DAT470 Assignment 2

Rikard Roos
Olof Lindberg

March 2025

Problem 1

a)

By Amdahl's law we have that the speedup S of parallelizing a process is proportional to the fraction of the running time that makes use of the parallelization p and the number of CPUs s. The formula of Amdahl's law is given below.

$$S(s) = \frac{1}{1 - p + \frac{p}{s}}$$

Since 70% of the algorithm is fully parallelizable our p is 0.7 and the speedup is calculated by:

$$S(s) = \frac{1}{0.3 + \frac{0.7}{s}}$$

The speedup for each of the number of processors is given below.

Processors(s)	Speedup
2	1.539
4	2.105
8	2.580
16	2.909
32	3.106

b)

As the number of processors s approaches infinity, the fraction $\frac{p}{s}$ approaches 0. Hence we get the following formula for speedup.

$$S(s) = \frac{1}{1 - p} = \frac{1}{0.3} \approx 3.33$$

c)

The theoretical value given by Amdahl's law assume that the parallelizable tasks are optimally distributed without any delays. In reality, parallelization always comes with overhead such as creating workers, distributing tasks, uneven distribution, synchronizing results etc. Therefore, we should not expect the speedup to reach the theoretical value. In reality, the speedup will be lower due to these overhead factors.

Problem 2

b)

```
Top 10 words:
the: 148499457
of: 89408738
and: 72120653
to: 61762640
a: 51211506
in: 44714455
that: 24646774
was: 24224366
is: 19939759
I: 19103566
```

c)

The major parts of the main function consist of these loops:

```
files = [get_file(fn) for fn in get_filenames(path)]

for file in files:
    file_counts.append(count_words_in_file(file))

for counts in file_counts:
    merge_counts(global_counts,counts)
```

The running time of the functions `get_file()` and `count_words_in_file()` are proportional to the size of the files. The functions `merge_counts()` and `get_top10()` are proportional to the number of unique words in the book. These are the blocks.

The file reading `get_file()` and the word counting `count_words_in_file()` can easily be parallelized since they are independent calculations. Merging counts `merge_counts()` is not easily parallelizable since it updates the global variable `global_counts`. The function `get_top10()` is not easily parallelized since it is a sorting algorithm over a global variable. The filename generator `get_filenames(path)` is parallelizable since it is iterating through directories recursively. However, this computation is very small in comparison to the other, and we do not need to bother parallelize it. Furthermore, parallelization in each folder could lead to overhead delays if there are many directories. We can check the number of files by the following command.

```
find /data/courses/2025_dat470_dit066/gutenberg/huge -type f | wc -l
47121
```

Given the checksum for huge : 14880058115, the proportion of files to file content is about 315 000. Hence, the running time of the filename generator is $\frac{1}{350000}$ of the algorithms with running times proportional to the file contents.

d)

```
Time spent on reading filenames: 0.18 seconds
Time spent on reading files: 78.57 seconds
Time spent on counting words: 388.91 seconds
Time spent on merging counts: 253.03 seconds
Time spent on getting top 10 and checksum: 26.47 seconds
Total time: 747.16 seconds
```

We calculate the parallelizable fraction f by adding the time spent on reading files and counting words and divide it by the total running time:

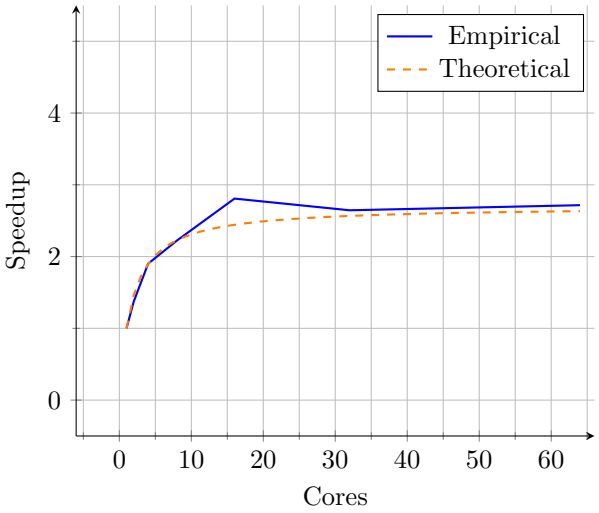
$$f = \frac{78.57 + 388.91}{747.16} = 0.62567589271 \approx 0.63$$

We calculate the upper bound on speedup by applying Amdahl's law on $p = 0.63$.

$$S(s) = \frac{1}{0.37 + \frac{0.63}{s}}$$

For $s \to \infty$ we get $S(s) = \frac{1}{0.37} \approx 2.7$.

e)

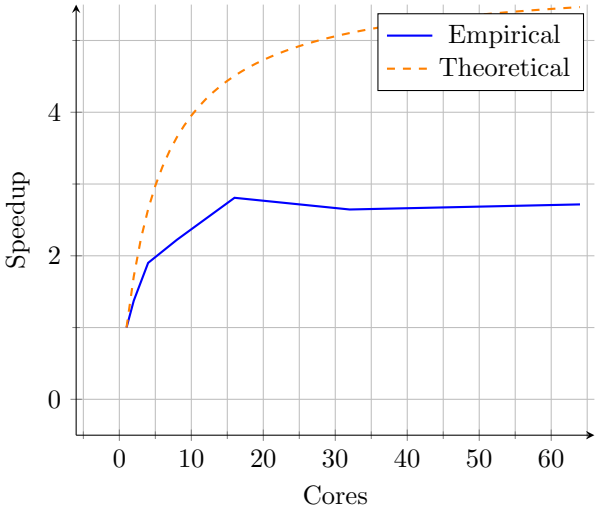



```
Total time with 64 workers: 421.96 seconds
```

As we can see the parallelized algorithm outperforms the theoretical upper bound which could be due to several factors. First, the theoretical upper bound is dependent on one run of the script which intrdocues some noice due to current performance of the node. Second, parallelization comes with overhead, and the total time with 1 worker in `assignment2_problem2e.py` is 1185 seconds, which is 438 seconds more than its counterpart in `assignment2_problem2d.py`. Furthermore, and most importantly, a larger fraction of the total time `assignment2_problem2e.py` is spent on counting words.

