Quest Journals
Journal of Software Engineering and Simulation
Volume 7 ~ Issue 9 (2021) pp: 09-11
ISSN(Online) :2321-3795 ISSN (Print):2321-3809
www.questjournals.org

Research Paper

Parallelization and Comparison of Sorting Algorithms

Archit Aggarwal
Student, School of Computer Science and Engineering, VIT University, Vellore, Tamil Nadu
architaggarwal21@gmail.com

ABSTRACT: *Sorting is the foremost technique that one comes across while performing operations on a computer. Data that is sorted is easier to comprehend and work upon for further analysis. A comparison is performed on three sorting algorithms: Merge Sort, Quick Sort and Bubble Sort using methods like OpenMP, POSIX Threads and Serial implementations.*

KEYWORDS: *Execution Time, Multi-Core Processors, OpenMP, Parallelization, POSIX Threads, Serial*

Received 08 September, 2021; Revised: 21 September, 2021; Accepted 23 September, 2021

I. INTRODUCTION

Since the creation of microprocessors in the late 20th Century increased, their performance has been achieved by increasing the clock speed. Until early 2000, the increase in clock speed was static. Every two years, the number of transistors that would fit on a chip would double.

It is ubiquitous with a multi-core processor in most computers, laptops, workstations, servers, phones. The addition of cores does not guarantee high performance; in some cases, it can result in a worse performance because the individual cores may not have a clock speed as high as that of a single-core processor. The software running on the hardware must utilize the cores that are being offered in order to get enhanced performance.

Developing algorithms and software that can efficiently use multiple cores can be done in many language frameworks. For experimentation, C was used as the primary language; the reasoning for using C was that C is used in many areas of development, and it also supports the two threading techniques - OpenMP and POSIX Threads.

II. DEFINING OPENMP AND POSIX THREADS

2.1 OpenMP

OpenMP implements multithreading, wherein a primary thread (a process) creates some sub-threads to divide the required task among them. These sub-threads perform their divided tasks concurrently, along the runtime environment to allocate these sub-threads to the different idle processors of the system. *OpenMP* is a widely used API that provides support for multithreading in popular programming languages.

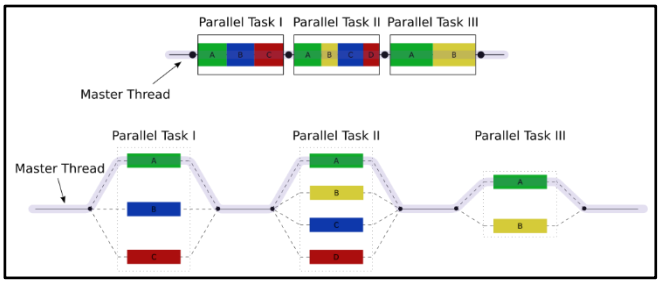

Figure. 1. OpenMP Implementation
Source: Wikipedia (https://en.wikipedia.org/wiki/OpenMP)

2.2 POSIX Threads

POSIX Threads *(Pthreads)* are a C/C++ library. It helps in creating simultaneous process flow and facilitates faster process flow by increasing speed through parallel and distributed processing. Pthreads are used to achieve possible parallelism in programs wherein threads are created by making calls to POSIX threads API and the task is subdivided among these threads. The task needs to be divided into discrete tasks that do not depend on each other so as to perform concurrently.

III. IMPLEMENTATION

Parallelization is used to make processes and tasks work much faster by distributing the work among different processors. Parallelization techniques are very prevalent in recent times to make our systems more efficient than ever before. In this paper, we will be comparing the performances of traditional sorting algorithms, namely – Bubble Sort, Quick Sort, Merge Sort.

The above algorithms will be tested in three different ways to determine the extent of parallelism and determine whether parallelization is needed. The three manners in which the algorithms will be Serial, OpenMP, POSIX Threads.

The parallelization performance is then compared using the algorithms' execution times for sorting by the methods mentioned above. The algorithm is tested for sorting an array of 1000 elements created randomly in all the methods.

IV. RESULTS

Table 1. Execution Time Summary

Sorting Algorithms	Serial	OpenMP	POSIX Threads
Bubble Sort	8ms	84ms	49ms
Merge Sort	35ms	17ms	16.5ms
Quick Sort	25ms	15ms	2ms

The above table summarizes the findings of the study. As shown in the table, parallelism does not necessarily guarantee lower execution times. Software and algorithms need to be modified to be able to perform better by parallelism. Some processes are best carried out serially.

