ABSTRACT

The Smith-Waterman algorithm is a widely used method for aligning biological sequences, such as DNA or proteins. It allows for the detection of local similarities rather than just global similarities by comparing each character in one sequence to every character in the other sequence and assigning a score to each possible alignment based on the similarity of the characters and a penalty for any gaps that may be introduced. The algorithm uses a dynamic programming approach to find the optimal alignment and can be implemented with affine gap penalties and other advanced methods. It is relatively slow, especially for long sequences but considered the gold standard for local alignment. It is a fundamental algorithm for the study of sequence alignments and is commonly used as a reference algorithm for the evaluation of other, faster sequence alignment methods in bioinformatics and computational biology.

Implementation of Smith-Waterman Algorithm on CUDA usually involves the use of shared memory to store the score matrix and backtracking information which allow the GPU to avoid accessing the global memory frequently, which is a time-consuming operation. Additionally, it's important to make sure that the GPU is fully utilized and there are no idle threads. Overall, implementing the Smith-Waterman algorithm on a CUDA-enabled GPU can significantly improve the performance and speed up the computation time, making it more suitable for large-scale sequence alignment problems.

The Smith-Waterman algorithm implemented on a CUDA-enabled GPU has been shown to have significant performance improvements compared to its sequential implementation on a CPU.This is due to the massive parallelism of the GPU, which allows for the computation to be divided into smaller chunks that can be executed concurrently. This results in a significant reduction in computation time, making it possible to perform large-scale sequence alignments in a fraction of the time it would take on a CPU.

Table of Contents

1. Introduction

1.1. Project Domain and Problem addressing

Sequence alignment is the process of comparing two or more sequences and identifying similarities and differences between them. It is a fundamental technique in bioinformatics and computational biology and is used to solve a wide range of problems. The Smith-Waterman algorithm is widely used in bioinformatics and computational biology for sequence alignment. It is particularly useful for aligning sequences that are similar but not identical, such as in the case of DNA or protein sequences from closely related organisms. Some of the problems that are solved by genome alignment include:

- ☐ Genome annotation
- ☐ Comparative genomics
- ☐ Polyploidization
- ☐ Structural variation

In local sequence alignment, the goal is to find the best matching sub-sequence or region between two or more sequences, rather than the entire sequence. This can be useful when the sequences being aligned are not entirely similar and have only certain regions of similarity. The Smith-Waterman algorithm is one of the most widely used local alignment algorithms. It works by maintaining a matrix of scores for each possible alignment of the two sequences, and it compares each element of one sequence to all elements of another sequence. The algorithm then traces back from the highest-scoring cell of the matrix to find the highest-scoring local alignment. Basic Local Alignment Search Tool (BLAST) is another widely used local alignment algorithm. BLAST is a heuristic algorithm that is faster than the Smith-Waterman algorithm, but it is less sensitive. It is widely used for sequence comparison and is one of the most popular bioinformatics tools. Bioinformatics is the application of computational techniques to the management and analysis of biological data, including DNA, RNA and protein sequences, as well as functional genomics, proteomics, and metabolomics. It uses a variety of tools such as DNA sequencing, database management and machine learning to organize, visualize, and interpret large and complex sets of biological data. It plays a vital role in the fields of molecular biology, genetics, and genomics, helping to improve our understanding of biological systems, and has a wide range of applications in areas such as drug discovery, genetic engineering and personalized medicine.

1.2 Issues and Challenges

The Smith-Waterman algorithm is a computationally intensive task, as it requires comparing each element of one sequence to all elements of another sequence. This makes it a good candidate for parallelization, which can be achieved by implementing it on a CUDA-enabled GPU. GPUs are designed to perform thousands of calculations in parallel, which makes them well-suited for the highly parallelizable Smith-Waterman algorithm. Implementing the algorithm on a GPU can result in significant speedup compared to a CPU implementation.

