UNIVERSITÀ DEGLI STUDI FIRENZE

Ngrams with Cuda

Parallel Computing

Alessandro Bianco 7147240

A.A. 2025/2026

Introduction

N-gram counting is a fundamental operation in NLP, essential for Language Modeling, Machine Translation, and Sentiment Analysis.

The problem: Processing billions of N-grams in modern text corpus creates a massive computational bottleneck for traditional CPUs.

The identified solution: Modern GPUs offer a massively parallel architecture ideal for this workload.

The challenge: Harnessing this parallelism is non-trivial. It introduces a critical trade-off between *atomic memory contention* and *exponential memory complexity*.

Proposed approach

In this report, we implement and analyze three GPU algorithms that embody this conflict:

- **V1** (Global Atomic Histogram)
- **V2** (Private Histograms)
- **B** (Map-Sort-Reduce).

The final objective is to demonstrate that **no single solution prevails**, but rather an optimal hybrid strategy that selects the most suitable algorithm for each specific scenario.

Metric	Histogram(V1 /V2)	Map-Sort-Reduce (B)
Memory	$O(C^n)$	$O(N)$
Time	$O(N)$	$O(N \log N)$

Core methodology: Flat indexing

All algorithms map each N-gram to a unique integer index, so this implies that each thread uses a sliding window approach: 1 thread = 1 N-gram.

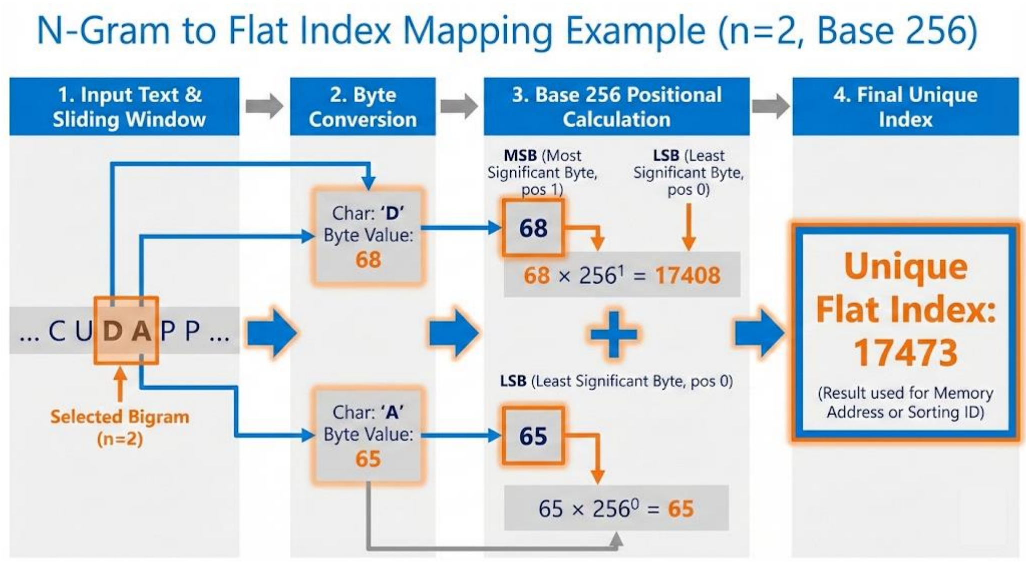
The text is treated as an array of bytes and so we interpret the N-gram as a number in base 256:

- **MSB**: the leftmost character multiplied by 256^{n-1}
- **LSB**: the rightmost character multiplied by 256^{0}

$$idx = \sum_{i=0}^{n-1} char[idx + i] \times 256^{n-1-i}$$

Core methodology: Flat indexing

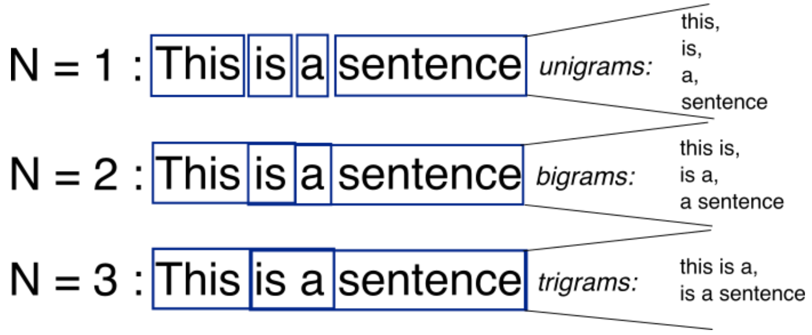
So in the V1/V2 histogram algorithms the index represents the memory address to atomically increment, while in the B algorithm (Map-Sort-Reduce) it represents the numeric ID that is being written inside the array.



