UNIVERSITÀ DEGLI STUDI FIRENZE

Parallel N-Grams Analysis with OpenMP

Parallel Computing

Andrea Bigagli 7036221

A.A. 2025/2026

Introduction

- N-grams are a fundamental building block in text analysis and **NLP**
- Used in applications such as language modeling, search, and text mining
- Computing n-grams on large datasets is computationally expensive
- Parallel computing can help reduce execution time

UNIVERSITÀ DEGLI STUDI FIRENZE

Problem Definition

The problem addressed in this project consists in computing the frequency of **n-grams** extracted from a large textual corpus.
An n-gram is defined as a contiguous sequence of ***n*** elements extracted from text, where the elements can be either **words** or **characters**.
Given a text corpus ***T***, the objective is to generate and count all the n-grams appearing in the dataset for fixed values of ***n***.

$$T = \{t_1, t_2, \ldots, t_N\}$$

$$g_i = \langle t_i, t_{i+1}, \ldots, t_{i+n-1} \rangle$$

where ***T*** represents the input text and g_i is an n-gram of size **n**.

Project Goal

The goal of this project is to compare a sequential and a parallel implementation of **n-gram counting using OpenMP**.

In this project, the analysis is limited to **bigrams** and **trigrams,** both **word-based** and **character-based** n-grams are considered.

The objective is to evaluate the impact of parallelization on execution time and speedup when processing large textual datasets.

UNIVERSITÀ DEGLI STUDI FIRENZE

Dataset

Textual datasets in English were selected from the **Leipzig Corpora Collection**,

Two different dataset sizes were considered in order to study scalability:

- **100K** sentences
- **1M** sentences

This choice allows the evaluation of parallel overhead on small inputs and scalability on larger datasets.

Text Preprocessing

Before computing **n-grams**, the input text is normalized in order to ensure consistent tokenization and fair performance comparisons.

The preprocessing pipeline includes:

- Conversion to **lowercase**
- Removal of **non-ASCII characters**
- Punctuation **handling and apostrophe removal**
- Preservation of internal hyphens (e.g., ***t-shirt***)
- **Tokenization** into words

The same preprocessing steps are applied to both the sequential and parallel implementations.

Sequential Implementation

The sequential implementation represents the baseline version of the project.

The input dataset is processed **line by line**. **For each line, the text is preprocessed and tokenized**.

Word-based and character-based n-grams are generated and counted using **hash-based frequency maps**.

This implementation is used as a reference to evaluate the performance of the parallel solution.

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**
  $L \leftarrow preprocess(L)$
  $T \leftarrow tokenize(L)$
  Compute and update word n-grams in G_w from tokens T
  Compute and update character n-grams in G_c by scanning characters inside each token in T
end for
return G_w, G_c

UNIVERSITÀ DEGLI STUDI FIRENZE

Parallel Implementation - Overview

The parallel implementation is based on **OpenMP** and targets **shared-memory architectures**.

The overall logic follows the sequential version, but the workload is distributed among **multiple threads**.

Each thread processes an independent portion of the input data, enabling thread-level data parallelism.

Parallelization key ideas:

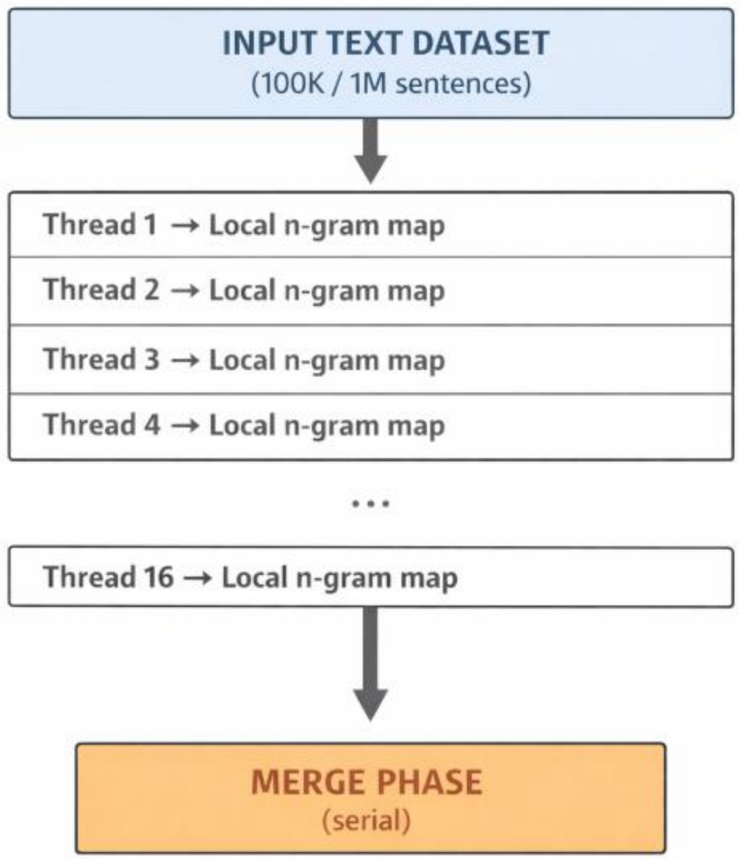
- Same algorithmic structure
- Independent data chunks
- Thread-level parallelism

Parallelization Strategy

The parallelization is applied to the outer loop that processes the input dataset.

