National University
of computer and emerging sciences

Project Report

CS2006 Operating System

Semester Project

Members

Syed Ahmed Mahmood 20K0153

Muhammad Qasim Fuzail 20K0157

Section D

Submitted to:

Ms Tania Iram

Department of Computer Science BS(CS)

FAST-NUCES Karachi

Comparison between Process And Threads

Objective

The objective of this project was to find out what would happen if we parallelized 3 common algorithms, namely Merge Sort, Quick Sort and Harmonic Progression. We wanted to know if it would be worth multithreading these algorithms or not and at what point, meaning at what size of a dataset, would we actually notice any time differences.

Project Details

The way we went about this project was we first implemented a simple single threaded (which would be a process) version of all the 3 algorithms and then we used the POSIX thread (pthread) library to multithread these 3 algorithms in the C language and then we used shell(BASH) scripting to automate a process where first it asks the user for a minimum and maximum number of elements in the array, then it runs the algorithms on the range of datasets given by the user while scaling the size appropriately, and it stores the time required to run the algorithm for each size of dataset in a .csv file, after the algorithm is run on the range of datasets, the .csv file is used by a Python program to generate a line graph for the 3 algorithms. These 3 graphs can be observed more accurately at runtime but images of these graphs can be seen later on in the report.

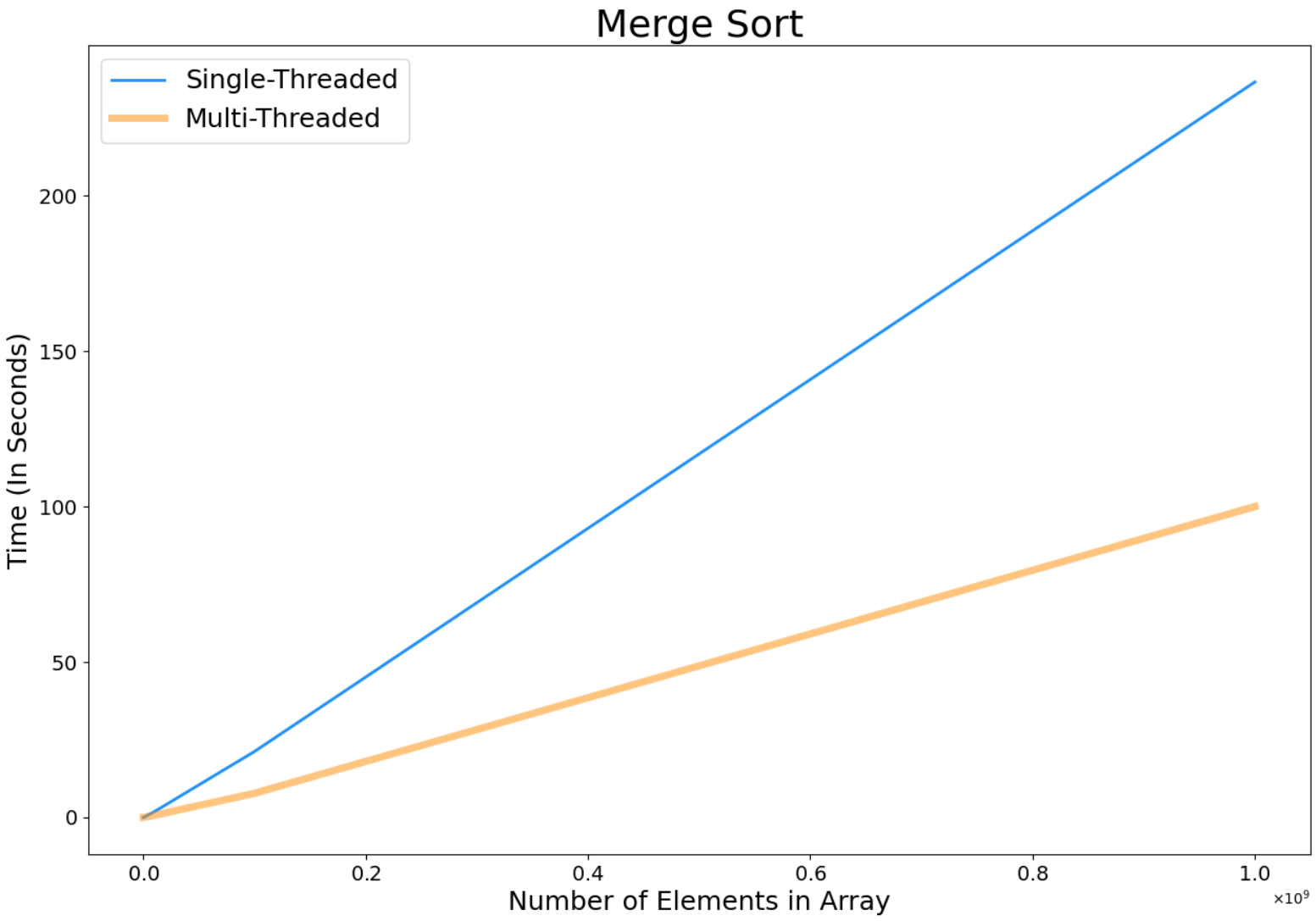
Tools used and Github Repository

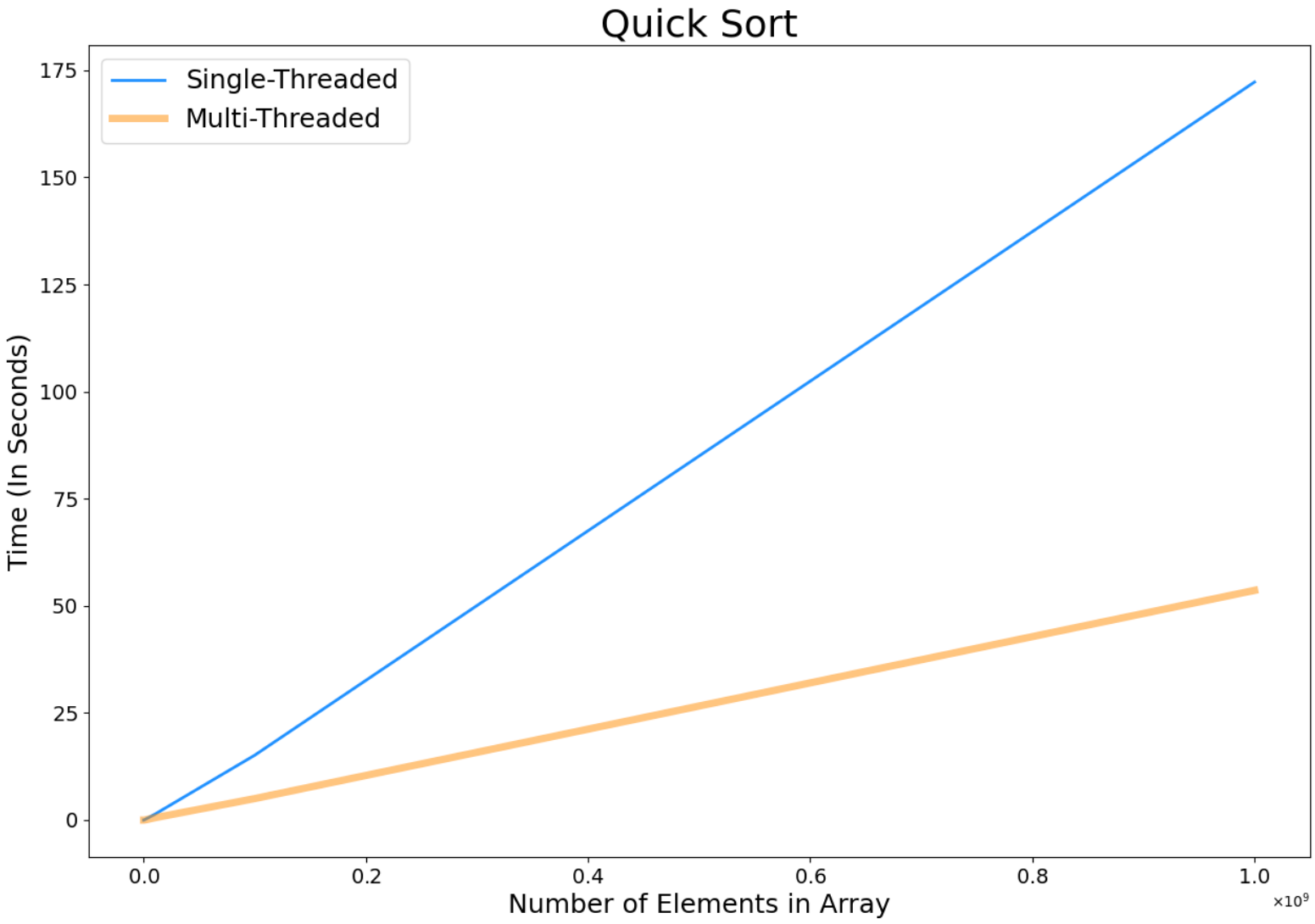
- Visual Studio Code
- C Language
- BASH language
- Python language
- Git (the project can be `viewed on Github`)

