



From Serial to Parallel: Enhancing Needleman-Wunsch Performance through GPU-Based Computing

Yustina Sri Suharini¹, Wisnu Ananta Kusuma², Sri Nurdianti³, Irmanida Batubara⁴

^{1,2}Department of Computer Science, School of Data Science, Mathematics and Informatics, IPB University, Bogor, Indonesia

³Department of Mathematics, School of Data Science, Mathematics and Informatics, IPB University, Bogor, Indonesia

⁴Department of Chemistry, Faculty of Mathematics and Natural Sciences, IPB University, Bogor, Indonesia

²Bioinformatics Study Program, Faculty of Mathematics and Natural Sciences, IPB University, Bogor, Indonesia

^{2,4}Tropical Biopharmaca Research Center, IPB University, Bogor, Indonesia

¹Department of Informatics Engineering, Institut Teknologi Indonesia, Tangerang Selatan, Indonesia

¹yustina.ss@iti.ac.id, ²ananta@apps.ipb.ac.id, ³nurdianti@apps.ipb.ac.id, ⁴ime@apps.ipb.ac.id

Abstract

The increasing demand for faster bioinformatics analysis calls for more efficient approaches for sequence alignment. In this study, we demonstrate that a GPU-based implementation of the Needleman-Wunsch algorithm can achieve up to $14.8\times$ speedup compared to its traditional CPU-based serial counterpart, without compromising alignment accuracy. By leveraging the parallel processing capabilities and shared memory of an NVIDIA GeForce RTX 3060 Laptop GPU, we significantly accelerated global sequence alignment tasks. Using clinically relevant genes such as NRAS, BRCA1, BRCA2, and Saccharomyces cerevisiae from NCBI ensures realistic alignment challenges and biological significance. Performance evaluation across a wide range of sequence lengths demonstrates the scalability and efficiency of the parallel approach. More importantly, this study provides a unique contribution by showing that a commodity GPU, such as the NVIDIA GeForce RTX 3060 Laptop, can serve as a practical alternative when high-performance computing clusters are unavailable or prohibitively expensive, thereby offering an accessible and cost-effective pathway to high-throughput bioinformatics workflows.

Keywords: global sequence alignment; GPU computing; Needleman-Wunsch algorithm

How to Cite: Y. S. Suharini, W. A. Kusuma, S. Nurdianti, and I. Batubara, "From Serial to Parallel: Enhancing Needleman-Wunsch Performance through GPU-Based Computing", *J. RESTI (Rekayasa Sist. Teknol. Inf.)*, vol. 9, no. 5, pp. 1183 - 1192, Oct. 2025.

Permalink/DOI: <https://doi.org/10.29207/resti.v9i5.6620>

Received: May 2, 2025

Accepted: September 26, 2025

Available Online: October 25, 2025

*This is an open-access article under the CC BY 4.0 License
Published by Ikatan Ahli Informatika Indonesia*

1. Introduction

Global sequence alignment plays a crucial role in bioinformatics, allowing researchers to identify both similarities and differences between genetic sequences. The Needleman-Wunsch algorithm, a well-established method for global sequence alignment, utilizes dynamic programming to find the optimal alignment between two sequences [1]. Despite its effectiveness, the algorithm's computational complexity grows rapidly with the size of the input sequences, resulting in long execution times and making it inefficient for large-scale genomic data analysis. To address this challenge, researchers have explored a variety of hardware acceleration techniques to speed up the execution of sequence alignment algorithms. Among these, Application-Specific Integrated Circuits (ASICs) have been employed to optimize sequence alignment for specific tasks, with notable contributions demonstrating the potential of ASICs in bioinformatics applications

[2], [3]. Similarly, Field Programmable Gate Arrays (FPGAs) have been widely used to accelerate DNA sequence alignment due to their reconfigurability and parallelism. Several studies have leveraged FPGA-based solutions, achieving significant speedups for sequence alignment tasks [4] - [6]. Alternatively, multi-core processors, such as quad-core CPUs, have been explored as a means to parallelize sequence alignment algorithms using parallel programming frameworks like OpenMP. In their work, the performance of the Needleman-Wunsch algorithm on multi-core processors with OpenMP was investigated, demonstrating improvements in execution time over single-core implementations [7], [8]. They also compared column-wise and row-wise paradigms to anti-diagonal-wise [9]. In recent years, Graphical Processing Units (GPUs) have garnered significant attention for accelerating DNA sequence alignment. The massive parallelism offered by GPUs makes them particularly well-suited for tasks such as dynamic

programming, which is central to algorithms like Needleman-Wunsch. Numerous studies have explored GPU acceleration for sequence alignment [10] - [12]. These efforts have demonstrated substantial speedup in alignment tasks through the use of GPUs, often in combination with multi-threading techniques to further optimize performance [13] - [15]. Moreover, research into heterogeneous clusters, combining multiple computing resources such as GPUs, CPUs, and FPGAs, has shown promising results in scaling DNA sequence alignment algorithms for large datasets. Notable contributions in this area highlighting the potential of hybrid systems for accelerating bioinformatics applications [16] - [18]. In recent studies, multi-GPU systems such as NVIDIA's Tesla series have been utilized to further enhance computational performance [19], [20]. These studies demonstrated the effectiveness of utilizing multiple GPUs in parallel to significantly reduce the execution time of sequence alignment tasks, thereby advancing the state of hardware acceleration in the field of bioinformatics. However, these studies often relied on high-end computational infrastructures, which are not always accessible to all researchers or institutions. Motivated by this limitation, our study explores an alternative approach by utilizing commodity hardware to evaluate the practical feasibility of GPU-based acceleration in more accessible settings.

