Wordcount - Semestral project

Filip Machulda*

1 Introduction

Wordcount is a simple command-line application, that allows users to quickly analyze their plain-text files in terms of word frequency and total word count.

2 How to use

If you are using the Linux environment, you can simply run the pre-compiled application ('wordcount'), which is included with the source code [1] in the cmake-build-release directory.

Once located, you can run the application in your terminal as *./wordcount [commands]*.

On other environments, you will have to compile the source code [1] yourself. The CMake file is included, though, it is expected of you to have Boost [2] already installed on your system.

Commands

Table 1: Commands

Command	Description
-? (–help)	Show all commands and their descriptions
-i (–input-file) [arg]	Path to the input file Mandatory
-t (–thread) [arg]	Select how many threads to use If not selected explicitly, one thread is used
-c (–count)	Prints a list of all (unique) words with the number of times they have appeared in the document
-u (–unique)	Prints the number of unique words found in the document
–total	Prints the total number of words found in the document

3 Library use

Boost

Boost is a set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, and unit testing. [4]

Program options

The program options library allowed me to get an easy access to the user-input commands and their values (in case of input-file and thread count).

Regex

I have used the Regex library to split each line into individual words (see word definition in the *Limitations* chapter), as it allows *easy* modification of the word definition.

CTPL

CTPL [3] allowed me to use even more abstraction in my code, as I didn't need to worry about creating and managing individual threads.

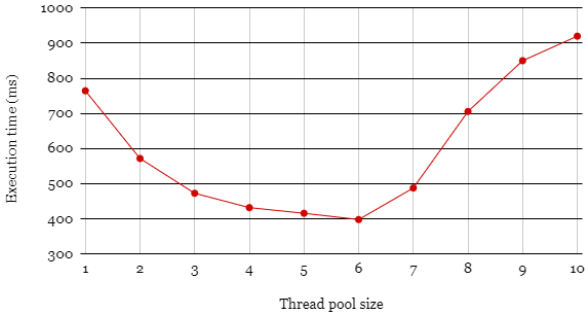
Instead, it allowed me to use a thread pool, that internally uses Boost's [2] thread-safe queue library, to which I pushed individual lines that I wanted to be parsed and later retrieved the results via std::future.

4 Multit-treaded performance

All of the tests were conducted on an AMD Ryzen 7 3700X CPU @ ~4.2GHz.

All results are a mean of 10 consecutive tests, performed over the same file[1], with the same exact settings[2].

Figure 1: Multi-threaded performance



[1] The file is also included in the cmake-build-release folder
[2] ./wordcount -i test_big.txt -u -t [count]

* machufi1@fel.cvut.cz

The drop in performance when using more than 6 threads is most likely caused due to the overhead of creating new threads and the fact that there are never more than 6 threads running concurrently.

5 Limitations

Word definition

In the context of this application, a word is defined as *a string of letters and numbers, with no other characters in between*.

I have defined it as such to keep the application simple, as there are many different ways to define a *word*, with many edge cases and complications, which I believe are out of the scope of this application.

This causes words such as 'eighty-five' or words split at the end of a line, to be counted as two individual words.

Multi-threading

The input text is not split into *absolutely* same-sized chunks, but is rather split by each line. This could potentially create uneven load on the threads and be optimized further, though, I believe that in a real-word application, the line length throughout a document is going to be roughly the same.

Splitting the text into chunks could also result in a longer execution time and higher memory consumption, as splitting the text would require you to load the whole file into the memory first (rather than efficiently using io streams) and then finding an adequate place to split the text (i.e. not in a middle of a word).

File support

Wordcount only supports plain-text files.

References

[1] Wordcount - Source code, Filip Machulda, 2020
`https://github.com/file59/Wordcount`

[2] Boost - C++ Libraries
`https://www.boost.org/`

[3] CTPL - Modern and efficient C++ Thread Pool Library, vit-vit
`https://github.com/vit-vit/CTPL`

[4] Wikipedia - Boost (C++ libraries)
`https://en.wikipedia.org/wiki/Boost_(C++_libraries)`