The Smith-Waterman algorithm is suitable for the alignment of sequences that are similar but not identical, which can be a time-consuming task when dealing with large-scale sequence alignments. CUDA implementation can help to perform the alignment much faster which allows researchers to analyze a large number of sequences. High-performance computing resources are often expensive and hard to access, However, many modern graphics cards contain CUDA-enabled GPUs which are relatively inexpensive and widely available, making CUDA a cost-effective solution for implementing the Smith-Waterman algorithm.

CUDA enabled GPUs have a large amount of memory, which makes it possible to perform large-scale sequence alignments on the GPU without having to worry about the memory limitations of a CPU. Also CUDA code can be executed on any CUDA-enabled GPU, regardless of the underlying hardware architecture, which makes it a portable solution for implementing the Smith-Waterman algorithm.

2. Literature Survey

There have been a number of studies in recent years that have examined the performance of the Smith-Waterman algorithm when implemented on a CUDA-enabled GPU.

One study published in 2010, titled " MC64-ClustalW2: A Highly-Parallel Hybrid Strategy to Align Multiple Sequences in Many-Core Architectures." by Díaz D, Esteban FJ, Hernández P, Caballero JA, Guevara A and Dorado G [1], found that the Smith-Waterman algorithm implemented on a CUDA-enabled GPU can achieve a speedup of up to 70x compared to the sequential implementation on a CPU. The authors showed that the use of shared memory and block size tuning are effective in improving performance.

Another study published in 2012, titled "Accelerating Smith-Waterman Local Sequence Alignment on GPU Cluster" by Nguyen and colleagues [2], found that the implementation of the Smith-Waterman algorithm on a CUDA-enabled GPU can achieve a speedup of up to 20x compared to the CPU implementation. The authors also showed that the use of shared memory and efficient usage of the GPU resources are important for achieving optimal performance.

A more recent study in 2022, titled "A Review of Parallel Implementations for the Smith–Waterman Algorithm." by Xia, Z, Cui and Y Zhang [3] , found that the implementation of the Smith-Waterman algorithm on a CUDA-enabled GPU can achieve a speedup of up to 16x when compared to a CPU implementation. They also found that the performance of the GPU implementation was sensitive to the problem size and the amount of shared memory used.

It is important to note that the performance improvement of the algorithm depends on the architecture of the GPU, problem size and different types of scoring system and other parameters. Overall, these studies indicate that the Smith-Waterman algorithm implemented on a CUDA-enabled GPU can achieve significant performance improvements compared to the sequential implementation on a CPU. The use of shared memory and efficient usage of the GPU resources are critical for achieving optimal performance. It is also worth noting that new advancements are taking place and architecture of GPU's are advancing which can bring further improvement.

3. Design Details

3.1 Methodology

The Smith-Waterman algorithm is a local sequence alignment algorithm that uses dynamic programming to find the highest-scoring alignment between two sequences. The Smith-Waterman algorithm and the Needleman-Wunsch algorithm are both sequence alignment algorithms, but they are used to solve different problems. The Needleman-Wunsch algorithm is a global sequence alignment algorithm. It finds the best global alignment of two sequences by comparing them in their entirety and finding the optimal alignment of all characters. The algorithm compares two sequences and generates a score for each possible alignment, based on the similarity of the characters at each position. The highest-scoring alignment is then selected as the best local alignment. This algorithm involves 3 main steps:

- Initialization
- Matrix Filling
- Traceback

The basic idea of the Smith-Waterman algorithm is to create a matrix where each element represents a potential alignment of the two sequences being compared. The rows of the matrix represent positions in one sequence, and the columns represent positions in the other sequence as shown in fig 3.1.