In this paper, we focus on GPU-based acceleration of the Needleman-Wunsch (NW) algorithm, comparing the performance of GPU and CPU implementations to demonstrate the potential for significant reduction in execution time. We present an analysis of the results using three key performance metrics: the break-even point, which identifies when GPU acceleration becomes advantageous over CPU execution; log performance analysis, which examines the scalability of the algorithm as the input size grows; and speedup, which quantifies the overall performance improvement achieved by leveraging the parallelism of GPU. The results are summarized in a performance table to provide a clear comparison across different hardware platforms. This context highlights the need for accessible performance benchmarks on realistic hardware setups. Through this study, we aim to contribute to the growing body of work on hardware-accelerated sequence alignment algorithms and to provide clear insights into the practical benefits of using GPUs in bioinformatics compared to traditional CPU-based approaches. Distinct from prior studies that often depend on specialized or high-end computing infrastructures, this paper presents a systematic performance benchmark of the Needleman-Wunsch algorithm on a readily available commodity GPU, covering realistic sequence lengths that span from short motifs to large genomic fragments. By identifying a practical break-even point and analyzing scalability under typical hardware constraints, this work demonstrates that high-performance sequence alignment can be achieved cost-effectively, making

advanced bioinformatics analyses more accessible to researchers and institutions operating outside of elite computing environments.

2. Methods

This section describes the methodology used to implement and evaluate the NW algorithm on both CPU and GPU environments. It outlines the design of the algorithm, the software and hardware setup, and the procedures used to measure and analyze computational performance.

2.1 NW Algorithm

If the nucleotide base pairs X and Y are defined by Formulas 1 and 2, respectively,

$$X_i = x_1, x_2, x_3, \dots, x_n \quad (1)$$

$$Y_i = y_1, y_2, y_3, \dots, y_m \quad (2)$$

The NW algorithm performs the following steps: matrix initialization, matrix filling, tracing back and alignment construction to do X and Y sequence alignment process [1].

Matrix Initialization: The matrix is first initialized together with the gap penalty g as shown in Formula 3. The scoring values for matches and mismatches are also defined, as described in [15].

$$D(0, j) = j \cdot g, \quad D(i, 0) = i \cdot g \quad (3)$$

Matrix Filling: After initialization, the matrix elements are filled according to the recurrence relations defined in Formula 4 [1], [13].

$$D(i, j) = \begin{cases} D(0,0) = 0 \\ \max \begin{cases} D(i-1, j-1) + s(x_i, y_i) \\ D(i-1, j) + g_x \\ D(i, j-1) + g_y \end{cases} \end{cases} \quad (4)$$

Each cell is computed based on the optimal score obtained from match, mismatch, or gap conditions, with scoring function $s(x_i, y_i)$ is given by Formula 5 as referenced in [9].

$$s(x_i, y_i) = \begin{cases} +matchscore, & x_i = y_i \\ -mismatchpenalty, & x_i \neq y_i \end{cases} \quad (5)$$

Traceback and Alignment Construction: Once the matrix is filled, the algorithm performs a traceback starting from the bottom-right corner of the matrix to reconstruct the optimal alignment between the sequences.

The traceback phase of the NW algorithm begins from the bottom-right cell of the dynamic programming matrix. If the nucleotide bases from the reference and query sequences match, the algorithm proceeds diagonally toward the top-left cell. In the event of a mismatch, the traceback direction is determined by selecting the path associated with the highest score among three possibilities: left (horizontal), up (vertical), or upper-left (diagonal). In the matrix

configuration used, the reference sequence is arranged along the top row, while the query sequence is placed along the leftmost column.

The reconstruction of the final alignment also commences from the bottom-right cell. During this process, a transition from a higher score to a lower score typically corresponds to a match, whereas a transition from a lower score to a higher score indicates a mismatch. Vertical movements represent insertions, and horizontal movements represent deletions. By consistently applying these principles throughout the traceback, the complete optimal alignment between the sequences can be accurately obtained.

2.2 Software, Tools, and Hardware Setup

The NW algorithm for sequence alignment was implemented both in serial and parallel using the C++ programming language. The serial implementation was designed to execute the alignment process sequentially on the CPU, while the parallel version utilized the CUDA platform to leverage the computational power of the GPU for acceleration. The parallel implementation offloaded computationally intensive tasks, such as the filling of the dynamic programming matrix, to the GPU, significantly reducing the time required for large-scale sequence alignments. Both versions of the program were developed using Joe's Editor, a lightweight and customizable code editor, and were executed on a system running Linux Ubuntu 6.8.0-45. The C++ standard library was used for basic data structures and algorithms, while CUDA-specific libraries and functions were employed to implement GPU parallelization. The software was compiled using the GCC and NVCC with optimization flags to ensure efficient execution on both the CPU and GPU.

The computational experiments in this study were performed on a system equipped with an AMD Ryzen 7 5800H processor, operating at a clock speed of 3.4 GHz. Built on the Zen 3 architecture, this processor features 16 physical cores and 16 threads, enabled by simultaneous multithreading, making it well-suited for parallel processing tasks. To further accelerate these tasks, the CPU is paired with an NVIDIA GeForce RTX 3060 Laptop GPU. The GPU is equipped with 3840 CUDA cores spread across 30 multiprocessors, delivering robust parallel processing capabilities. With a maximum clock rate of 1425 MHz and a memory clock rate of 7001 MHz, it is designed to handle high-performance computations. The GPU has 5938 MB of global memory, offering ample capacity for large datasets, and supports CUDA Capability 8.6, optimizing the execution of parallel workloads. Additionally, the GPU features Unified Addressing (UVA) for more efficient memory management between the GPU and system memory, along with a 3 MB L2 cache and 49152 bytes of shared memory per block to further enhance performance. GPU memory

usage was managed through explicit allocation and deallocation using `cudaMalloc` and `cudaMemcpy`. After each allocation, the implementation checked for allocation errors to ensure that the available global memory was sufficient for the required matrices; if the allocation failed due to insufficient memory, the process was halted to prevent overflow. This approach was consistently applied to both short and long sequences without separate handling. This hardware configuration was specifically selected to evaluate the performance of the NW algorithm under both serial and parallel execution conditions.

