COP290 Assignment 2
REPORT

Anshik Sahu 2021CS10577
Aryan Dhaka 2021CS50597
Sanya Mittal 2021CS10565

Abstract

The assignment required us to give the cumulative count of words across N text files with the use of hashmaps. We had to then compare the performance, in terms of time elapsed to complete the operation on the same input files, across single threading and multiple threading, where multithreading was pre-implemented.

1 Introduction

The following points are covered in the report:

- Comparative analysis of single and multi-threading ie Part 2 and Part3
- Implementation of locks to avoid race condition
- Data of file size Vs time (plots graphs and tables for the test)
- Cause of difference in speeds of single and multi-threading

Part 1 of the assignment utilizes pointers to follow the process flow and track changes in function, while contexts are analogous to CPU registers and related functions like ucontext.

2 Threading

.

2.1 Single vs Multi-threading

Single threading executes the process one at a time (concurrency) ie in the problem statement only one text file is handled at a time and other files are switched between one another but at any time only one file is being read. It uses the functions `mythread_create`(makes the thread), `mythread_join` (waits for the context to be executed after which it eliminates it from the stack), `mythread_yield`(which executes the part of the context)

Multi threading practices parallelism and executes multiple files under the same function, in this case, all the input text files are read simultaneously. The functions for this are already provided.

2.2 Locks and Race condition

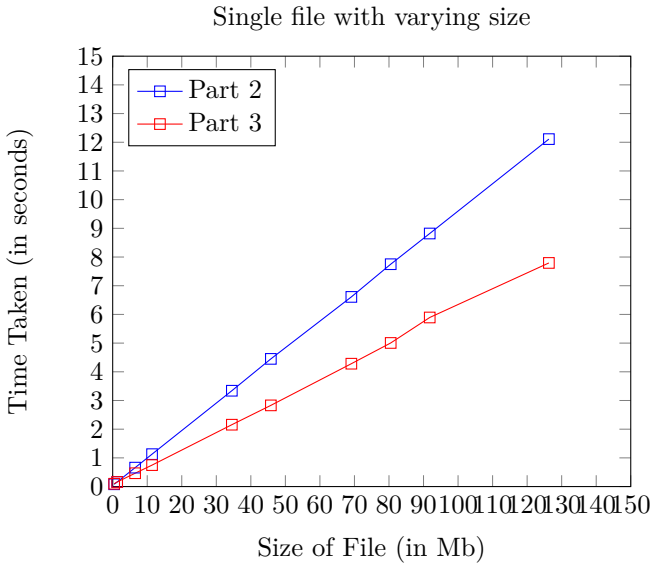
The introduction of locks in the single thread program rectified the race conditions, where two contexts were trying to work on the same word thus, the latter was not receiving the updated value by the first, rather both were simultaneously racing for the word. Thus when we were running two files with the same words then the count of the word was coming out to be incorrect because of race conditions. The introduction of locks made sure that only one context was incrementing the count at a time, thus the values were corrected. Thus the output varies. The output varies because post introduction of locks because now if a context is incrementing a word count, the other waits until the first has completed working on that word. It continuously yields until the lock has been released by the previous context. And only then lets the new context have the lock.

2.3 Comparative Testing

The types of threading are compared on a variety of factors with the variables being the number of files, file size and cumulative file size

2.3.1 Single Varied size file

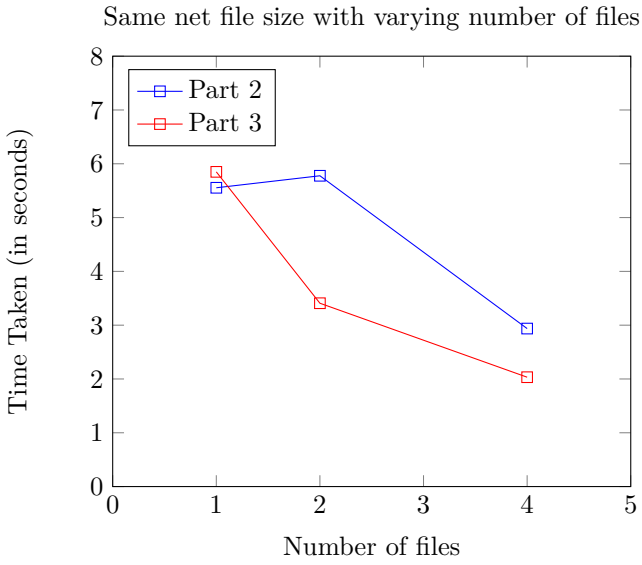
File Size(Mb)	Single-thread time(s)	Multi-thread time(s)
0.43	0.08	0.10
1.3	0.16	0.16
6.5	0.66	0.46
11.4	1.13	0.75
34.5	3.34	2.16
45.8	4.45	2.83
69.1	6.61	4.28
80.5	7.75	5.01
91.8	8.82	5.89
126.3	12.11	7.79