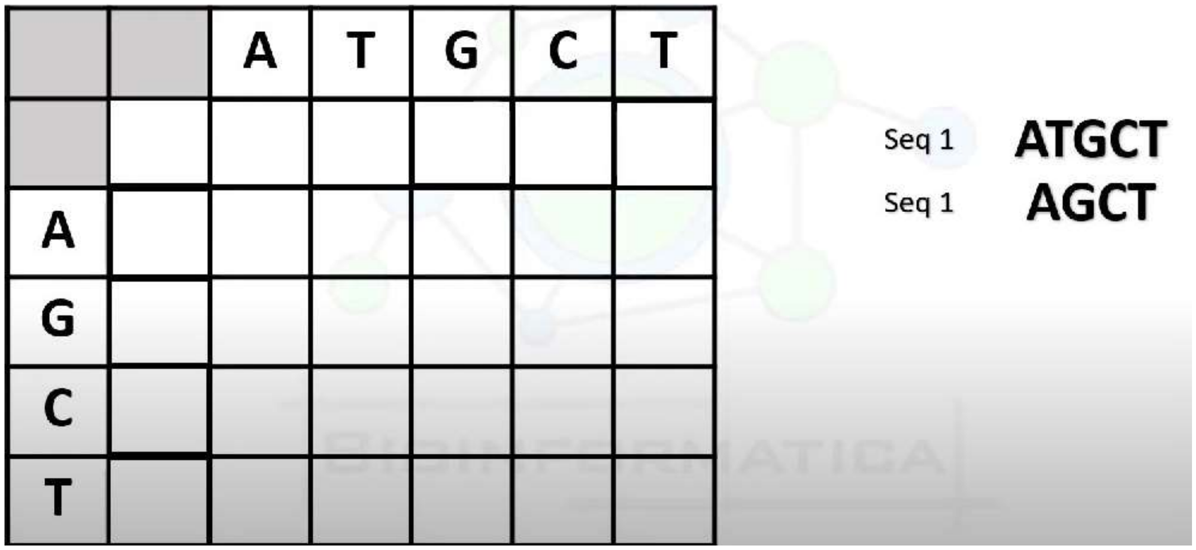
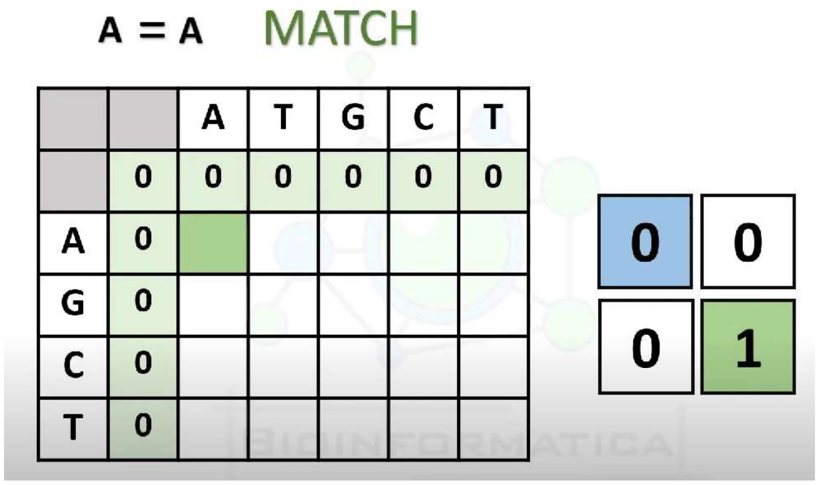


