Parallel Computing: N-gram Analysis with OpenMP

Andrea Bigagli - andrea.bigagli1@edu.unifi.it

February 2026

Abstract

The computation of n-grams is a common operation in text analysis and natural language processing, but it can become computationally expensive when applied to large textual datasets. This project investigates the parallelization of n-gram counting using OpenMP on a shared-memory system, focusing on both word-based and character-based bigrams and trigrams.

A sequential baseline and a parallel implementation based on thread-level data parallelism were developed and experimentally evaluated. To avoid synchronization overhead during the counting phase, the parallel solution adopts thread-local data structures followed by a final merge step.

Experimental results obtained on datasets of different sizes show that character-based n-grams achieve significant speedup, while word-based n-grams benefit less from parallelization due to higher overhead and a larger serial fraction. The analysis confirms that the effectiveness of parallelization strongly depends on workload characteristics and input size, in accordance with the theoretical principles of parallel computing.

1 Introduction

The analysis of large textual datasets is a common task in many application domains, including natural language processing, information retrieval, and data mining. One of the most widely used techniques in text analysis is the computation of *n-grams*, which are contiguous sequences of n elements extracted from a text, such as words or characters.

When dealing with large corpora, the computation of n-grams becomes computationally expensive due to the large amount of data and the high number of distinct sequences that must be processed and stored. This makes the problem a suitable candidate for parallelization, as the workload can potentially be divided among multiple processing units.

The goal of this Mid Term project for the *Parallel Computing* course is to study the effectiveness of parallel programming techniques in the computation of n-grams. In particular, the project focuses on the calculation of bigrams and trigrams, both at the word level and at the character level, and compares a sequential implementation with a parallel implementation based on OpenMP.

The performance of the two versions is evaluated through experimental measurements of execution time and speedup, using datasets of different sizes. The objective is not only to assess the performance improvements achieved through parallelization, but also to analyze the impact of overhead, scalability, and data structure management on the overall efficiency of the parallel solution.

2 Problem Description

The problem addressed in this project consists in the computation of *n-grams* from large textual datasets. An n-gram is defined as a contiguous sequence of n elements extracted from a text, where the elements can be either words or characters.

In this work, two values of n are considered: $n = 2$ (bigrams) and $n = 3$ (trigrams). The analysis is performed both at the word level and at the character level, leading to four distinct categories of n-grams.

Word-based n-grams are obtained by first tokenizing the text into individual words and then considering consecutive sequences of words. A word bigram is

therefore defined as a pair of consecutive words, while a word trigram consists of three consecutive words.

Character-based n-grams are computed within each individual word. In this case, a character bigram corresponds to a pair of consecutive characters, while a character trigram corresponds to a sequence of three consecutive characters extracted from the same word.

The goal of the computation is to count the frequency of each distinct n-gram appearing in the dataset. Given the potentially large size of the input text and the high number of distinct n-grams, especially in the word-based case, this problem presents significant computational and memory challenges.

3 Dataset

The experiments were conducted using textual datasets in English obtained from the Leipzig Corpora Collection. In particular, news articles were selected in order to work with realistic and heterogeneous natural language data.

Two datasets of different sizes were used:

- a medium-sized dataset containing approximately 100 000 sentences;
- a larger dataset containing approximately 1 000 000 sentences.

Each line of the input files corresponds to a single sentence. The datasets are not included directly in the repository due to their size, but they are loaded at runtime from a dedicated `dataset/` directory.

The use of two datasets with significantly different sizes allows the evaluation of the scalability of the implementations. In particular, it makes it possible to observe how the performance and the achieved speedup change when increasing the amount of input data, highlighting the impact of parallelization overhead on smaller inputs and its benefits on larger ones.

4 Text Preprocessing

Before computing the n-grams, the input text is processed through a preprocessing phase that is common to both the sequential and the parallel implementations. The goal of this phase is to normalize the input and ensure consistent tokenization, while keeping the preprocessing cost limited with respect to the overall computation.

