## Introduction and Objective

The aim of this project is to improve the efficiency of the CPU-based C++ text search program by leveraging the computational power of a GPU. Originally, the CPU program sequentially counted occurrences of specific target words within large text files, this approach limits performance due to the CPU’s sequential processing nature.

In a CPU-based program, counting the number occurrences of specific words in large text files is done sequentially, which is done by scanning each character in the file one at a time, comparing it to the target words selected. While this method is straightforward, this sequential approach is limited by the CPU’s linear processing capabilities, which means, it processes each position in the text individually and in order. This limitation becomes especially apparent with large datasets, since each comparison has to wait for the previous one to finish, which causes delays. In contrast, parallel processing, such as on a GPU, could handle multiple sections of text simultaneously, dramatically improving performance by reducing the reliance on sequential processing.

While CPUs process in a limited and sequential manner, the GPU on the otherhand, uses thousands of cores to process many tasks in parallel, Thus, GPUs can process big datasets extremely quickly, therfore, it can produce an unparalleled speed advantage compared to the CPU process. This allows for an high throughput while ensuring efficient memory access and scalable resource usage across multiple threads. This level of parallelism makes this ideal finding occurrences of specific words within a text file, as each thread may handle its sub-part with total independence, and this notably accelerates the performance.

By utilizing the parallel processing capabilities of the GPU, the objective was to transform this algorithm into a highly efficient version capable of handling large data volumes rapidly.  
  
Counting word occurrences across large text files is a key task in applications such as document indexing, data analytics, and natural language processing. CPUs are effective at handling such tasks on smaller datasets but are constrained by limited cores and their inherently sequential operations. In contrast, GPUs, with their thousands of cores, offer a way to perform these operations in parallel, providing significant time savings when managing large datasets.

## Porting the Algorithm to the GPU

To port the text search algorithm to the GPU, several critical steps were taken to maximize the parallel processing and also to ensure accurate results:

**Firstlly the tasks were Divided across the threads**, the tasks were divided so that the text data for each GPU thread was responsible for checking specific segments, enabling multiple comparisons to be done simultaneously. This division ensured that each part of the text was analyzed concurrently, leveraging the GPU’s massive parallelism.

Secondly, the CUDA kernel was created, called calc\_token\_occurrences\_kernel, which is designed to process the data in parallel. By ensuring that each thread within the kernel is checking for the target words, starting at its assigned position, coordinating through atomic operations to the record occurrences.

**Thirdly,** A custom function, gpu\_strncmp, was implemented to compare strings directly on the GPU. This allowed threads to determine if a target word matched within their assigned text segment, using a parallel version of standard string comparison.

**Finally, to handle the out of bounds conditions**, to ensure accuracy, each thread, checked that the words matches occurred at word boundaries (e.g., “the” vs. “their”). This check verified that found words were not part of larger words, refining results to include only whole-word matches.

## Optimizing the Algorithm and Kernel Setup

Optimization was crucial to maximize the performance of the GPU implementation. Several strategies were tested and applied:

1. Global Memory Coalescing  
2. Exploring Shared Memory Usage  
3. Occupancy and Thread Count Tuning  
4. Optimizing Atomic Operations  
5. Experimental Adjustment of Block Sizes

## 4. Hardware Setup, Results, and Analysis

### Hardware Setup

## Number of devices: 1

## Device 0

## Name NVIDIA GeForce RTX 3070

## Revision 8.6

## Memory 8191MB

## Warp Size 32

## Clock 1725000

## Multiprocessors 46

## 5. Conclusion

By porting the text search algorithm to the GPU, this project achieved substantial performance gains compared to the original CPU-based version. The optimizations applied—including occupancy management, block size tuning, and strategic use of global memory—enabled the GPU implementation to handle large text files efficiently. Shared memory proved unnecessary, as its overhead outweighed the potential benefits for non-reusable data access patterns. Atomic operations introduced a bottleneck for high-frequency words, highlighting an area for further optimization using techniques like histogram-based counting.