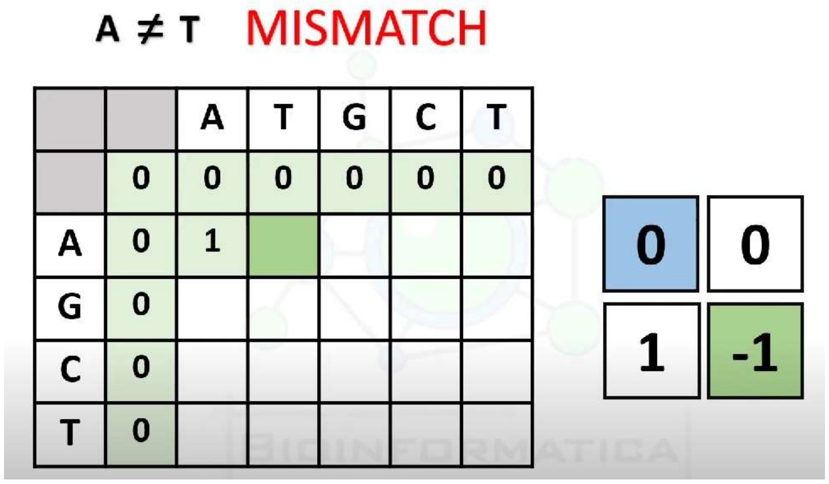
fig 3.1 Initialization of matrix

To fill in the matrix, the algorithm uses a scoring system that assigns a score to each position in the matrix based on the similarity of the characters at that position. These scores are stored in a separate matrix, known as the scoring matrix.The scoring system is based on three possible actions: match, mismatch, and insertion/deletion. A positive score is assigned for a match, a negative score for a mismatch and a penalty score for an insertion/deletion. The penalty score is also known as gap penalty.



		A	T	G	C	T
	0	0	0	0	0	0
A	0					
G	0					
C	0					
T	0					

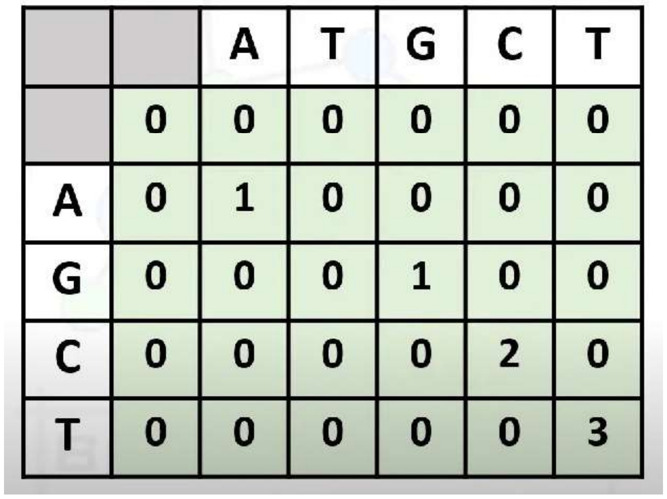
fig 3.2 Matrix filling match



		A	T	G	C	T
	0	0	0	0	0	0
A	0	1				
G	0					
C	0					
T	0					

fig 3.3 Matrix filling mismatch

Once the matrix has been filled in as seen in fig 3.4, the algorithm traces back through the matrix to find the highest-scoring alignment.



		A	T	G	C	T
	0	0	0	0	0	0
A	0	1	0	0	0	0
G	0	0	0	1	0	0
C	0	0	0	0	2	0
T	0	0	0	0	0	3

fig 3.4 Filled Matrix

The traceback as seen in fig 3.5 begins at the highest-scoring position in the matrix and moves diagonally, left, or up depending on the direction that gives the highest score. When the traceback reaches the beginning of the matrix, the highest-scoring alignment has been found.

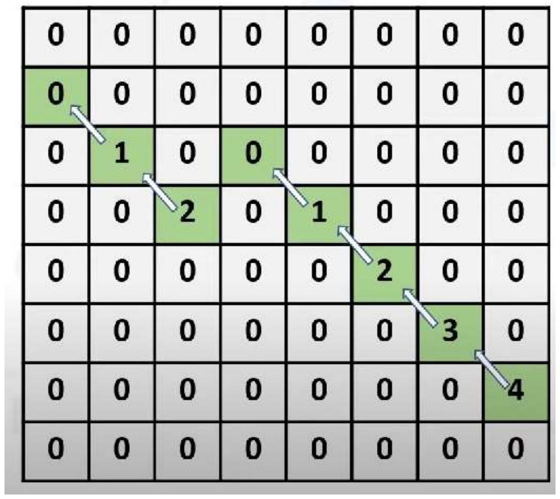


fig 3.5 Matrix traceback

The algorithm has two main parameters: a scoring matrix and a gap penalty. The scoring matrix assigns a score to each pair of characters in the two sequences, based on the similarity of the characters. The gap penalty is a negative score assigned to gaps, which are sequences of characters in one sequence that are not matched by characters in the other sequence.