2.3 Serial Algorithm and Parallel Implementation

For example, if there is a nucleotide base pair CGAG and CTAG, then filling matrix of serial Needleman-Wunsch (NW) algorithm will perform 16 steps as shown in Figure 1. In contrast, parallel version of the algorithm is illustrated in Figure 3.

The serial version of the NW algorithm was implemented in C++ and executed on the CPU to perform global sequence alignment. The algorithm employs dynamic programming to construct an optimal alignment by filling a scoring matrix that represents the similarity between two input sequences. Each cell in the matrix corresponds to a possible alignment between characters from both sequences, and the matrix is filled iteratively by considering matches, mismatches, and gaps. The scoring matrix is generated based on several key parameters: the match score, mismatch penalty, and gap penalty. By default, if the user does not specify custom values, the following scoring scheme is applied: a match score of +1, a mismatch penalty of -1, and a gap penalty of -2. This scoring scheme was chosen to provide a balanced trade-off between matching characters and penalizing mismatches and gaps. Users have the flexibility to adjust these parameters to optimize the alignment for different biological sequences or alignment criteria. The algorithm was designed to ensure efficient computation, using a two-dimensional matrix to track intermediate results and minimize redundant calculations.

The parallel version of the NW algorithm is similar to the serial version, with the primary difference occurring in the matrix filling step. In the sequential NW algorithm, the matrix is filled element by element, where each element depends on the previously three computed values from the matrix. In contrast, the parallel version leverages multiple cores of GPU to fill different parts of the matrix simultaneously, reducing the time complexity of this step. However, the initialization and traceback steps remain the same as in the traditional algorithm, as they do not require parallel computation. Thus, while the parallelized algorithm improves efficiency during the matrix filling process, the overall structure and logic of the NW algorithm remain unchanged.

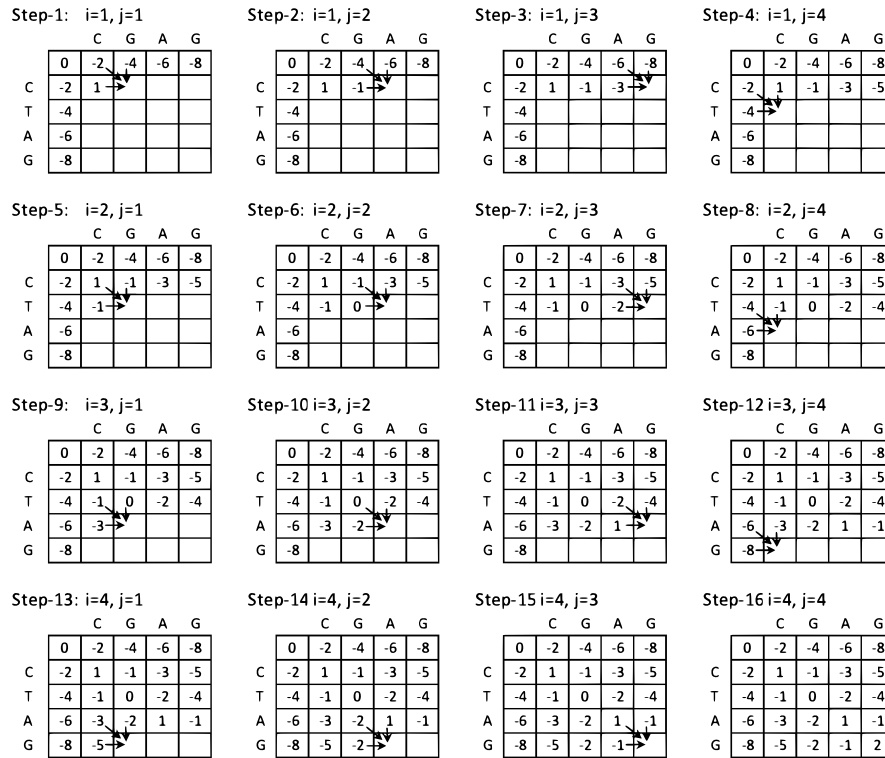


Figure 1. Serial Matrix Filling for NW Algorithm (CGAG vs. CTAG)

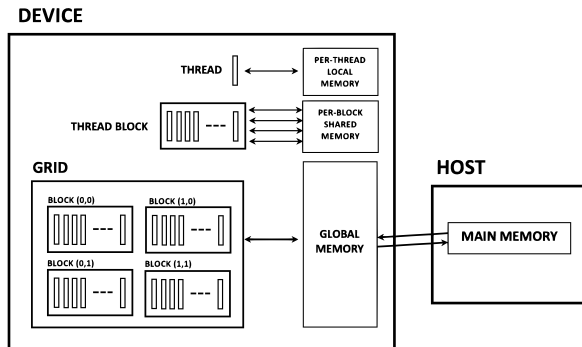


Figure 2. Host and Device Memory in CUDA Implementation

Figure 2 shows the block diagram of memory used in this experiment, illustrating the memory architecture for both serial and parallel execution. For serial execution, the alignment process is performed entirely on the host, utilizing the main memory of the CPU (host memory) to store and process the entire dataset sequentially. In parallel execution, the computation involves both the host and the GPU (device) [21]. The host uses its main memory to store the data, while the GPU leverages its global memory for large datasets accessible by all threads across different blocks, and shared memory for intermediate data within individual blocks to reduce memory access latency. The interaction between the host and the device, as well as their respective memory systems, is depicted in Figure 2.