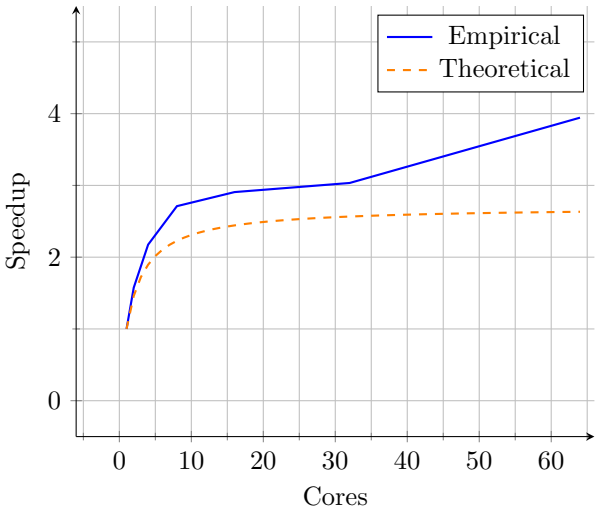
```
Running assignment2_problem2e.py with 1 worker(s)...
Time spent on reading files: 37.39 seconds
Time spent on counting words: 945.37 seconds
Time spent on merging counts: 178.10 seconds
Time spent on computing top 10 and checksum: 23.73 seconds
Total time with 1 workers: 1184.59 seconds
```

Calculating the parallelizable fraction for `assignment2_problem2e.py` gives us $f = 0.83$. Using this value for the theoretical upper bound gives us the following comparison.



Finally, SLURM would not allow us to run all the jobs on the same node. 1-16 cores were run on ganymede, 32 run on callisto, and 64 run on neptune. This could also affect the running time of these jobs.

f)

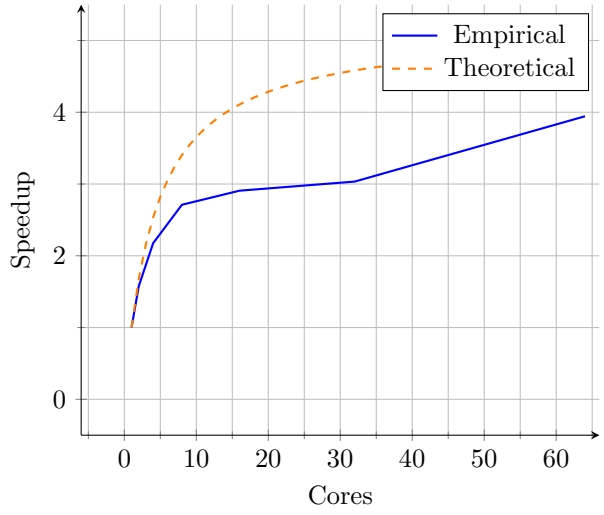


Total time with 64 workers: 278.03 seconds

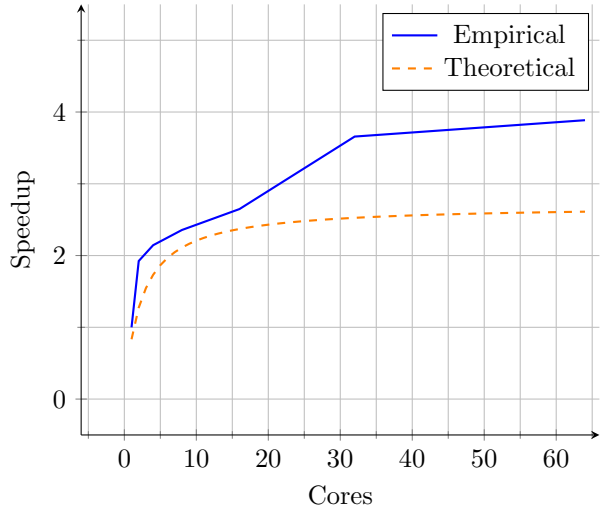
Again, the parallelization outperforms the theoretical upper bound. For 1 worker, we have the following data.

```
Time spent on counting words: 885.30 seconds
Time spent on merging counts: 187.64 seconds
Time spent on computing top 10 and checksum: 23.67 seconds
```

That means that the parallelizable fraction $f = 0.807$. We insert this into Amdahl's law and get the following plot.



g)



Total time with 64 processes: 211.21 seconds

We chose to use $2w$ as batch sizes. We don't want too large batches, because it could slow down the performance if some workers are slow (get very large files) leading to other workers being idle while waiting for it to finish. We also don't want too small batches since it will increase the number of I/O operations.

Note: the algorithm continues to outperform the theoretical bound (and by a larger margin). However, it is not easy to set a theoretical upper bound for $n = 1$ workers since the parallelizable and non-parallelizable operations are dependent of each other and not sequentially separable.