Each thread works on an independent sub of the data and maintains its own local n-gram maps.

This design avoids synchronization during counting phase and eliminates data races.

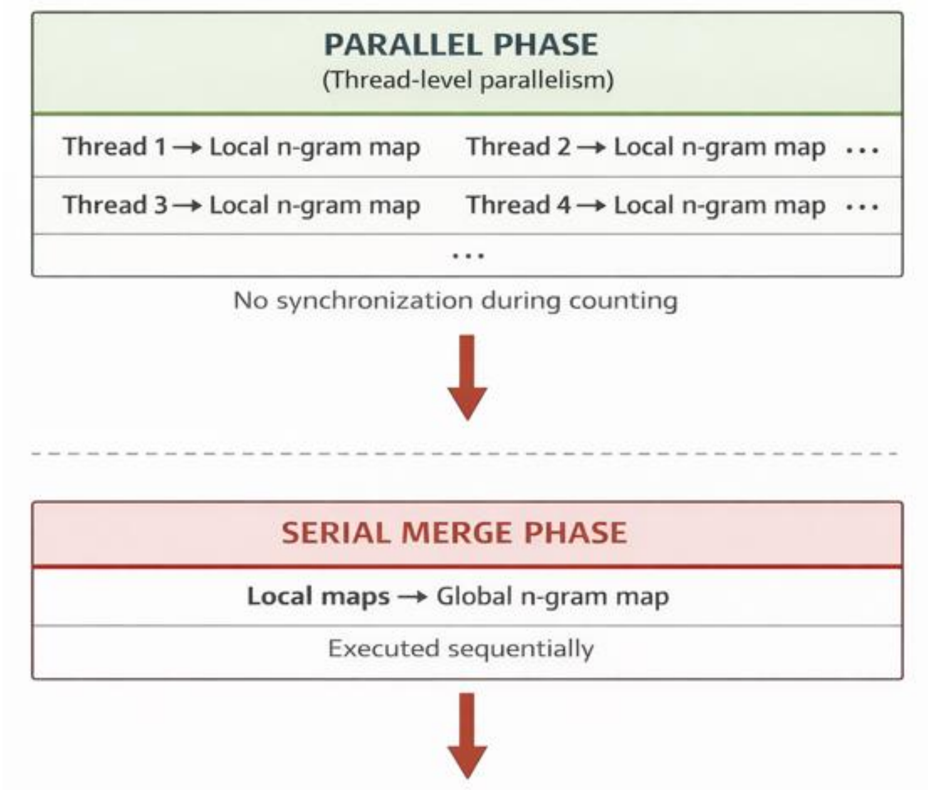


Merge Phase

After the parallel counting phase, the local n-gram maps produced by **each thread are merged into a single global map**.

The merge operation is executed sequentially and represents the only serial section of the timed counting workflow.
The merge is performed once for each n-gram category (4 merges in total).

This phase introduces overhead and limits the maximum achievable speedup, according to **Amdahl's Law**.



Most Frequent N-grams (100K vs 1M datasets)

The ranking of the most frequent n-grams remains stable across dataset sizes.

Word Bigrams – Top 5

Bigram	100K dataset	1M dataset
of the	11,817	117,187
in the	10,675	107,109
to the	5,489	54,194
on the	4,317	43,424
for the	4,072	41,731

Word Trigrams – Top 5

Trigram	100K dataset	1M dataset
one of the	900	8,761
shares of the	827	8,646
a lot of	533	5,482
company s stock	514	5,578
as well as	513	4,930

Character Bigrams – Top 5

Bigram	100K dataset	1M dataset
th	233,059	2,337,122
he	198,909	1,994,505
in	193,688	1,941,998
er	145,274	1,462,242
an	141,592	1,422,365

Character Trigrams – Top 5

Trigram	100K dataset	1M dataset
the	152,327	1,524,241
ing	76,723	770,232
and	62,381	627,919
ion	40,078	399,587
ent	37,345	373,992

Execution Times (Averaged Results)