The main preprocessing steps are the following:

- conversion of all characters to lowercase;
- removal of non-ASCII characters;
- handling of punctuation symbols;
- removal of apostrophes;
- preservation of internal hyphens within words (e.g., `t-shirt` is treated as a single token);
- removal of leading numeric identifiers from each line;
- splitting of each line into word tokens.

This preprocessing phase directly affects the number of generated tokens and, consequently, the number of n-grams that are produced and counted. For this reason, the same preprocessing logic is applied consistently in both implementations, ensuring a fair comparison between the sequential and parallel versions.

5 Sequential Implementation

The sequential implementation represents the baseline version of the project and is used as a reference for the performance evaluation of the parallel solution. Its purpose is to compute the frequency of the different n-grams appearing in the input text without exploiting any form of parallelism.

The program processes the dataset line by line. For each line, the same preprocessing steps described in the previous section are applied, and the resulting tokens are used to generate word-based and character-based n-grams. Each n-gram is then counted using a hash-based associative container.

Algorithm 1 Sequential N-gram Counting

Require: Text dataset D, n-gram size n
Ensure: Global maps G_w, G_c
Initialize $G_w, G_c \leftarrow \emptyset$
for all line L in D **do**
    $L \leftarrow preprocess(L)$
    $T \leftarrow tokenize(L)$
    Update word n-grams in G_w from T
    Update char n-grams in G_c scanning each token in T
end for
return G_w, G_c

The pseudocode outlines the core logic of the sequential approach. The same structure is reused in the parallel version, where the outer loop over the input data is distributed among multiple threads.

6 Parallel Implementation

The parallel implementation is based on OpenMP and targets shared-memory architectures. The goal is to exploit thread-level parallelism to reduce the execution time of the n-gram counting phase, while preserving correctness and avoiding data races.

The overall structure of the program closely follows the sequential version. The main difference lies in the distribution of the workload among multiple threads and in the management of the data structures used to store the n-gram frequencies.

6.1 Parallelization Strategy

The parallelization is applied to the main loop that iterates over the input dataset. Each iteration processes an independent portion of the text, making this loop a suitable candidate for data parallelism.

An OpenMP `parallel for` directive is used to distribute the iterations of this loop among the available threads. This approach allows multiple lines of the dataset to be processed concurrently, while preserving the same logical structure of the sequential algorithm.

Algorithm 2 Parallel N-gram Counting with Thread-Local Maps (OpenMP)

Require: Text dataset D, n-gram size n
Ensure: Global maps G_w, G_c
$p \leftarrow$ `omp_get_max_threads()`
Allocate local maps $\{L_w^{(k)}, L_c^{(k)}\}_{k=0}^{p-1}$
Initialize $G_w, G_c \leftarrow \emptyset$

Parallel region
$k \leftarrow$ `omp_get_thread_num()`
for all indices i assigned to thread k (`#pragma omp for`) **do**
    Update word n-grams in $L_w^{(k)}$
    Update char n-grams in $L_c^{(k)}$
end for
End parallel

for $k \leftarrow 0$ **to** $p-1$ **do**
    Merge $L_w^{(k)}$ into G_w
    Merge $L_c^{(k)}$ into G_c
end for
return G_w, G_c

6.2 Thread-Local Data Structures

A critical aspect of the parallel implementation is the management of the data structures used to count the n-grams. Using a single shared map would lead to frequent write conflicts and require synchronization mechanisms such as locks or atomic operations, significantly reducing performance.

To avoid this issue, each thread maintains its own local frequency map. During the parallel region, threads update only their private maps, completely eliminating data races during the counting phase.

6.3 Merge Phase

At the end of the parallel counting phase, the local maps produced by each thread are merged into a single global map. This merge operation is performed sequentially and combines the partial results by summing the counts of identical n-grams.

