COP290 Lab 2

Anish Banerjee Aman Singh Dalawat Jay Patel

February 19, 2023

1 Introduction

In this report, we perform a comparative analysis of concurrency and parallelism. **Concurrency** refers to the ability of a computer system to execute multiple tasks using shared resources in such a way that at a given instant of time only one task is running. **Parallelism** is the ability to execute multiple tasks or processes at the same time using multiple processors or cores.

2 Comparative Analysis

To time the files, we have used the *time* command on the shell. It provides us with three metrics

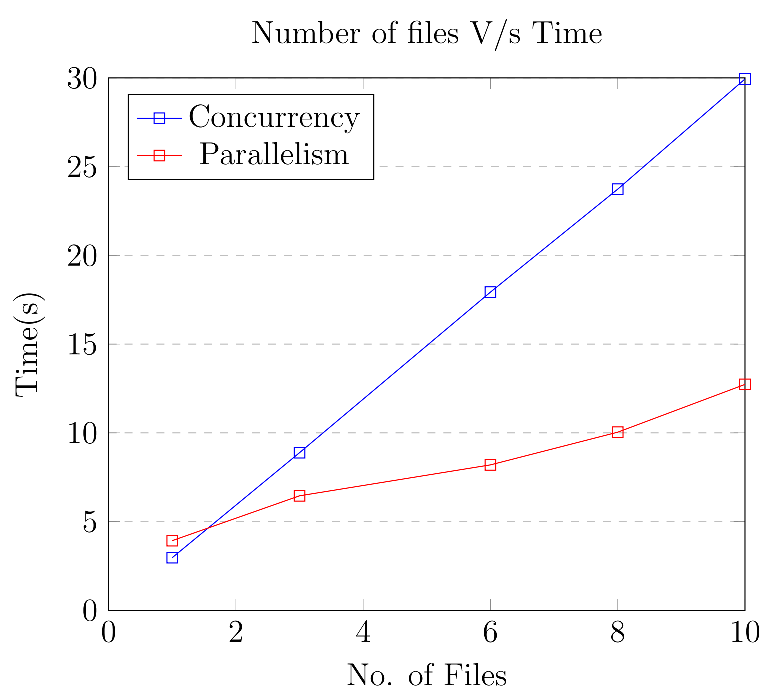
1. **real** is wall clock time - time from start to finish of the call.
2. **user** is the amount of CPU time spent in executing code that is written in the program itself.
3. **sys** is the amount of CPU time spent in executing operating system functions on behalf of the program.

Thus, in our analysis, we use real times to compare concurrency with parallelism.

2.1 Number of Files V/s Time

Number of files	Time(Concurrency)	Time(Paralellism)
1	real 2.968s user 2.644s sys 0.324s	real 3.926s user 3.516s sys 0.410s
3	real 8.882s user 7.885s sys 0.996s	real 6.452s user 14.899s sys 2.018s
6	real 17.929s user 15.841s sys 2.068s	real 8.195s user 33.042s sys 7.040s
8	real 23.730s user 21.224s sys 2.504s	real 10.037 user 48.944s sys 12.401s
10	real 29.937s user 26.450s sys 3.427s	real 12.729s user 1m 9.223s sys 21.343s

Table 2.1: Here files with 1 million lines (70mB) each are used

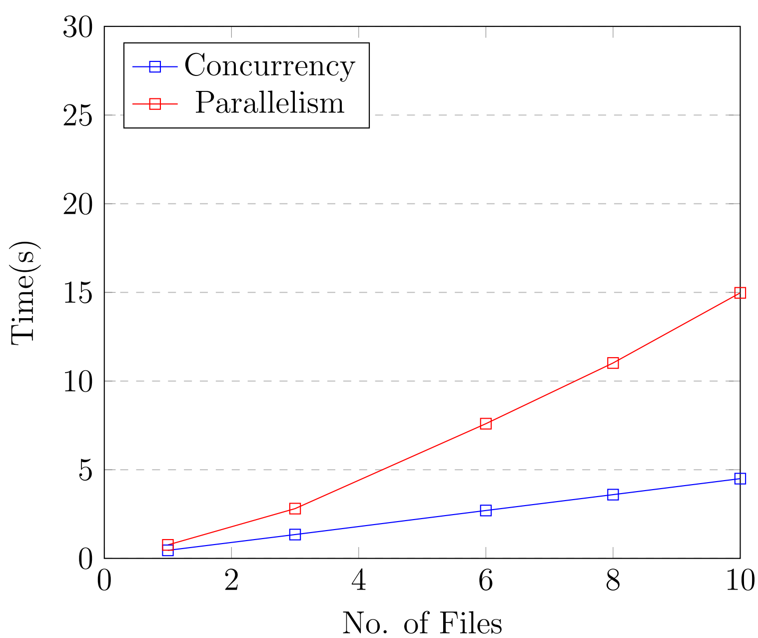


2.2 Same Word Repeated Multiple Times V/s Time

Number of files	Time(Concurrency)	Time(Parallelism)
1	real 0.454s user 0.405s sys0.048s	real 0.761s user 0.651s sys0.082s
3	real 1.344s user 1.220s sys 0.124s	real 2.807s user 2.595s sys 1.991s
6	real 2.701s user 2.420s sys 0.281s	real 7.596s user 8.119s sys 11.310s
8	real 3.595s user 3.215s sys 0.380s	real 11.019s user 12.691s sys 21.746s
10	real 4.497s user 4.085s sys 0.412s	real 14.974s user 17.555s sys 35.812s