Average Execution Times — 100K Dataset (3 runs)

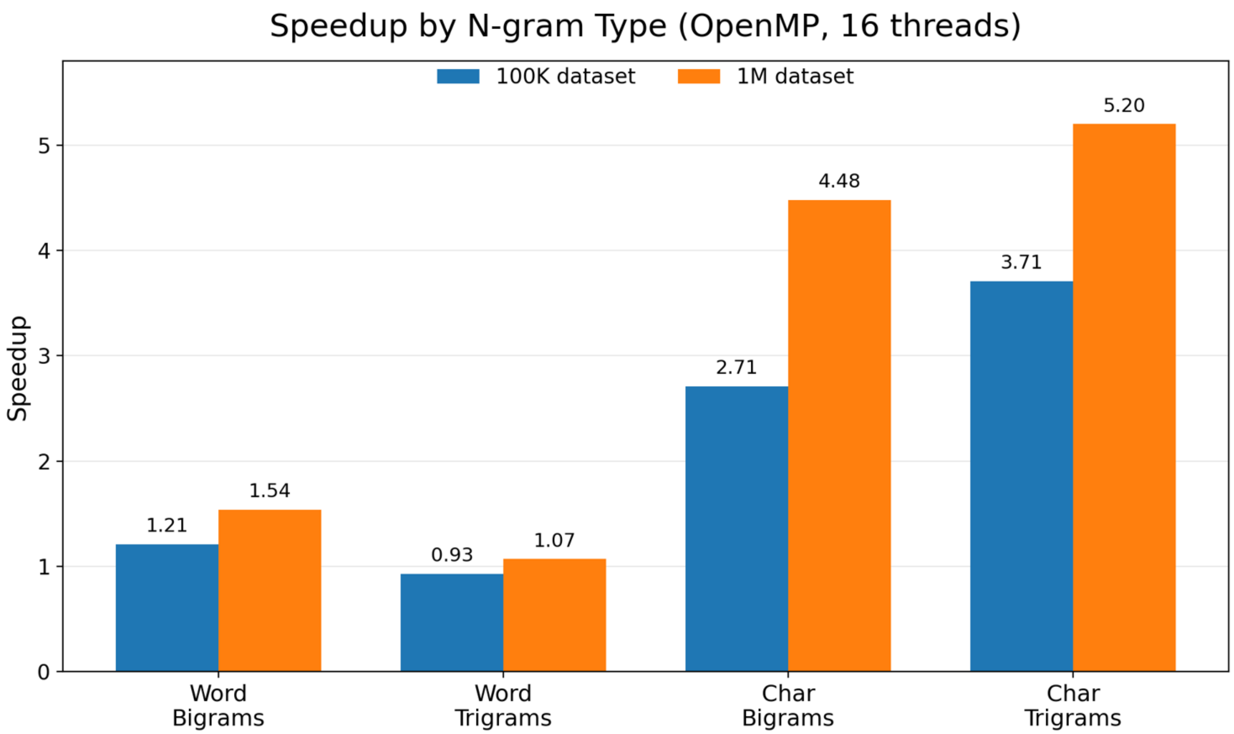
N-gram type	Sequential (s)	Parallel (s)
Word bigrams	0.67	0.56
Word trigrams	0.96	1.04
Character bigrams	0.09	0.03
Character trigrams	0.13	0.04

Average Execution Times — 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

- 3 independent runs were executed for each dataset
- Execution times were measured for each n-gram category
- Reported values are averages over the 3 runs
- Speedup is computed from averaged times

Results: Speed Up



- Character n-grams achieve significantly higher speedup
- Speedup increases when moving from 100K to 1M sentences
- Word trigrams show limited improvement due to overhead

Performance Analysis

The experimental results show a **clear difference between word-based and character-based n-grams.**

Character n-grams achieve higher speedup due to shorter keys, a smaller number of distinct n-grams, and more regular workloads.

Word trigrams generate many unique keys and require more memory operations during the merge phase, which increases overhead and limits scalability.

Increasing the dataset size **from 100K to 1M sentences amortizes parallel overhead, leading to higher speedup values**.

Key observations:

- **Character n-grams scale better**
- **Word trigrams are merge-bound**
- **Merge phase is the main serial bottleneck**
- **Speedup follows Amdahl's Law**

Conclusions

- **Parallelization Strategy:**
  OpenMP parallelization is effective when applied to independent data chunks, as in n-gram counting over large text corpora.

- **Data Characteristics:**
  Character-based n-grams scale better due to shorter keys and fewer distinct elements, while word-based n-grams suffer from higher memory and merge overhead.

- **Scalability Limits:**
  The merge phase represents the main serial bottleneck and limits speedup, consistently with Amdahl's Law.