Sequential Baseline

The standard workflow for counting the n-grams is the following:

- Iterate linearly through the corpus byte array
- Extract the N-gram at each position using a sliding window
- Decode bytes to string and update the frequency count in the dictionary



UNIVERSITÀ DEGLI STUDI FIRENZE

V1 Algorithm

V1 algorithm uses a direct parallelization strategy mapping 1 thread to 1 N-gram. The structure is a single shared global histogram in VRAM.

We allocate a contiguous memory block of size 256^n and calculate the flat index for the specific N-gram performing an *atomicAdd* on the global counter.

The main bottleneck is the memory constraint given by histogram array allocation which has an exponential complexity:

- n=2: size ≈ **256kb**
- n=4: size ≈ **16gb**

UNIVERSITÀ DEGLI STUDI FIRENZE

V2 Algorithm

In V2 we want to mitigate the traffic on single memory addresses so we replace the global histogram with K separate private histograms.

There is a two stage implementation:

- Private population: threads compute the flat index and instead of writing globally, they select a specific private histogram based on their block Id

- Global Reduction: A synchronization barrier separates the kernels and a second kernel iterates through the k private histograms summing the partial counts into the global result

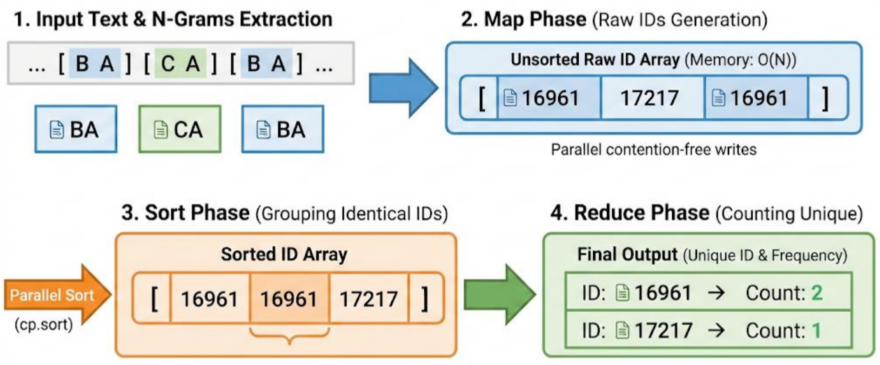
We eliminate the global atomic contention but there is now a multiplied memory factor by K.

This results in faster results for small n but causes OOM errors for greater values

B Algorithm

We use a Map-Sort-Reduce strategy where now the memory footprint is ruled by the number of N-grams O(N).

- Phase 1 **Map**: each thread computes the flat index not using the *atomicAdd* but using a contention-free write of the ID into the array
- Phase 2 **Sort**: sorts group identical N-grams IDs contiguously
- Phase 3 **Reduce**: parallel reduction that counts the lenghts of contiguous segments to determine the frequency



Test suite structure

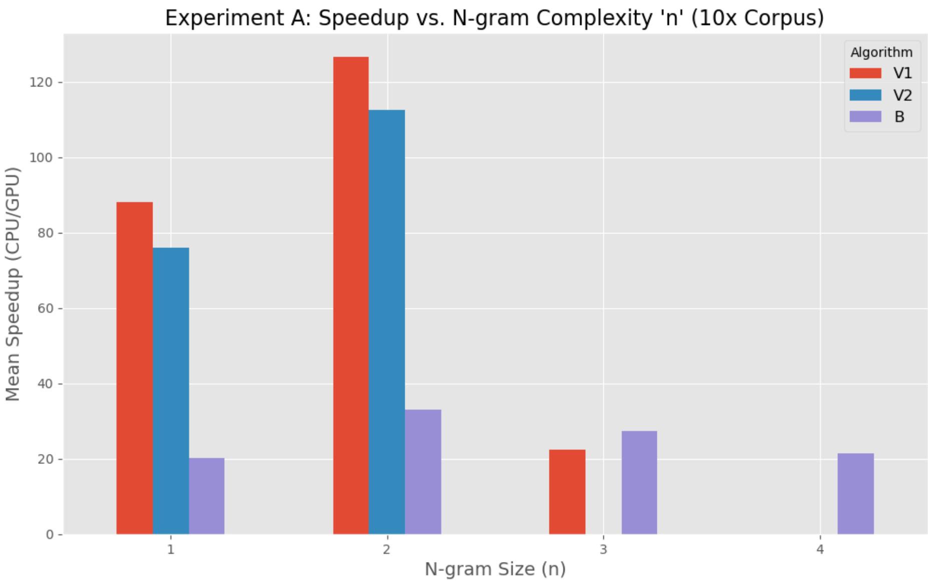
Each algorithm receives a text input (corpus) that is duplicated in memory by an amplification factor to simulate different workload sizes and its also converted to an array of bytes to ensure a 1:1 mapping between characters and 8-bit values.