Table 2.2:Here the same word "hello" repeated 2 million times

Number of files V/s Time

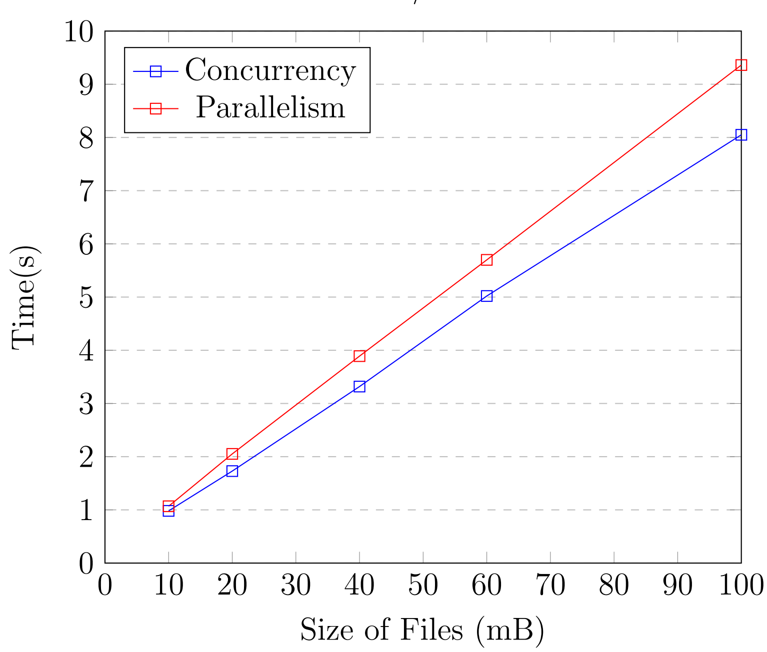


2.3 Size of Files V/s Time

Size of file	Time(Concurrency)	Time(Parallelism)
10mB	real 0.981s user 0.791s sys 0.072s	real 1.068s user 0.942s sys 0.112s
20mB	real 1.729s user 1.519s sys 0.157s	real 2.052s user 1.832s sys 0.171s
40mB	real 3.318s user 3.106s sys 0.201s	real 3.890s user 3.493s sys 0.371s
60mB	real 5.020s user 4.689s sys 0.305s	real 5.699s user 5.244s sys 0.428s
100mB	real 8.050s user 7.491s sys 0.548s	real 9.361s user 8.654s sys 0.636s

Table 2.3: Size of files V/s Time

Size V/s Time



System Specification used to run above files:
Processor: Ryzen 7 5700U 8 cores
RAM: 16GB DDR4

3 Observations

- In case 2.1, parallelism takes much less time as compared to concurrency as the number of files increases
- In case 2.2, parallelism takes more time than concurrency as the file size increases
- In case 2.3, parallelism and concurrency take almost same time (though, concurrency is slightly better)

4 Why are the Differences Observed?

- **Case 2.1** :In concurrency, we are running files with the help of multiple threads but at a given time only one of the threads is working and reading a specific file. In parallelism, we have multiple threads running on multiple files, and at a given instant multiple files can run, so the overall time when running more than one file can be seen to be a lot less. In the case of only one file running we see there is not much difference in the running time because at a given instant only a single file is running in both implementations.
- **Case 2.2**:In this case we see the code with concurrency runs considerably faster than the parallelism-based code, even on multiple files. This is because of the implementation of locks in the case of parallelism. Since many files are being read using multiple threads at the same time, all of them try to increment the count in the bucket of the same word, resulting in contention. Hence at a given instant only one of them is able to actually increment the word in the bucket. This defeats the whole purpose of running multiple files at the same time as the actual work is done only for one file at a time. Hence in this case, parallelism-based implementation falls behind concurrency-based implementation by a larger margin because of the complexity of repeated acquisition and release of locks.

- **Case 2.3**:In the case of a single file running, where the task is not necessarily computationally intensive, concurrency slightly outperforms the approach of parallelism because the overhead of managing multiple threads or processes outweighs the performance gains achieved by the parallelism approach. Additionally, parallelism may not be useful if the task is not divisible into smaller sub-tasks.

5 Conclusion

Thus, we conclude that both concurrency and parallelism have their own advantages and disadvantages. In the case of running several files with dissimilar words, it is better to use parallelism, whereas while handling similar words, it is better to use concurrency.

6 Questions in Part 2 Checkpoint 3

6.1 Does the output vary?

Yes, the output is lesser than the one obtained while running without mythread_yield().

6.2 Why do you think the output varies?

The output varies because of the **race conditions** created due to the swapping of contexts in mythread_yield(). When we are running mythread_yield on a given word, say *hello*, the execution of the context is paused. Assume that the initial count of *hello* is 5. Then its value is not updated to 6 before swapping the context. Now when a new context encounters *hello*, then it reads its count as 5 and not 6. So it will update the value as 6 when the context returns, resulting in lesser counts. This problem can be solved by the use of locks.

Group Work Assessment

Name	Number of Tokens	Work done
Aman Singh Dalawat (2021CS50610)	10/30	Hashmap, Comparative analysis
Anish Banerjee (2021CS10134)	10/30	Part1, mythread_yield, locks, LaTeXreport
Jay Patel (2021CS50617)	10/30	Linked list, mythread_create, mythread_join, Documentation using Doxygen