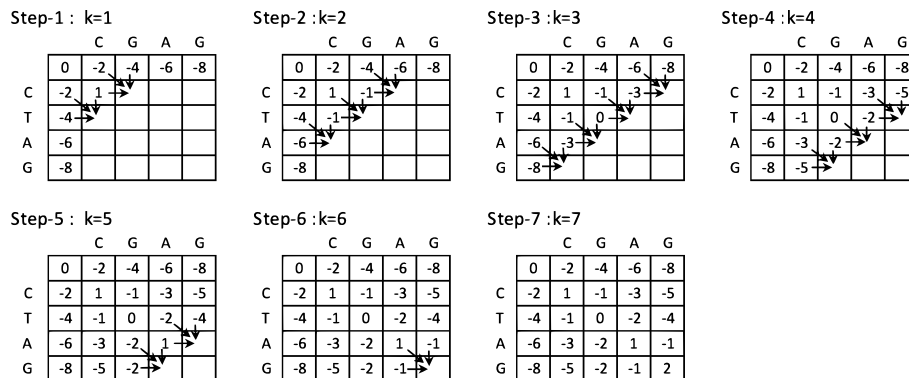


Figure 3. Parallel Matrix Filling Using GPU Cores: Each Matrix Cell on The Same Antidiagonal is Assigned to A Different GPU Thread for Concurrent Execution. Threads are Organized to Process Antidiagonals Sequentially While Maintaining Data Dependency Constraints.

Figure 3 illustrates how the GPU cores execute these parallel tasks, with each thread calculating different portions of the matrix simultaneously after the initial seed value has been established. The parallel version of the NW algorithm was implemented using CUDA C++ to leverage the computational power of the GPU for accelerating the sequence alignment process. In this parallel implementation, the most computationally intensive steps, particularly the filling of the dynamic programming matrix, were offloaded to the GPU. Each thread on the GPU was responsible for calculating individual matrix elements, allowing the algorithm to process multiple elements simultaneously, significantly reducing the overall computation time for large sequence pairs. The GPU's cores worked in parallel to compute the optimal alignment scores, handling tasks such as calculating matches, mismatches, and gap penalties across multiple positions in the sequences. In contrast, the serial version of the algorithm would require 16 steps to fill the matrix, whereas the parallel version reduces this to only 7 steps. Meanwhile, the CPU maintained control over the overall process, managing the initialization of the scoring matrix, the transfer of data between the CPU and GPU, and the orchestration of the final traceback phase. After the first element of the matrix was computed, the GPU was tasked with performing parallel computations for the remaining matrix elements.

2.4 Material

Following the development and implementation of both the serial and parallel versions of the NW algorithm, it was necessary to select biologically meaningful sequences for performance evaluation and validation of the approach. Accordingly, DNA sequences from the Neuroblastoma RAS Viral Oncogene Homolog (NRAS), which is implicated in multiple cancer types, and the Breast Cancer genes (BRCA1 and BRCA2), which are well-established markers for breast and ovarian cancer susceptibility, were utilized in this study. These biomarker sequences, along with *Saccharomyces cerevisiae* gene sequences, were sourced from the National Center for Biotechnology Information (NCBI) database [22]. They were selected for their relevance in bioinformatics applications and were used as input for the NW pairwise alignment algorithm. BRCA1 and BRCA2 genes were chosen due to their long coding regions and clinical importance as tumor suppressor genes, making them representative of realistic, complex alignment tasks in cancer genomics. *Saccharomyces cerevisiae* was included as a model organism to validate alignment scenarios for well-studied, shorter genomes.

Prior to inputting into the algorithm, the sequences underwent a pre-processing stage in which extraneous characters, such as non-genomic symbols, were removed to ensure the integrity of the nucleotide data. In this study, the sequences were trimmed to target lengths ranging from very short examples (e.g., 5 and 5 characters, or 8 and 10 characters) to larger cases up to 61440 characters. Input pairings were intentionally

designed to include both equal-length and varying-length pairs to reflect realistic biological cases and to evaluate the algorithm's performance under different alignment conditions. The trimming strategy involved selecting the first N characters from each sequence without random cutting, ensuring that the process was consistent and reproducible. Proper pre-processing is critical to eliminate noise that could affect alignment accuracy and to ensure that the computational experiments reflect realistic and reproducible conditions, thereby enhancing the reliability of the resulting performance comparisons.

2.5 Execution Time Measurement

The execution times for both the serial and parallel implementations of the NW algorithm were measured using C++'s built-in time library, which provides high-resolution timing functionality. Time measurements were recorded from the initiation of the alignment process to the final result, allowing for a comprehensive evaluation of the total computational time required for each implementation. The serial version, executed solely on the CPU, and the parallel version, utilizing GPU acceleration via CUDA, were compared to assess the performance improvement achieved by offloading computation to the GPU. Experiments were conducted across different sequence lengths to evaluate the scalability and efficiency of both implementations. Short sequences, consisting of only a few characters (several nucleotides), were tested to observe the baseline performance. Medium-length sequences, ranging from hundreds of nucleotide characters, were used to test the algorithm's handling of typical biological data. Finally, long sequences, up to 30,720s characters, or matrix dimension about 943,718,400 elements were aligned to assess the effectiveness of the parallel implementation in handling larger length of pair sequences. These experiments were designed to highlight the differences in execution time between the serial and parallel approaches, particularly in terms of scalability and the impact of GPU acceleration for large-scale sequence alignments.

Recent studies have also demonstrated the benefits of GPU-based computation in bioinformatics, particularly for large-scale sequence alignment and related tasks [10], [23], [24]. Incorporating such GPU acceleration strategies is therefore crucial not only for performance improvement but also for ensuring scalability in modern biological data analysis. The source code for the GPU-based Needleman-Wunsch implementation is available upon request and can be provided to support reproducibility.

3. Results and Discussions

After implementing both the serial and GPU-based versions of the Needleman-Wunsch algorithm, a series of computational experiments were conducted to evaluate their performance and scalability. This section presents the experimental findings, focusing on

alignment accuracy, execution time, and computational efficiency. The results are analyzed to highlight the performance improvement achieved through GPU acceleration compared to the traditional CPU-based implementation. Furthermore, the discussion emphasizes critical aspects such as the break-even point, speedup behavior, and scalability trends across varying sequence lengths.

3.1 Alignment Results

The alignment result of two nucleotide sequences, "CGAG" and "CTAG," was performed using the NW algorithm, as shown in Figure 4.