Two core experiments:

- **Experiment A**: the goal is to test the impact of exponential memory requirements varying the N-gram size while keeping the corpus size fixed
- **Experiment B**: tests the performance by varying the corpus size while keeping n fixed to test the throughput scalabilty

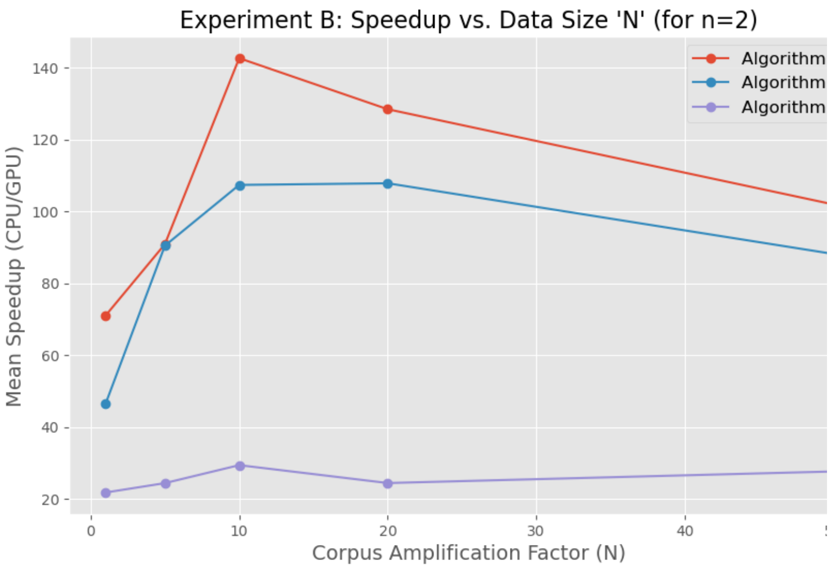
Experiment A: Scalability vs Complexity (n)

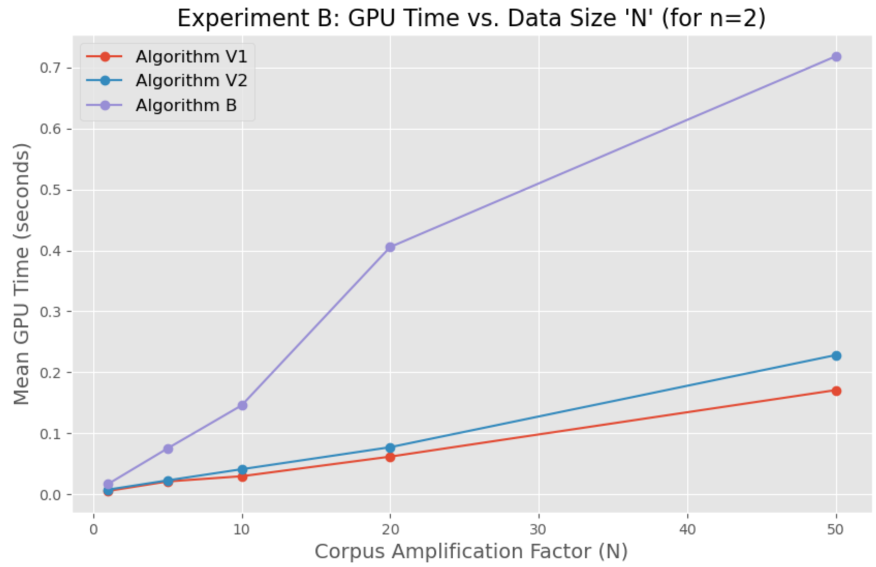
Alg	N	Hists	CPU Time (s)	GPU Time (s)	Speedup
V1	1	-	2.686±0.015	0.031±0.004	88.0±10.8x
V1	2	-	3.727±0.550	0.029±0.000	126.7±20.1x
V1	3	-	3.676±0.081	0.174±0.050	22.5±7.1x
V1	4	-	4.003±0.519	∞ (FAIL)	0.0±0.0x
V2	1	128	2.818±0.007	0.073±0.081	75.9±51.1x
V2	2	128	4.555±0.622	0.040±0.000	112.5±15.6x
V2	3	128	3.516±0.020	∞ (FAIL)	0.0±0.0x
V2	4	128	3.692±0.022	∞ (FAIL)	0.0±0.0x
B	1	-	3.095±0.478	0.155±0.017	20.2±4.4x
B	2	-	4.783±0.533	0.144±0.001	33.1±3.7x
B	3	-	4.259±0.531	0.156±0.001	27.3±3.5x
B	4	-	4.362±0.540	0.203±0.000	21.5±2.7x