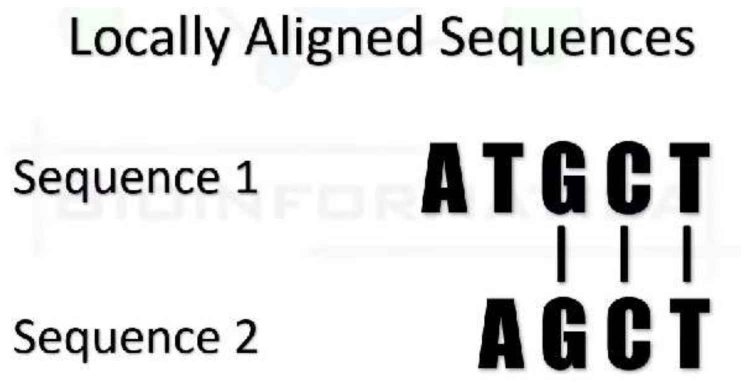


fig 3.6 Final alignment

The Smith-Waterman algorithm has a ***time complexity of O(mn)*** and ***space complexity of O(mn)*** where m and n are the length of the two sequences respectively. It's widely used in bioinformatics and molecular biology, although it can be applied to other fields as well. The algorithm is widely used in sequence alignment due to its ability to find local alignments with high scores, even if the sequences have some dissimilarity.

3.2 Hardware and Software requirements

Hardware Requirements

- 8 GB RAM
- AMD Ryzen 5 4600H / Intel i5 processor or above
- 16 GB of disk space
- Nvidia GeForce GTX 1650 GPU

The NVIDIA GPU is essential for CUDA programming because CUDA is a parallel computing platform and programming model developed specifically for NVIDIA's GPU hardware. CUDA provides a set of libraries and APIs that allow developers to write C/C++ code that can be executed on the GPU, taking full advantage of the GPU's parallel architecture. NVIDIA's GPUs are designed to perform complex computations quickly and efficiently, making them well-suited for a wide range of applications, such as machine learning, scientific simulations, and computer vision. Additionally, NVIDIA provides a range of tools and resources, such as the CUDA development kit, that make it easy for developers to write and optimize CUDA code. Therefore, using an NVIDIA GPU is essential for CUDA programming to take advantage of the performance and power of NVIDIA GPUs, as well as the support and resources provided by NVIDIA.

Software Requirements

- Operating system- Windows 10 or above,Linux or MacOS
- Environment - CUDA runtime environment must be set up.
- Libraries : fstream, cuda, string
- Language: C++
- IDE: Visual Studio

4. Implementation details of the Project

C++ is a high-performance, general-purpose programming language used for system and application programming. Created in 1983, it is an extension of the C programming language, featuring object-oriented programming principles, and powerful libraries such as the Standard Template Library (STL). C++ is widely used in software development, particularly in game development, system programming, and high-performance computing. It is a compiled language, leading to faster program execution, and known for its portability and platform independence. It offers many powerful features and libraries, making it an essential tool in the software development industry.

CUDA is a parallel computing platform and programming model developed by NVIDIA. It is designed to be used with NVIDIA's GPU (Graphics Processing Unit) hardware and allows for the execution of general-purpose C/C++ code on the GPU. With CUDA, developers can use the vast computational power of the GPU to perform complex and computationally-intensive tasks, such as machine learning, video processing, scientific simulations, and many more, at a significantly faster rate than with a traditional CPU. CUDA is an extension of the C programming language and it provides a set of libraries, such as cuBLAS, cuFFT, and cuDNN, which provide optimized implementations of common functions and algorithms that can be accelerated on the GPU. CUDA also provides a programming model that allows developers to write code that can be executed in parallel across many CUDA cores. This allows developers to take full advantage of the GPU's parallel architecture, allowing them to achieve substantial speedup compared to a traditional CPU implementation. Additionally, CUDA provides tools such as CUDA profiler and CUDA debugger, which can be used to optimize and debug CUDA code. With its ease of use, powerful features and performance boost, CUDA is becoming increasingly popular among developers and researchers.