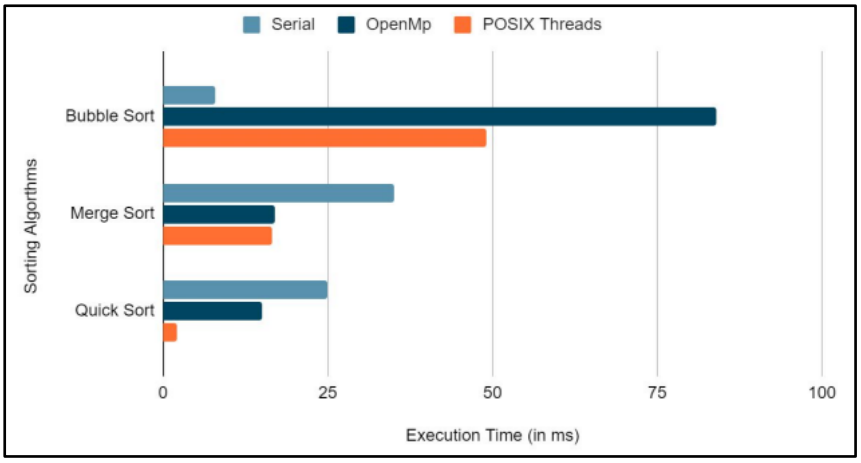

Figure. 2. Execution Time vs Sorting Algorithms

According to Figure 2, Bubble Sort is the least efficient sorting algorithm upon parallelism. The execution time of Bubble Sort is much less for Serial Implementation. It may also be noted that efficiency of the algorithms is dependent on the method of parallelism as shown in Figure 3.

POSIX Threads method performed best for Quick Sort and the execution times while lower than OpenMP in most cases did not decrease the execution time for the Bubble Sort.

OpenMP performed reasonably well to reduce execution times of both the Quick and Merge Sort when compared to serial implementations, however there was a significant delay notices when OpenMP was coupled with Bubble Sort.

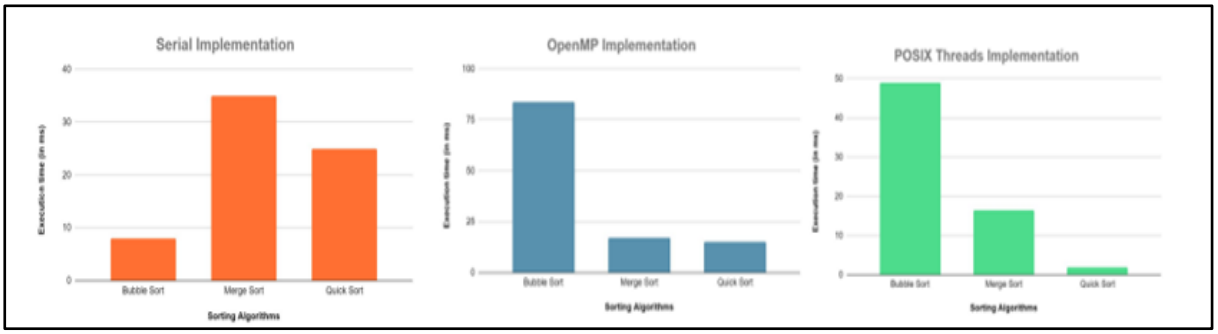

Figure. 3. Comparison of Execution Times for different Implementations

V. CONCLUSION

Upon performing the above implementations, we can infer that parallelization of algorithms may not always be beneficial in increasing the efficiency and performance of the algorithm. In Bubble Sort Algorithm, it was evident that it best works in a serial manner and takes more compute time in any other parallel paradigm.

Furthermore, it was proved that POSIX threads implementation proved to give better results than the OpenMP implementation among the parallel processing paradigms.

Hence, it can be concluded that parallelization may not constantly improve the performance of an algorithm, and serial implementation can be the viable option.

REFERENCES

[1]. N C, Senthilkumar&Parappilly, J.J. & Shilpa, S.. (2014). A survey paper on sorting techniques in parallel databases. International Journal of Applied Engineering Research. 9. 2611-2618.

[2]. Bozidar, D., &Dobravec, T. (2015). Comparison of parallel sorting algorithms. *arXiv preprint arXiv:1511.03404*.

[3]. E. Solomonik and L. V. Kalé (2010), "Highly scalable parallel sorting," 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), , pp. 1-12, doi: 10.1109/IPDPS.2010.5470406.

[4]. Nancy Amato, Ravishankar Iyer, Sharad Sundaresan, and Yan Wu. 1998. A Comparison of Parallel Sorting Algorithms on Different Architectures. Technical Report. Texas A & M University, USA.