Traceback is initiated from the bottom-right cell of the dynamic programming matrix. When the reference and query nucleotides are identical, a diagonal move toward the top-left is performed. In the case of a mismatch, the traceback follows the direction corresponding to the maximum value among the left (horizontal), top (vertical), and diagonal (upper-left) neighboring cells. The reference sequence is aligned along the top axis, while the query sequence is aligned along the left axis of the matrix.

Alignment construction follows the traceback path starting from the bottom-right. A move from a higher to a lower score indicates a match, while a move from a lower to a higher score indicates a mismatch. Vertical moves correspond to insertions (gaps in the reference sequence), and horizontal moves correspond to deletions (gaps in the query sequence). Applying these rules ensures the correct generation of the aligned sequences.

Figure 4 illustrates an example of the traceback process when aligning the sequences CGAG (reference) and

CTAG (query). The traceback starts from the bottom-right cell of the matrix. The last nucleotides, G and G, match, prompting a diagonal move toward the top-left. Moving backward, A and A also match, leading to another diagonal move. At the next position, a mismatch between G (reference) and T (query) occurs, and the traceback follows the neighboring cell with the highest score, which in this case involves a vertical or horizontal move to introduce a gap. Finally, C and C match again through a diagonal move. By consistently applying these steps, the aligned sequences are reconstructed, as depicted in Figure 4.

		C	G	A	G		
		0	-2	-4	-6	-8	
C	-2	1	-1	-3	-5		C G A G
T	-4	-1	0	-2	-4		
A	-6	-3	-2	1	-1		C T A G
G	-8	-5	-2	-1	2		

Figure 4. Traceback Illustration

3.2 Break-even Point

After obtaining the sequence alignment results, the next logical step was to assess the efficiency of the methods used. To understand how much improvement was achieved by leveraging GPU acceleration, we conducted a comparative analysis between the GPU and CPU execution times across different matrix sizes. Table 1 presents the details of this performance comparison. For inputs with relatively short lengths, or when the number of matrix elements remains relatively small, the CPU's serial version of the algorithm remains feasible and efficient. However, for longer inputs, the serial algorithm becomes increasingly impractical due to significantly longer execution times, which can limit its usability for large-scale bioinformatics analyses.

Table 1. Execution time of Serial NW and Parallel NW

Length (characters)	Matrix Dimension (NxM characters)	NW-CUDA Execution time (ms)	NW-CPU Execution time (ms)	Speedup GPU vs CPU
10	25	0,055296	0,001052	0,019024884
18	80	0,055296	0,002915	0,052716291
30	200	0,070592	0,007735	0,109573323
64	1024	0,060416	0,028716	0,475304555
100	2500	0,111616	0,086595	0,775829630
120	3600	0,110528	0,099090	0,896514910
120	3600	0,107520	0,111121	1,033494234
150	5625	0,132096	0,159395	1,206660308
200	10000	0,236544	0,270137	1,142015862
1846	849528	2,628608	23,013838	8,755142646
2664	1774224	4,133888	44,439976	10,75016449
3892	3786916	7,959424	102,597039	12,89000800
4609	4993188	10,46835	140,113600	13,38449452
4814	4638024	12,271616	181,876082	14,82087461
20480	104857600	453,65863	3.196,940433	7,047017783
30720	209715200	1942,041626	6272,037568	3,229610264
30720	209715200	2086,812744	8285,666385	3,970488684
40960	419430400	3951,493164	13354,19976	3,379532548
61440	943718400	12858,04199	29805,70944	2,318059737

In this context, identifying the break-even point (BEP) between CPU and GPU execution times becomes crucial. The break-even point serves as a key threshold, indicating the minimum input size at which parallel

execution on the GPU begins to outperform serial execution on the CPU. Understanding this threshold not only guides the optimal selection of computational resources but also enhances the scalability and

efficiency of sequence alignment pipelines in real-world applications. From Table 1, it can be estimated that the break-even point lies between 100 and 120 characters. We modeled the execution times of CPU and GPU as quadratic Formulas 6 to 10 of the problem size x [17], [20].

$$y_{CPU}(x) = a_c x^2 + b_c x + c_c \quad (6)$$

$$y_{GPU}(x) = a_g x^2 + b_g x + c_g \quad (7)$$

y_{CPU} is the CPU execution time (milliseconds); y_{GPU} is the GPU execution time (milliseconds); x is the sequence length (number of nucleotides); a_c, b_c, c_c are the coefficients of the polynomial fit for CPU data; a_g, b_g, c_g are the coefficients of the polynomial fit for GPU data.

The break-even point is mathematically defined as the point at which the execution time for the GPU equals that of the CPU as shown in Formula 8. Therefore, the two polynomial equations are equated and solved to determine this intersection as shown in Formula 9.

$$y_{CPU}(x) = y_{GPU}(x) \quad (8)$$

which yields

$$(a_c - a_g)x^2 + (b_c - b_g)x + (c_c - c_g) = 0.$$

From the experiment we have

$$y_{GPU} = (5 * 10^{-6})x^2 - (0,0857)x + 89,206$$

$$y_{CPU} = (8 * 10^{-6})x^2 - (0,0070)x + 1,8881$$

$$(-3 * 10^{-6})x^2 - (0,0787)x + 87,3179 = 0 \quad (9)$$

We can obtain break-even point is 117 characters from Formula 10.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = 116,67 \approx 117 \quad (10)$$

In practice, shorter sequences below 117 characters (or 117 nucleotides) are often associated with functional elements such as primers, motifs, or non-coding RNAs like miRNAs, which require rapid alignment but with lower computational burden. In contrast, longer sequences above this threshold commonly represent exons, gene fragments, or full-length coding sequences, which benefit from more robust alignment accuracy despite higher computational cost.

3.3 Speedup

Another performance analysis of both GPU and CPU involves the calculation of speedup, which measures the relative improvement in execution time achieved by the GPU over the CPU. For small numbers of input characters, CPU-based computation often outperforms GPU-based computation due to the overhead associated with initializing the GPU environment, transferring data between the host and device, and launching kernels. These setup times become significant when the problem size is small, outweighing the benefits of parallel execution. However, as the size of the input sequences

increases, the GPU's ability to execute multiple operations simultaneously begins to dominate, leading to significant reductions in total computation time. Consequently, the speedup becomes more pronounced for larger matrix dimensions. The speedup values presented in the fourth column of Table 1 are determined using the following Formula 11, where speedup is defined as the ratio of CPU execution time to GPU execution time [25]. This metric provides a clear quantitative measure of the acceleration achieved through parallelization as shown in Figures 5, 6 and 7.