Observation: Multi-threading is faster because more CPU resources are at the disposal

2.3.2 Same net file size with varying number of files

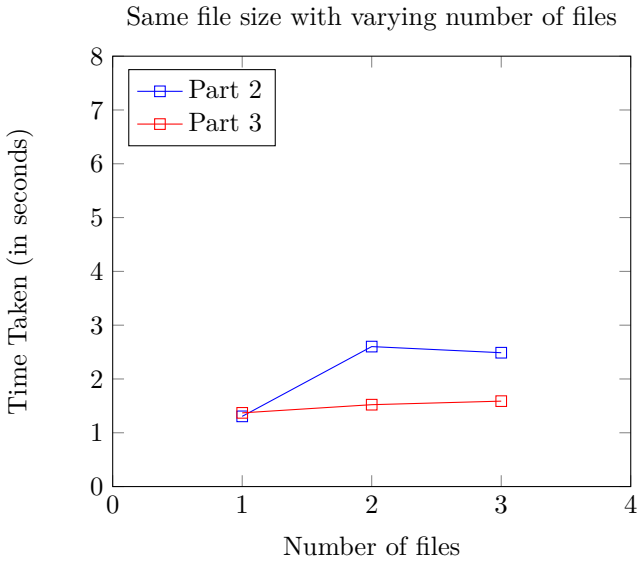
Number of Files	Single-thread time(s)	Multi-thread time(s)
1	5.554	5.850
2	5.777	3.408
4	2.937	2.033



Observation : Here one big file is step-wise split into 2 parts at each step. Multi-threading is faster for more files as they are executed simultaneously whilst the total characters across all files (total size) remains the same.

2.3.3 Same single file size with varying number of files

Number of Files	Single-thread time(s)	Multi-thread time(s)
1	1.305	1.370
2	2.602	1.522
3	2.488	1.370



2.4 Why Multi-threading is faster?

Multi-threading employs the use of more CPU power so the task that is, reading files is done faster due to the allocation of more resources. To simplify things, the procedure reads and processes more than one file at the same time, which ultimately speeds things up. This is the advantage of parallelism over concurrency, wherein more than one thing can be processed at the same time, and all the cores of the CPU are put to use parallel based on the number of threads. The creation of a new thread does take up some time, but this is more than made up for by the benefits described in the previous bullet point. This is also the reason why we observe relatively small time differences in small files, sometimes multi-threading even takes more time.

3 Team Contribution

Name	Token
Anshik Sahu	10
Aryan Dhaka	10
Sanya Mittal	10

Anshik worked on hashmaps and threads. He implemented the hash table with keys and linked lists associated with each key, hashmap operations including get, put and thread operations including yield, create and join. He also worked on makefiles and race conditions, where the count values are incorrect when two contexts try to acquire the same key.

Aryan worked on the theoretical aspects of threading and he developed the report on Latex, using tables and graphs. He deeply understood the differences

in concurrency and parallelism and thus performed a comparative analysis in single and multi-threading along with data testing, and compared the graphs obtained via running the codes with the ideal scenario.

Sanya worked on the first part which was about storing and restoring contexts, on locks to rectify counts in single-threaded program, and in data testing for comparative analysis between single and multiple-threaded programs. She worked on the theoretical understanding of how and why locks rectify race condition errors. She also developed doxygen file for the code.

We also worked collaboratively and helped each other with their aspects. We learned git and had a good experience with collaborative coding as well.