While this strategy effectively avoids synchronization during the parallel phase, it introduces a serial section in the program. As a result, the merge phase represents a source of overhead that limits the achievable speedup, in accordance with Amdahl's law.

7 Experimental Methodology

This section describes the methodology adopted to evaluate the performance of the sequential and parallel implementations. The goal of the experimental analysis is to measure execution times, compute speedup values, and assess the scalability of the parallel solution under different input sizes. For each dataset size, three independent experimental runs were performed. In each run, execution times were measured separately for each n-gram category, including word-based and character-based bigrams and trigrams. The execution times reported in the following sections correspond to the average of the three runs. Speedup values were computed using the averaged sequential and parallel execution times, rather than averaging individual speedup measurements.

7.1 Execution Setup

All experiments were performed on the same machine to ensure consistency of the results. The sequential and parallel versions were compiled with the same optimization settings, differing only in the use of OpenMP directives.

The number of OpenMP threads was set to the maximum value returned by the OpenMP runtime through `omp_get_max_threads()`, which corresponds to 16 logical processors on the target system. All parallel experiments were therefore executed using 16 threads.

Table 1: System specifications used for the experiments

Parameter	Specification
Operating System	Windows 11
Machine	MSI Cyborg 15 A13V
CPU	Intel Core i7-13620H
Physical cores	10
Logical processors	16
Memory	16 GB RAM
GPU	NVIDIA RTX 4050 (not used)

Each experimental configuration was executed three times to reduce variability due to system noise and runtime effects. For each dataset and n-gram category, the reported execution times correspond to the average of the three runs.

Speedup values were computed using the average sequential and parallel execution times, rather than averaging individual speedup measurements.

8 Results

This section presents the experimental results obtained from the execution of the sequential and parallel implementations. Execution times and speedup values are reported for the different n-gram categories and dataset sizes described in the previous sections. The numerical values discussed in this section refer to the average execution times and speedup computed over three independent runs, as described in Section 7.

8.1 Most Frequent N-grams

In addition to execution times and speedup measurements, the frequency distribution of the extracted n-grams was analyzed. This analysis provides insight into the characteristics of the input data and helps explain some of the observed performance behaviors.

For readability, the following tables report the most frequent n-grams (Top-10 for word bigrams and Top-5 for the other categories).

Table 2: Most frequent word-based n-grams (100K vs 1M datasets)

Word Bigrams			Word Trigrams		
Bigram	**100K**	**1M**	**Trigram**	**100K**	**1M**
of the	11 817	117 187	one of the	900	8 761
in the	10 675	107 109	shares of the	827	8 646
to the	5 489	54 194	a lot of	533	5 482
on the	4 317	43 424	company s stock	514	5 578
for the	4 072	41 731	as well as	513	4 930

Table 3: Most frequent character-based n-grams (100K vs 1M datasets)

Character Bigrams			Character Trigrams		
Bigram	**100K**	**1M**	**Trigram**	**100K**	**1M**
th	233 059	2 337 122	the	152 327	1 524 241
he	198 909	1 994 505	ing	76 723	770 232
in	193 688	1 941 998	and	62 381	627 919
er	145 274	1 462 242	ion	40 078	398 587
an	141 592	1 422 365	ent	37 345	373 992

Across all n-gram categories, the ranking of the most frequent n-grams remains stable when moving from the 100K to the 1M dataset, while absolute frequencies scale proportionally with the input size. This behavior is consistent with the typical skewed distribution observed in textual data.

9 Performance Analysis

This section analyzes the performance trends observed in the experimental results. The analysis focuses on the impact of n-gram type and dataset size on the effectiveness of the OpenMP-based parallelization, highlighting the role of overhead, workload granularity, and serial sections of the code.

9.1 Word vs Character N-grams

A clear performance difference emerges between word-based and character-based n-grams. Character n-grams consistently achieve higher speedup values compared to word n-grams for both dataset sizes.