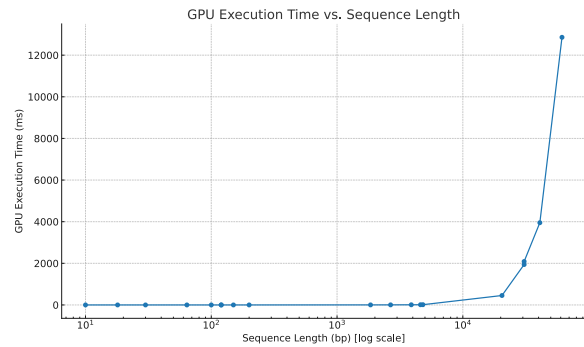


Figure 5. GPU Execution Time vs. Sequence Length. Execution Time Increases with Longer Sequence Lengths, Fitted with A Second-Degree Polynomial Trendline. Axes are Labeled in Milliseconds and Base Pairs

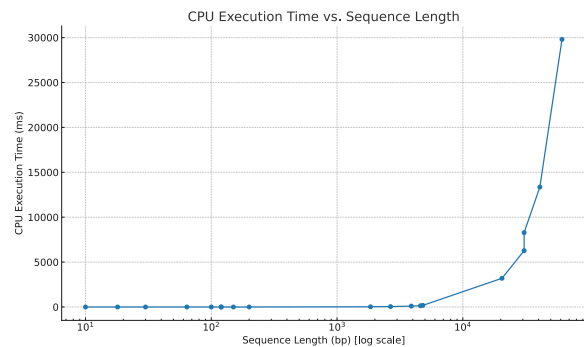


Figure 6. CPU Execution Time vs. Sequence Length

$$Speedup = \frac{y_{CPU}}{y_{GPU}} \quad (11)$$

y_{CPU} is the CPU execution time (milliseconds); y_{GPU} is the GPU execution time (milliseconds).

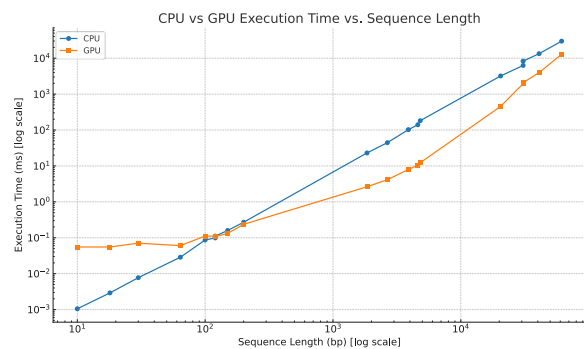


Figure 7. Execution Time of GPU and CPU Across Sequence Lengths

From Table 1, it is evident that the speedup values closely correlate with the break-even point discussed earlier. In other words, the CUDA implementation begins to outperform the CPU version significantly after this point is reached. However, the relationship is not strictly linear. As shown in Figure 8, the speedup continues to increase with input lengths up to approximately 4,800 nucleotides, reaching a peak of about 14.82 times. Beyond this length, the speedup gradually decreases. This trend is typical in GPU-accelerated computing, where initial increases in performance result from better utilization of parallel threads and GPU resources. However, after the peak, the performance gain tapers off due to factors such as global memory bottlenecks, thread divergence, and increased kernel execution overhead. Moreover, for very large sequences (e.g., over 20,000 nucleotides), the cost of memory access and synchronization begins to outweigh the benefits of parallelism. This indicates that optimal GPU performance lies within a specific range of input sizes, where computational parallelism is maximized and hardware constraints are minimally impactful.

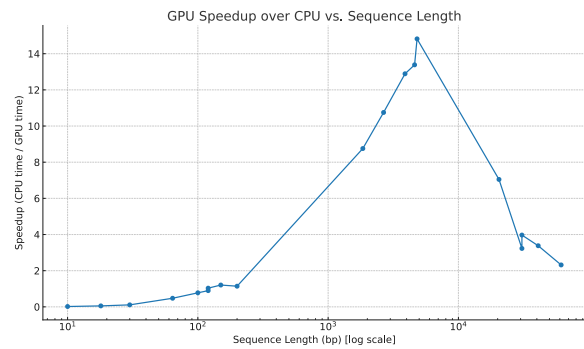


Figure 8. Speedup GPU Vs CPU Across Sequence Lengths: The Speedup Increases with Input Size and Peaks at Around 4,800 Characters, Reaching A Maximum Of 14.82×. Beyond This Point, Performance Declines Due to GPU Resource Limitations and Kernel Execution Overhead

We also calculate the log to facilitate the graphical representation becomes readable. Table 2 presents the Log10 of the matrix dimensions, GPU execution times, and CPU execution times. Figure 9 and Figure 10 show the log plots of this study.