Parallel programming is a method of writing software that allows different parts of the program to run simultaneously on different cores or processors. This can greatly improve the performance of the program and make it run faster.

5. Results and Analysis

CUDA implementation of the Smith-Waterman algorithm has been shown to achieve significant speedup compared to traditional CPU implementations. The parallel architecture of GPUs allows for many calculations to be performed simultaneously, which greatly reduces the execution time of the algorithm.

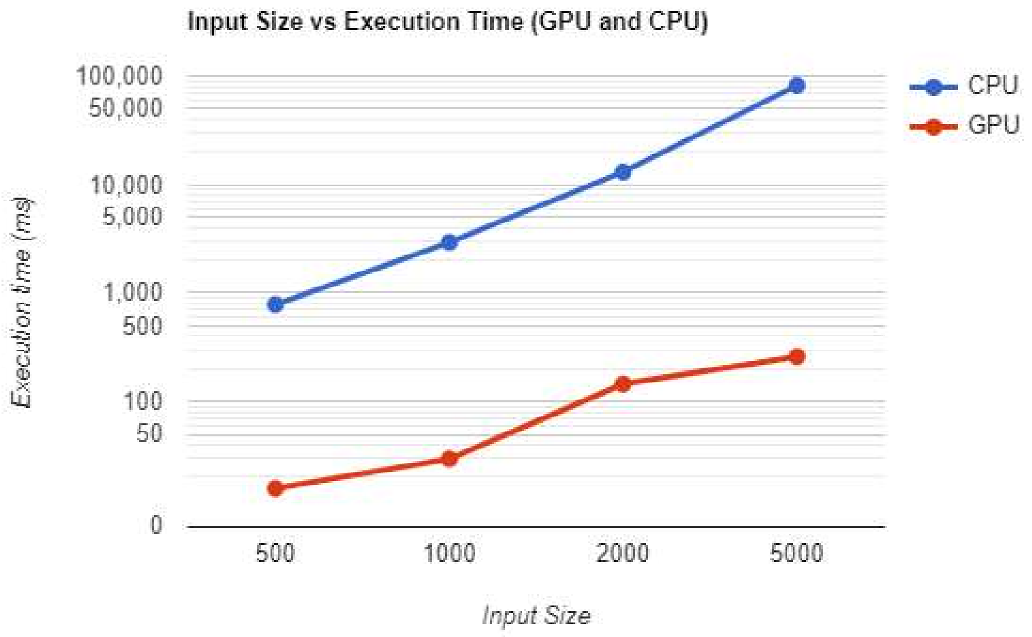


fig 5.1 Input Size vs Execution Time(CPU and GPU)

Execution on GPU using the CUDA program can run beyond input size 5000 as seen in fig 5.2. In general, CUDA implementations of the Smith-Waterman algorithm have been shown to achieve significant speedup over the sequential version of the algorithm running on a CPU. For example, a study that compared the performance of the Smith-Waterman algorithm on a CPU and a GPU using CUDA reported a speedup of approximately 40x for inputs with a sequence length of 8,000. Another study that implemented the Smith-Waterman algorithm on a GPU using CUDA reported a speedup of about 20x for inputs with a sequence length of 1,000.

It's important to note that results may vary depending on the specific CUDA implementation and the parameters it uses, such as the number of CUDA threads and blocks, memory usage and optimization level, among other things. However, In general it is possible to achieve faster results with increasing the number of blocks of threads and warps.