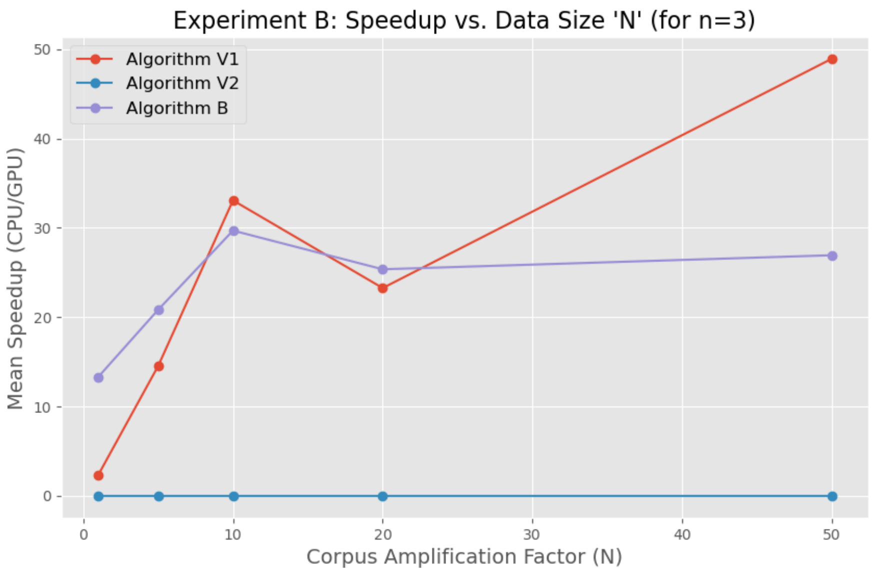
UNIVERSITÀ DEGLI STUDI FIRENZE

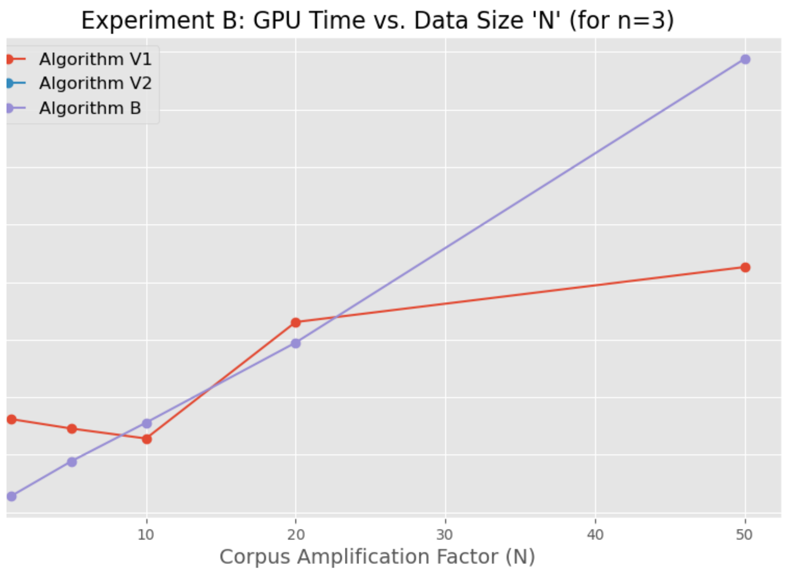
Experiment B: Scalability vs Data Size (N)





Experiment B: Scalability vs Data Size (N)





UNIVERSITÀ DEGLI STUDI FIRENZE

Conclusions

There is no single best algorithm; efficiency depends strictly on problem complexity (n).

- **V1**: Dominates for n≤3 due to linear time complexity, but fails at large values of n due to exponential memory limits.
- **V2**: The memory multiplier causes early OOM failures (n≥3) and overhead slows down low-complexity cases.
- **B**: The only viable option for high-complexity domains (n≥4) thanks to linear memory complexity.