Table 2. Logarithmic (Base 10) Values of Matrix Dimensions and Corresponding CPU and GPU Execution Times

Matrix Dimension (NxM characters)	Log10 (Matrix Dimension)	Log10 (GPU ms)	Log10 (CPU ms)
25	1,397940009	-1,257306284	-2,97798426
80	1,903089987	-1,257306284	-2,535361441
200	2,301029996	-1,151244514	-2,111539682
1024	3,010299957	-1,218848032	-1,541876055
2500	3,397940009	-0,952273545	-1,062507183
3600	3,556302501	-0,956527688	-1,003970172
3600	3,556302501	-0,968510744	-0,954202686
5625	3,750122527	-0,879110333	-0,797525306
10000	4,000000000	-0,626088063	-0,568415927
849528	5,929177698	0,419725825	1,361989052
1774224	6,249008450	0,616358706	1,647773816
3786916	6,578285672	0,900881640	2,011134827
4993188	6,698377918	1,019878317	2,146480292
4638024	6,666332992	1,088901757	2,259775590
104857600	8,020599913	2,656729177	3,504734544
209715200	8,321629909	3,288258534	3,797408651
209715200	8,321629909	3,319483480	3,918327443
419430400	8,622659905	3,596761235	4,125617868
943718400	8,974842423	4,109174840	4,474299463

The data presented in Table 2, the graph in Figure 9, and Figure 10, demonstrate the performance comparison between CPU and GPU for varying dimensions of matrix used in this experiment. As the matrix size increases, the execution times for both processors grow, but at different rates. Initially, for smaller matrices such as 50x50 or 2500, the CPU and GPU show relatively similar execution times, with the GPU slightly outperforming the CPU. However, as the matrix size increases beyond 100x100 or 10000, the disparity between the two processors becomes more significant.

The GPU's execution time grows at a slower rate than the CPU's, reflecting its ability to handle larger matrix operations more efficiently due to its parallel processing capabilities. In contrast, the CPU experiences a steeper increase in execution time, indicating its limitations in scaling with larger matrix dimensions. These trends

emphasize the GPU's superior scalability and performance for large-scale computations, making it the better choice for computationally demanding tasks involving large matrices.

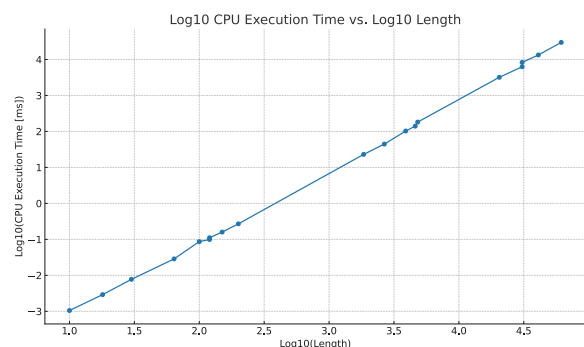


Figure 9. Log Plots of CPU Execution Time vs Matrix Dimensions

Complementary evidence is provided by Zeni et al. (2020), who implemented GPU-accelerated X-drop long-read alignment on a Tesla V100 HPC cluster and reported up to $30.7\times$ speed-ups over multi-threaded CPU baselines at scale [20]. Our work extends the context of GPU acceleration to commodity laptop hardware, offering practical insight in a low-resource environment.

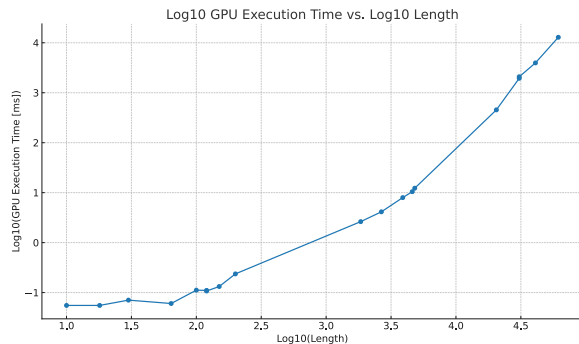


Figure 10. Log Plots of GPU Execution Time vs Matrix Dimensions

To improve computational efficiency on longer DNA sequences, future research will implement optimization based on tiling, which breaks the dynamic programming matrix into sub matrixes for more optimal parallel processing and memory usage.

4. Conclusions

To conclude the findings from the experimental analysis, several key observations can be summarized. The break-even point for this study is estimated to be around 116.67, or approximately 117 characters of input. It is observed that computations using the GPU exhibit a more polynomial quadratic growth compared to the CPU. The maximum speedup achieved by the GPU over the CPU in this case is 14.8 times, though this level of acceleration is not consistently reached across all scenarios. Moreover, the parallel version of this algorithm demonstrates a significant reduction in processing time compared to the traditional CPU-based serial implementation. Looking ahead, further performance analysis will be conducted, and the use of multiple GPUs for tasks such as multiple sequence alignment will be explored to further enhance computational efficiency. Overall, the results underscore the scalability potential of GPU-based approaches for sequence alignment tasks, suggesting that high-performance bioinformatics is increasingly within reach even for research groups operating outside elite computing centers. This highlights an encouraging future where powerful genomic analyses can be performed cost-effectively and more widely, accelerating discoveries in many areas of life sciences. Future work could extend this approach to multiple sequence alignment or integrate it into clinical genomic pipelines that require high-throughput, cost-effective sequence analysis.

Acknowledgements

This work was supported by the Tropical Biopharmaca Research Center, IPB University, Bogor, Indonesia and the Postgraduate Research - Doctoral Dissertation Research Grant (PPS-PDD) under Contract Number: 0070/C3/AL.04/2025, dated May 23, 2025. The authors declare no conflict of interest.