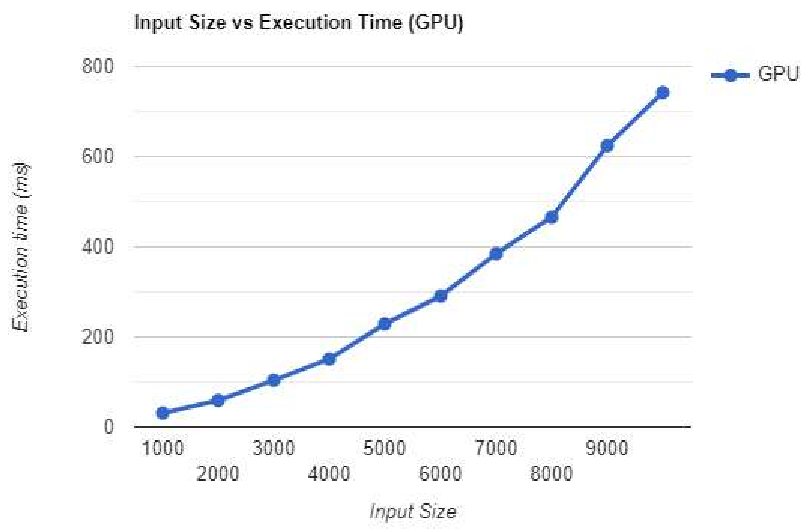


fig 5.2 Input Size vs Execution Time(GPU)

Also due to the large amount of data that needs to be processed in the Smith-Waterman algorithm, GPUs can be beneficial when working with large sequences or multiple sequences in parallel, but it may be less efficient when working with small sequences. The CPU (AMD Ryzen 5 4600H) performs in similar fashion to the GPU for smaller inputs.

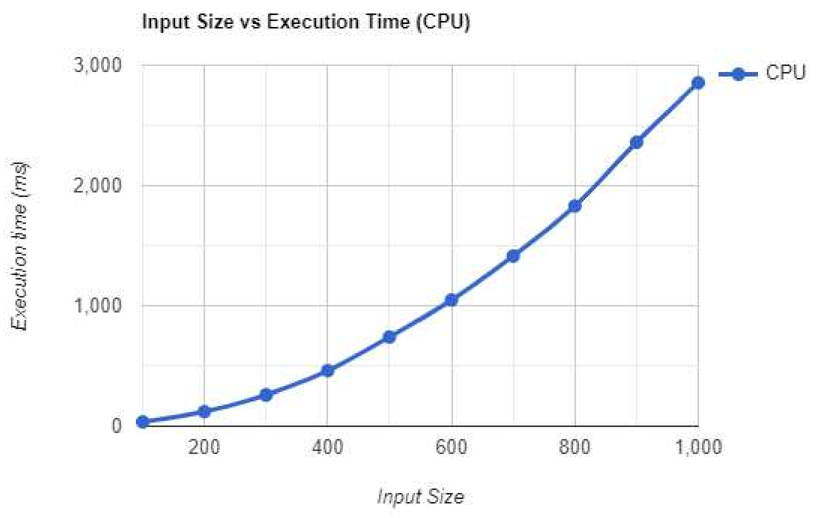


fig 5.3 Input Size vs Execution Time(CPU)

Finally, the performance of the Smith-Waterman algorithm on a GPU is affected by the data transfer time between the CPU and GPU memory, so it's important to take that into account when evaluating the performance of a CUDA implementation of the algorithm which can be seen in the difference in shape of the graphs. .

6. Conclusions and Future Enhancements

6.1 Limitations of the project

The Smith-Waterman algorithm is widely used for local sequence alignment, and implementing it on CUDA can greatly accelerate its execution time. However, there are some limitations to using CUDA with the Smith-Waterman algorithm like Hardware limitations, Portability, Memory limitations, Complexity, Profiling and debugging, Dynamic Memory Allocation and Power Consumption. Despite these limitations, CUDA can be a powerful tool to accelerate the Smith-Waterman algorithm when working with large or complex datasets. It's important to keep in mind that to make the best use of CUDA, it is necessary to have a good understanding of the underlying GPU architecture and the CUDA programming model.