This behavior can be explained by the lower computational and memory overhead associated with character n-grams. They involve shorter keys, a smaller number of distinct n-grams, and more uniform workloads, which are well suited for data parallelism.

9.2 Impact of Dataset Size

The size of the input dataset has a significant impact on the effectiveness of parallel execution. For the smaller dataset (100K sentences), the benefits of parallelization are limited for word-based n-grams, and in the case of word trigrams the parallel version is slightly slower than the sequential one.

When the dataset size increases to 1M sentences, the overhead associated with thread management and the merge phase is amortized over a larger amount of work. As a result, all n-gram categories benefit from parallelization, and speedup values increase consistently.

9.3 Serial Fraction and Amdahl's Law

Despite the use of thread-level parallelism during the n-gram counting phase, the overall speedup is limited by the presence of serial sections in the program. In particular, the merge phase, which combines the thread-local maps into a global result, is executed sequentially.

According to Amdahl's law, the maximum achievable speedup is bounded by the fraction of the program that cannot be parallelized. This explains why speedup values tend to plateau and why increasing the number of threads does not lead to linear scaling.

Table 4: Average execution times for the 100K dataset (3 runs)

N-gram type	**Sequential (s)**	**Parallel (s)**
Word bigrams	0.67	0.56
Word trigrams	0.96	1.04
Character bigrams	0.09	0.03
Character trigrams	0.13	0.04

Table 5: Average execution times for the 1M dataset (3 runs)

N-gram type	**Sequential (s)**	**Parallel (s)**
Word bigrams	6.05	3.92
Word trigrams	9.71	9.04
Character bigrams	0.96	0.21
Character trigrams	1.48	0.28

Table 6: Average **speedup** comparison for the two dataset sizes

N-gram type	100K	1M
Word bigrams	1.21	1.54
Word trigrams	0.93	1.07
Character bigrams	2.71	4.48
Character trigrams	3.71	5.20

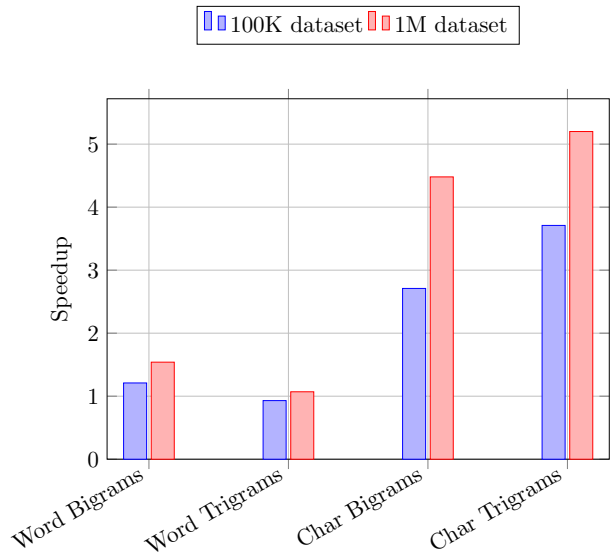

Figure 1: Average speedup comparison for different n-gram types and dataset sizes

10 Conclusions

This project investigated the parallel computation of word-based and character-based n-grams using OpenMP on a shared-memory system. A sequential baseline and a parallel implementation were developed and compared through experimental measurements on datasets of different sizes.

The results show that parallelization is highly effective for character n-grams, which achieve significant speedup thanks to their regular structure and lower overhead. Word-based n-grams, on the other hand, benefit less from parallel execution, especially on smaller datasets, due to the higher cost of string handling and the impact of the merge phase.

The experimental analysis confirms that the effectiveness of parallelization strongly depends on both the nature of the workload and the size of the input. When the dataset is sufficiently large, the overhead introduced by thread management and serial sections is amortized, leading to improved performance.

Overall, the results are consistent with the theoretical principles of parallel computing and highlight the importance of carefully selecting parallelization strategies and data structures to achieve meaningful speedup.