efficiency.

# **Components of the Project**

Using the concepts on multiprogramming taught in the course of Parallel and Distributed Computing, we have implemented six sorting algorithms. We have then compared each technique’s efficiency by calculating the runtime. Comparison is made and patterns of unsorted data are studied to reach a conclusion.

## **Algorithms:**

1. Merge Sort
2. Bubble Sort
3. Insertion Sort
4. Shell Sort
5. Quick Sort
6. Heap Sort

## **Tools:**

* **Language:** C/C++
* **Platform:** Oracle VM VirtualBox
* **System:** Ubuntu

# **Methodology**

## **Merge Sort**

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. The **merge ()** function is used for merging two halves.

The graph below shows the difference in the running times of merge sort on different data sets when implemented serially, with p-threads and with open-mp.

From the above graph we can conclude that:

In the case of merge sort, as we increase the data and check for the runtime, openmp works the most efficiently, while pthreads is the worst and serial lies in between.

## **Bubble Sort**

Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of Ο(n2) where **n** is the number of items.

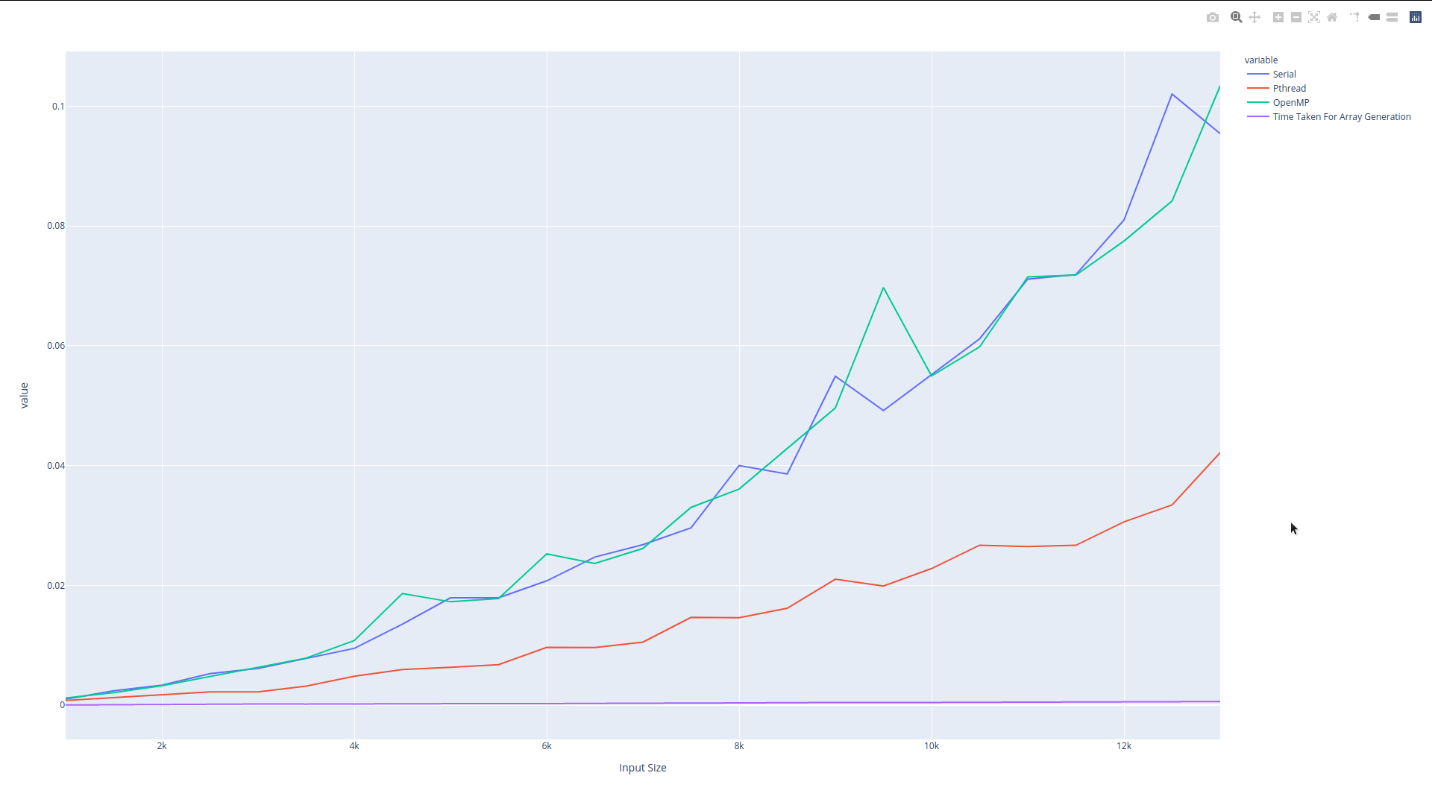
Bubble sort is a simple sorting algorithm. This sorting algorithm is comparison-based algorithm in which each pair of adjacent elements is compared and the elements are swapped if they are not in order. This algorithm is not suitable for large data sets as its average and worst case complexity are of Ο(n2) where **n** is the number of items.