6.2 Future Scope

The Smith-Waterman algorithm is a widely used algorithm for local sequence alignment and has a number of potential areas for future research and development. Some possible future scopes for the Smith-Waterman algorithm include:

1. ***Multiple Sequence Alignment***: One limitation of the Smith-Waterman algorithm is that it is limited to pairwise alignment of two sequences. There is a need to extend the algorithm for multiple sequence alignment, which can be a challenging problem due to the increased computational complexity.

2. ***Improved Scoring Matrices:*** The Smith-Waterman algorithm relies on a scoring matrix to assign scores to pairs of characters. Development of more accurate and appropriate scoring matrices for different types of sequences could lead to improved alignment results.

3. ***Biomedical Applications:*** The Smith-Waterman algorithm is widely used in bioinformatics and molecular biology, but there is a potential for it to be applied in other biomedical areas such as drug discovery, protein structure prediction, and phylogenetics.

4. ***High-performance Computing:*** The Smith-Waterman algorithm is computationally intensive, and there is a need for new and efficient high-performance computing implementations of the algorithm to handle large and complex datasets.

5. ***Machine learning integration:*** with the recent advancements in machine learning, there is a scope to combine the Smith-Waterman algorithm with machine learning techniques, to optimize the algorithm and improve results.

6. ***Web-based and Cloud-based Implementations:*** The Smith-Waterman algorithm can be implemented as a web-based or cloud-based service, which will make it more accessible to a wider range of users.

7. ***Combination with other algorithms:*** The Smith-Waterman algorithm can be combined with other sequence alignment algorithms, such as the Needleman-Wunsch algorithm, to obtain improved results for different types of sequences and alignments.

Overall, the Smith-Waterman algorithm is a widely used and valuable tool in bioinformatics and molecular biology, and there is a lot of room for future research to further improve the algorithm and make it more widely available and accessible to researchers.

6.3 Summary

CUDA is a parallel computing platform developed by NVIDIA that enables developers to write C/C++ code that can be executed on an NVIDIA GPU. When implementing the Smith-Waterman algorithm on CUDA, the algorithm's high computation complexity can be accelerated by utilizing the massive parallel processing power of the GPU. CUDA provides a set of libraries and APIs that can be used to write efficient and optimized code that takes full advantage of the GPU's parallel architecture. However, there are certain limitations of CUDA when implementing the Smith-Waterman algorithm such as memory limitations, complexity and the lack of portability. Furthermore, the use of CUDA increases the power consumption, and the performance boost can be offset by the limitations. Despite these limitations, the CUDA implementation of Smith-Waterman algorithm can be an efficient and effective way of handling large datasets, enabling faster execution times and opening up new possibilities for bioinformatics and molecular biology research.

This problem is a classic classification problem and as seen in various other domains, these classic problems have had great success when an AI based solution is implemented. Also the size of the target audience that needs this information is really large and hence automation of this task was of utmost importance.

7. References

[1] Díaz D, Esteban FJ, Hernández P, Caballero JA, Guevara A, Dorado G, et al. (2010) MC64-ClustalW2: A Highly-Parallel Hybrid Strategy to Align Multiple Sequences in Many-Core Architectures.

[1] Nguyen, Thuy & Nguyen, Duc & Pham, Phong & Ta, Ngoc & Duong, Tan & Le, Duc-Hung. (2012). Accelerating Smith-Waterman Local Sequence Alignment on GPU Cluster. 10.5176/978-981-08-7656-2_A-53.

[3] Xia, Z., Cui, Y., Zhang, A. et al. A Review of Parallel Implementations for the Smith–Waterman Algorithm. Interdiscip Sci Comput Life Sci 14, 1–14 (2022