Parallel Computing

Bigrams and Trigrams Histogram
An Open Mp implementation

Giacomo Magistrato

Introduction

- **Objective**: The aim of this project is to outline the differences, in terms of performances, between a Sequential and a Parallel implementation of an algorithm which create histogram of Bigrams/Trigrams of words and character of a given text dataset.
- The Parallel implementation makes use of **Open Mp** in order to implement parallelism. This can potentially lead to a much faster execution compered to the sequential implementation of the instructions.
- In order to evaluate performance, both implementation uses the same dataset.
- The dataset used for testing the project, was created mixing different books provided by Gutemberg Project.
- The source code of this project can be retrieved at the following location:
- https://github.com/Mayo98/BigramsTrigrams_Parallel

Algorithm Overview

Initialization

- The algorithm starts by calling the function *clean_txt()* which reads the source txt file and provides a new txt file with the same content of the original one but without punctuation and special characters.

- This new source file will be used as input of the algorithm.

- Setting the parameter *n*, we define the type of N-Grams we want to count (e.g. n = 2 for Bigrams).

Algorithm Overview

N-Grams of Words

- The algorithm uses istringstream to read the text file and than saves all words in a string vector.
- Than the vector is iterated and the words are saved in a variable(currentNgram). Once currentNgram reach the lenght n, is added to the histogram and resetted for the next n-gram.
- The histogram is implemented using an unordered_map. This uses words n-grams as the Keys and a counter as Value for each key.

Algorithm Overview

N-grams of Character

- As for the n-grams of words, all the words of the file are stored in a string vector.
- This time the algorithm iterate words and each characters belonging to the word.
- Characters are temporary saved in a variable (currentN-gram) and added to the histogram once reached the correct length.
- The histogram is implemented as for n-grams of words implementation. This time Keys are characters n-grams.

Algorithm Overview

Printing phase

- This part belong to both Characters and Words N-grams implementation.
- Once the iteration of the words is finished, *printHistogram()* function is called.
- Each element of the unordered_map is pushed into a priority_queue which will automatically order all the n-grams in descending order by Value(the counter).
- Then using pop() function, most common n-grams are popped out and printed.
- *Note*: the unordered_map (the histogram) store each n-gram as Key and the counter for each n-gram as Value

UNIVERSITÀ DEGLI STUDI FIRENZE

Parallel Version

- The parallel version of the algorithm uses **Open Mp** to implement parallelism.
- This is done dividing the vector of words into equal parts and assigning each part to a specific thread.
- Each thread will iterate the part of the vector belonging to it. It will count all the n-grams and update a localHistogram.
- Once it finished to iterate, the thread update the globalHistogram with the values of the localHistogram that it created.
- This operation is done by each thread using a *omp critical* directive in order to avoid concurrency on globalHistogram.
- Both Words and Characters N-grams counters follows this strategy to implement parallelism.

Testing Machine

Specification

- Macbook Air 2017, MacOS Monterey
- CPU: 1,8 GHz Intel Core i5 dual-core
- Memory: 8 GB 1600 MHz DDR3

Testing

The testing is based on two strategies:

- Testing the execution time using different sizes of the data set.
- Testing the execution time using different numbers of thread for the parallel version.

In order to evaluate the performance and compare the execution times of the sequential version(T_{seq}) and the parallel one(T_{par}), is used the Speedup function:

$$SpeedUp = \frac{Tseq}{Tpar}$$

What do I expect

Changing size of dataset

- Speedup is expected to be lower to 1 when the text file size is small (e.g. <= 5MB) because of the overhead introduced by the parallelization for the allocation and synchronization of the threads, that makes sequential version faster. While speedup could be grater than 1 as soon as text file size grows.

Changing numbers of threads

- Speedup is expected to be grater proportionally to the number of threads and the text file size.

-

Results (1)

In the table below are shown the speedups for the executions of the algorithm. Rows of the tables displays the .txt file size while columns displays the number of threads used.

Bigrams of Words

	2	3	4
1.3MB	0.78	0.84	0.81
19MB	1.45	1.43	1.52
50MB	1.68	1.74	1.77
128MB	1.79	2.08	2.11
450MB	1.87	2.15	2.20

Trigrams of Words

	2	3	4
1.3MB	0.79	0.75	0.66
19MB	1.22	1.30	1.40
50MB	1.60	1.66	1.70
128MB	1.75	1.95	2.10
450MB	1.86	2.13	2.47

Average execution time for trigrams of words

Size \ N threads	Sequential	2	3	4
128 MB	27.20s	15.54s	13.94s	12.95s
450 MB	96.68s	51.89s	45.34s	39.15s

UNIVERSITÀ DEGLI STUDI FIRENZE

Results (2)

In the table below are shown the speedups for the executions of the algorithm. Rows of the tables displays the .txt file size while columns displays the number of threads used.

Bigrams of Characters

	2	3	4
1.3MB	1.60	1.72	1.77
19MB	1.82	1.97	2.10
50MB	1.87	2.00	2.04
128MB	1.89	2.04	2.15
450MB	1.90	1.98	2.27

Trigrams of Characters

	2	3	4
1.3MB	1.50	1.55	1.61
19MB	1.86	1.96	2.10
50MB	1.88	2.08	2.25
128MB	1.90	2.06	2.13
450MB	1.93	2.10	2.21

Average execution time for bigrams of characters

Size \ N threads	Sequential	2	3	4
128 MB	45.97s	24.80s	22.50s	21.11s
450 MB	169,45s	89.20s	85,27s	74,65s

Performance Evaluation

Conclusions

- Execution times for the analysis of character n-grams are much higher than for word ngrams. This is because the algorithm has to analyze many more elements. At the same times speedups grow up.

- As I expected the speedups for small files are below 1, as the size of the files increases they tend to grow. This means that for small files size, parallelism doesn't bring benefits.

- The worst speedups (0.66) occurs in the analysis of words trigrams in the case where the file size is the smallest and it is consistent because it is the case in which there are fewer operations to perform and the overhead of the parallelism is relevant.