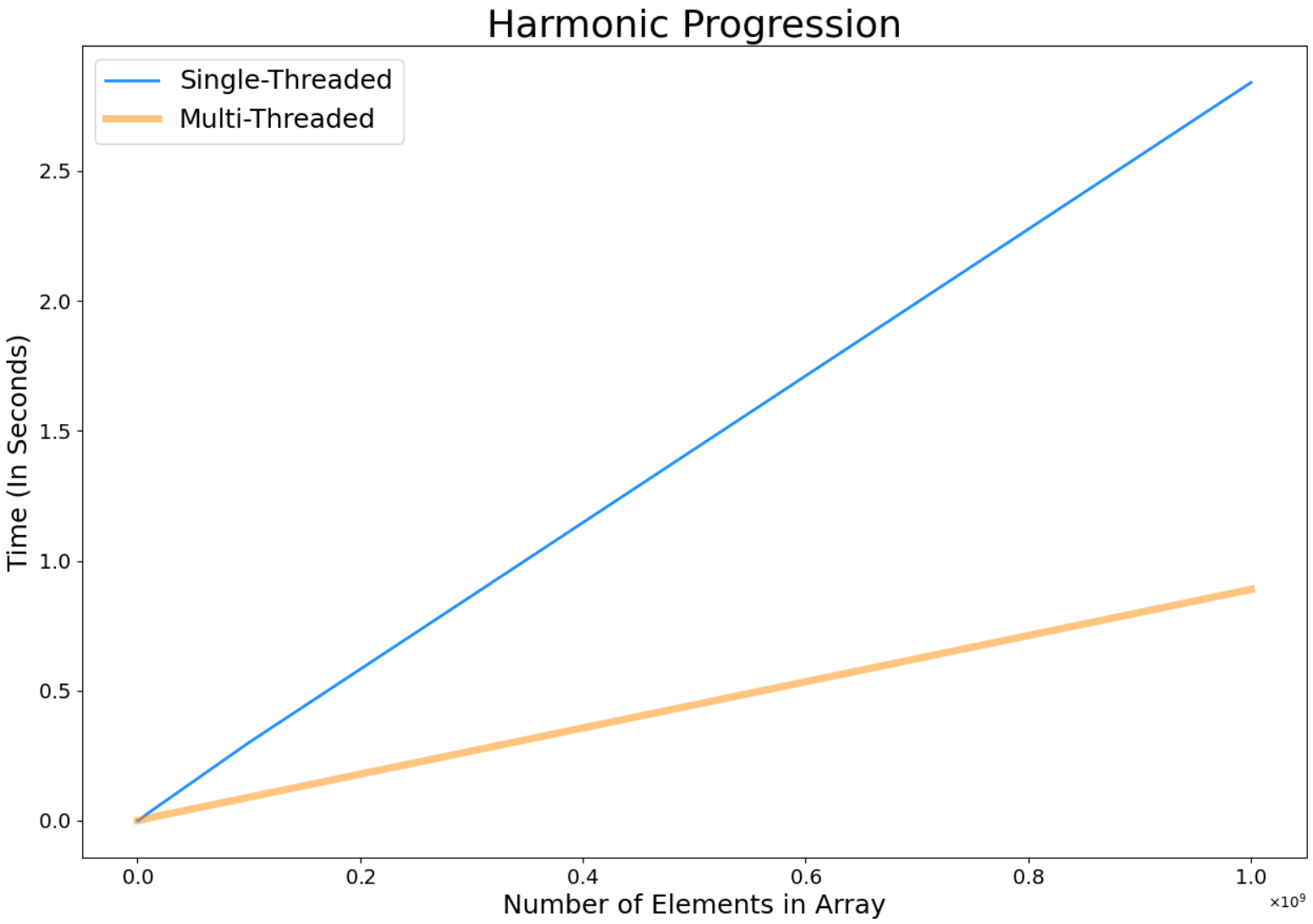
Results

The shell script automatically ran the 6 programs (3 multithreaded and 3 singlethreaded) on the ranges of datasets from 1 to 1000,000,000 Elements in Array with a 10x increase or an increase of 1000% between each iteration. We observed that when the graphs were plotted the relationship between *Time* and *Number of Elements in Array* seemed approximately linear so we can tell that the reciprocal of the gradient for these graphs would be the the rate (*Number of Elements in Array*/*Time*), thus the steeper the gradient looks, the slower the rate of that algorithm would be so it is obvious to see that all the 3 algorithms experienced a great speed up due to multithreading. An another important observation in all of the graphs is that the gradient keeps increasing as the *Number of Elements in Array* increase meaning the rate decreases for both multithreaded and single threaded algorithms but it decreases faster for single threaded algorithms, which is why when the graph is plotted for a smaller range of *Number of Elements in Array* (such as till 1,000,000 for Merge Sort) the lines do not diverge as much as they do for a much larger range such as the ones shown in the graphs below.

Graphs


Merge Sort
Single-Threaded
Multi-Threaded
Time (In Seconds)
0
50
100
150
200
0.0
0.2
0.4
0.6
0.8
1.0
Number of Elements in Array
×10^9


Quick Sort
Single-Threaded
Multi-Threaded
Time (In Seconds)
175
150
125
100
75
50
25
0
0.0
0.2
0.4
0.6
0.8
1.0
Number of Elements in Array
×10^9


Harmonic Progression
Single-Threaded
Multi-Threaded
Time (In Seconds)
0.0
0.5
1.0
1.5
2.0
2.5
0.0
0.2
0.4
0.6
0.8
1.0
×10^9
Number of Elements in Array

Conclusion

The results above lead us to conclude that as the size of the dataset (*Number of Elements in Array*) increases, the benefits of multithreading become more visible and apparent, when the time required to execute the multithreaded algorithms slows down less quickly compared to single threaded algorithms. Especially in this age of “Big Data”, where the size of data available and required to be processed is rapidly increasing, it would be highly beneficial to parallelize commonly used algorithms so more time can be spent on using this data then it is spent on processing it.