Bubble sort is the probably the most inefficient sort algorithm in existence, and throwing "multithreading" at a poor algorithm to make it faster (when better single-threaded algorithms exist) is a poor approach. Hence, both Pthreads and OpenMP take longer than Serial Bubble Sort which further proves the point.

## **Insertion Sort**

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Insertion Sort is an alogirthm where each iteration is dependent on other iterations which makes it a not-so-good choice for multithreading.

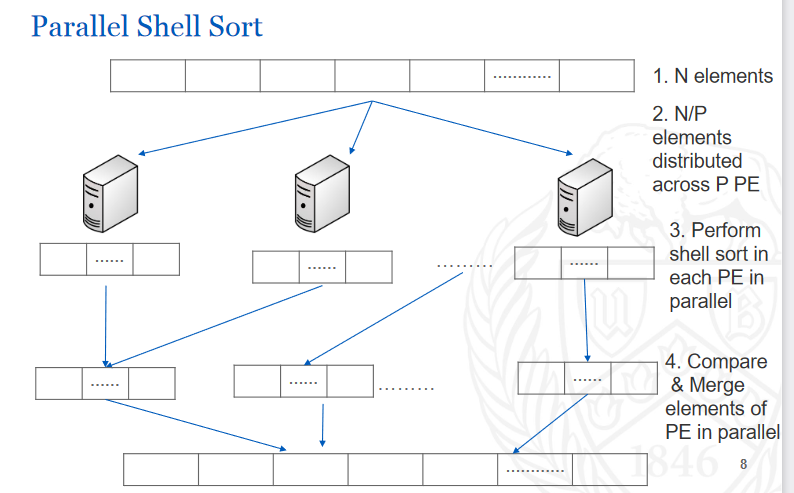


As illustrated above, Insertion Sort takes the longest using OpenMP and least runtime using Pthreads. Whereas, Serial lies somewhere in the middle.

## **Shell Sort**

Shell sortis a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort*,* if the smaller value is to the far right and has to be moved to the far left.

The algorithm used to apply multithreading in Shell Sort is illustrated as follows:



Shell Sort takes the longest when implemented with the technique of Pthreads. However, it takes the least amount of time using OpenMP and Serial lies somewhere in the middle. As we increase the data set values, the trend continues.

# **Quick Sort:**

Quicksort (sometimes called partition-exchange sort) is an efficient sorting algorithm, serving as a systematic method for placing the elements of an array in order. When implemented well, it can be about two or three times faster than its main competitors, merge sort and heapsort. It is a fast [sorting algorithm](https://brilliant.org/wiki/sorting-algorithms/) that takes a [divide-and-conquer](https://brilliant.org/wiki/divide-and-conquer/) approach to sorting [lists](https://brilliant.org/wiki/array). While sorting is a simple concept, it is a basic principle used in complex programs such as file search, data compression, and pathfinding. Running time is an important thing to consider when selecting a sorting algorithm since efficiency is often thought of in terms of speed. Quicksort has a very slow worst-case running time, but a fast average and best-case running time.

The graph below shows the difference in the running times of quick sort on different data sets when implemented serially, with p-threads and with open-mp.

From the above graph we can conclude that:

In the case of quick sort, as we increase the data and check for the runtime, openmp works the most efficiently, while pthreads takes time and is the worst according to the time and serial is the median of both.

# **Heap Sort:**

Heapsort is a comparison-based sorting algorithm. Heapsort can be thought of as an improved selection sort: like that algorithm, it divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element and moving that to the sorted region. It is a comparison based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the maximum element and place the maximum element at the end. We repeat the same process for remaining element. The graph below shows the difference in the running times of heapsort on different data sets when implemented serially, with p-threads and with open-mp.

From the above graph we can conclude that:

In the case of heap sort, as we increase the data and check for the runtime, Serial works the most efficiently, threads is the worst and openmp is the median of both. They have a very little difference in the runtime.

# **Project Codes**

Project Codes of all algorithms are attached in the zip file blow:



Project Output Screenshots are attached in the zip file below:



# **Project Aspect in Current/Future Technologies**

Sorting algorithms are most widely used for the sorting of different kind of data, when using the sorting algorithms in a company one can tell which would be beneficial and cost effective for their organization whether to use the serial way or the parallel programming way.

In the future, we plan to implement and test some of the ideas and additions to the specification we have suggested into an actual compiler, as well as investigate other algorithmic problems beyond sorting.

# **References**

* <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.5003&rep=rep1&type=pdf>
* <https://cse.buffalo.edu/faculty/miller/Courses/CSE633/prasad-salvi-Spring-2017-CSE633.pdf>
* <http://www.hpcc.unn.ru/mskurs/ENG/PPT/pp10.pdf>