References

- [1] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *J Mol Biol*, vol. 48, no. 3, pp. 443–453, 1970. [https://doi.org/10.1016/0022-2836\(70\)90057-4](https://doi.org/10.1016/0022-2836(70)90057-4)
- [2] D. Stathis et al., "eBrainII: a 3 kW Realtime Custom 3D DRAM Integrated ASIC Implementation of a Biologically Plausible Model of a Human Scale Cortex," *J Signal Process Syst*, vol. 92, no. 11, pp. 1323–1343, 2020. <https://doi.org/10.1007/s11265-020-01562-x>
- [3] S. Ullah, S. Rehman, M. Shafique, and A. Kumar, "High-Performance Accurate and Approximate Multipliers for FPGA-based Hardware Accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021. <https://doi.org/10.1109/TCAD.2021.3056337>
- [4] M. Alser, T. Shahroodi, J. Gómez-Luna, C. Alkan, and O. Mutlu, "SneakySnake: A fast and accurate universal genome pre-alignment filter for CPUs, GPUs and FPGAs," *Bioinformatics*, vol. 36, no. 22–23, pp. 5282–5290, 2020. <https://doi.org/10.1093/bioinformatics/btaa1015>
- [5] S. Salamat and T. Rosing, "FPGA Acceleration of Sequence Alignment: A Survey," 2020. <http://arxiv.org/abs/2002.02394>
- [6] A. E. E. D. Rashed, "Accelerating DNA pairwise sequence alignment using FPGA and a customized convolutional neural network," *Computers and Electrical Engineering*, vol. 92, 2021. <https://doi.org/10.1016/j.compeleceng.2021.107112>
- [7] W. Chen, C. Yao, Y. Guo, Y. Wang, and Z. Xue, "pmTM-align: scalable pairwise and multiple structure alignment with Apache Spark and OpenMP," *BMC Bioinformatics*, vol. 21, no. 1, p. 426, Sep. 2020. <https://doi.org/10.1186/s12859-020-03757-2>
- [8] M. Malik, S. Malhotra, and N. Prasanth, "Time Improvement of Smith-Waterman Algorithm Using OpenMP and SIMD," 2020. https://doi.org/10.1007/978-981-15-4451-4_54
- [9] A. R. Akbar, H. Sukoco, and W. A. Kusuma, "Comparison of data partitioning schema of parallel pairwise alignment on shared memory system," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 13, no. 2, pp. 694–702, 2015. <https://doi.org/10.12928/TELKOMNIKA.v13i2.1415>
- [10] Q. Aguado-Puig et al., "WFA-GPU: Gap-affine pairwise alignment using GPUs," *bioRxiv*, 2022. <https://www.embase.com/search/results?subaction=viewrecord&id=L2018217170&from=export%0Ahttp://dx.doi.org/10.1101/2022.04.18.488374>
- [11] L. N. J. Gavarraju, K. K. Pavan, A. D. P. Swaroop, and H. C. Narne, "Gpu based parallel tlbo and parallel jaya for multiple sequence alignment using mapreduce (Gpu-ptlbo & Gpu-Pjaya)," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 7, pp. 250–259, 2019, [Online]. <https://www.scopus.com/inward/record.uri?partnerID=HzOxMe3b&scp=85067857896&origin=inward>
- [12] S. B. Iryanto, W. A. Kusuma, R. Sadikin, and I. W. A. Swardiana, "GPU-accelerated protein sequence alignment for Jamu prediction," 2017. <https://doi.org/10.1109/IC3INA.2017.8251754>
- [13] V. Gancheva and I. Georgiev, "Multithreaded parallel sequence alignment based on needleman-wunsch Algorithm," in *Proceedings - 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering, BIBE 2019*, IEEE, Oct. 2019, pp. 165–169. <https://doi.org/10.1109/BIBE.2019.00037>

- [14] S. Bhattacharya, R. Roche, B. Moussad, and D. Bhattacharya, "DisCovER: distance- and orientation-based covariational threading for weakly homologous proteins," *Proteins: Structure, Function and Bioinformatics*, vol. 90, no. 2, pp. 579–588, 2022. <https://doi.org/10.1002/prot.26254>
- [15] S. R. Perumalla, "Needleman-wunsch algorithm using multi-threading approach," 2020. https://doi.org/10.1007/978-981-15-1480-7_25
- [16] B. Kieu-Do-Nguyen, C. Pham-Quoc, and C.-K. Pham, "Heterogeneous Hardware-assisted Parallel Processing for BWA-MEM DNA Alignment," in *2020 RIVF International Conference on Computing and Communication Technologies (RIVF)*, IEEE, Oct. 2020, pp. 1–7. <https://doi.org/10.1109/RIVF48685.2020.9140768>
- [17] L. Alawneh, M. A. Shehab, M. Al-Ayyoub, Y. Jararweh, and Z. A. Al-Sharif, "A scalable multiple pairwise protein sequence alignment acceleration using hybrid CPU–GPU approach," *Cluster Comput*, vol. 23, no. 4, pp. 2677–2688, Dec. 2020. <https://doi.org/10.1007/s10586-019-03035-8>
- [18] X. Gao et al., "CBEx: A Hybrid Approach for Cancer Biomarker Extraction," in *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, IEEE, Dec. 2020, pp. 2958–2958. <https://doi.org/10.1109/BIBM49941.2020.9313403>
- [19] D. Elettronica, A. Zeni, S. Onken, D. Elettronica, and M. Domenico, "Leveraging Difference Recurrence Relations for High-Performance GPU Genome Alignment". <https://doi.org/10.1145/3656019.3676894>
- [20] A. Zeni et al., "LOGAN: High-Performance GPU-Based X-Drop Long-Read Alignment," *Proceedings - 2020 IEEE 34th International Parallel and Distributed Processing Symposium, IPDPS 2020*, pp. 462–471, 2020. <https://doi.org/10.1109/IPDPS47924.2020.00055>
- [21] T. Benefits and U. Gpus, "CUDA C ++ Programming Guide Changes from Version 12.4". <https://docs.nvidia.com/cuda/cuda-c-programming-guide/>
- [22] "National Center for Biotechnology Information (NCBI)," National Center for Biotechnology Information (NCBI). <https://www.ncbi.nlm.nih.gov/>
- [23] B. Schmidt, F. Kallenborn, A. Chacon, and C. Hundt, "CUDASW++4.0: Ultra-fast GPU-based Smith-Waterman Protein Sequence Database Search," *bioRxiv*, vol. 25, no. 1, p. 2023.10.09.561526, Nov. 2023. <https://doi.org/10.1186/s12859-024-05965-6>
- [24] C. Zhong and H. Sun, "Parallel algorithm for sensitive sequence recognition from long-read genome data with high error rate," *Tongxin Xuebao/Journal on Communications*, vol. 44, no. 2, pp. 160–171, 2023. <https://doi.org/10.11959/j.issn.1000-436x.2023009>
- [25] D. AnySeq, "High-speed alignment on GPUs (AnySeq/GPU)," *Comp Methods*, 2022. <https://doi.org/10.48550/arXiv.2205.07610>