Periodico di Mineralogia
ISSN: 0369-8963 Volume 94, No. 6, 2025

Enhancing DNA Sequence Alignment Using Parallel CPU–GPU Computing with Unified Memory

Mohammed Fadhl Abdullah[1,2], **Khaled Hassan Mahdi Balhaf**[1,3]

[1] *College of Engineering and Computing, University of Science and Technology, Aden, Yemen*

[2]*m.albadwi@ust.edu,* [3]*khaledbalhaf2021@gmail.com*

Abstract: *Sequence alignment is a crucial procedure in bioinformatics, facilitating the comparison and study of DNA and protein sequences for evolutionary and functional investigations. The growing amount and complexity of genomic datasets pose considerable computational problems for conventional CPU-based alignment techniques. This study presents an improved parallel computing framework that combines Central Processing Units (CPUs) and Graphics Processing Units (GPUs) with NVIDIA's Unified Memory (UM) architecture to enhance the efficiency of DNA sequence alignment operations. The suggested method utilizes the extensive parallelism of GPUs while ensuring effective memory management via UM, thus reducing data transfer overhead between the host and the device. Experimental results indicate a significant enhancement in execution speed and computational efficiency relative to traditional sequential methods. The results validate that hybrid CPU-GPU processing, augmented by Unified Memory, offers a scalable and high-performance solution for contemporary bioinformatics applications necessitating intense sequence analysis.*

Keywords: Parallel Computing, Unified Memory, Sequence Alignment, Dynamic Programming, CUDA Optimization.

1. INTRODUCTION

Recently, bioinformatics has emerged as a prominent subject within computer science, providing extensive insights into biology and human-related information. The primary challenge in bioinformatics is computational biology, which necessitates the application of mathematics, computer science, and engineering to address this issue [1] . The researchers engage in numerous bioinformatics domains, including motif recognition [2] and sequence alignments, due to the significant relevance of this discipline today, with many scholars from diverse fields focusing on motif discovery and next-generation sequencing [3].

Due to the large and complex nature of biology data, many biologists have turned to modern technologies and computer science to analyze and understand it [4]. Bioinformatics focuses on the development of computation processes and software tools to achieve this goal. Many studies use bioinformatics for broad goals, whereas others use sequence alignments. Sequence alignments compare or assess string similarity. Text comparison has been employed in several amino acid and protein research. Three sequence types are seen in biological sequences [5].

a) DNA string consists of four letters (A, C, G, and T).
b) RNA string has four letters (A, C, G, and U).
c) The protein string has the following letters: A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V.

Biology data research uses high-quality algorithms to ensure comparison accuracy and proper results. Thus, several researchers adopted these sequence alignment techniques to improve comparison and similarity operations. They improved sequence alignment execution time with high-performance computing. Numerous experts in the domain of textual comparison have developed several mathematical matrices for quantifying similarities, referred to as edit distance methods. For instance, the Levenshtein edit distance algorithm [6], the Needleman-Wunsch algorithm (NW) [7], and the Smith-Waterman algorithm (SW) [8]. These algorithms are the most prevalent for computing sequence alignment for textual or biological sequences (DNA, RNA, and proteins).

The accuracy of sequence alignment algorithms has been the subject of much research in the past 30 years, leading to numerous updates that have helped refine similarity and difference calculations [9]. Contrarily, researchers aimed to enhance the performance of sequence alignment algorithms by utilizing newly developed technology.

Finding suitable methods to analyze big data, like cloud computing, high-performance computing, and parallel programming, has recently become vital with the emergence of big data. Our study's goals are to (1) increase the speed of sequence alignment algorithms while keeping similarity calculations accurate and (2) achieve the same results more quickly by making use of current technologies associated with Graphics Processing Units (GPUs) that enable parallel programming. This document comprises the following sections: The second section will address the background of the algorithms and contemporary technology, followed by a review of prior research in this domain. Subsequently, we will outline the methodology employed in this study, present and analyze the results, and conclude with a discussion on future work.

2. BACKGROUND AND TECHNICAL FOUNDATIONS

A. Dynamic Programming Algorithms for Sequence Alignment

An approach to dynamic programming that simplifies large problems by breaking them down into smaller ones, solving each one separately, and then integrating the results to form a final solution. The ideal alignment solution is provided by the dynamic programming approach for sequence alignment, although it takes a long time to execute for very large sequences. To take use of new hardware and technology in a variety of computer activities, researchers had to resort to employing alternative approaches for aligning sequences. Here we'll take a look at the inner workings of three different dynamic programming methods commonly employed for sequence alignments.

The Levenshtein technique calculates the edit distance between two sequences to determine their resemblance, utilizing three cells in the matrix: left, upper, and upper left, to derive the minimal value [6]. The Levenshtein algorithm operates as demonstrated in Equation 1, and Equation 2 is employed to compute the enhanced iteration of the method.

$$H[i.j] = \begin{cases} \min(i-1.j-1) + \ score \\ \min(i.j-1)+1 \\ \min(i-1.j)+1 \end{cases} \quad \dots\dots\dots\dots\dots (1)$$

$$H[i.j] = \begin{cases} \begin{cases} \min(i-1.j-1) + \ score \\ \min(i.j-1)+1 \\ \min(i-1.j)+1 \\ \min(i-2.j-2)+1 \end{cases} & if(i.j > 1 \ and \ Ai = Bj-1 \ and \ Ai-1 = \ Bj) \\ \begin{cases} \min(i-1.j-1) + \ score \\ \min(i.j-1)+1 \\ \min(i-1.j)+1 \end{cases} & otherwise \end{cases} \quad \dots\dots (2)$$

An intelligent algorithm minimizes the extensive array of possibilities that must be evaluated, while still ensuring the discovery of the ideal answer. The Needleman-Wunsch (NW) algorithm is a dynamic programming technique that employs a divide-and-conquer approach. The NW algorithm decomposes the sequence alignment problem into smaller sub-problems, solving each individually and utilizing their solutions to generate an optimal resolution for the original problem [7].

$$H[i.j] = \begin{cases} \max(i-1.j-1) + \ score \\ \max(i.j-1)+Gap \\ max(i-1.j)+Gap \end{cases} \quad \dots\dots\dots\dots\dots (3)$$

In 1981, T. F. Smith and M. S. Waterman [8] created a method to calculate the similarity of sequences using the Needleman-Wunsch algorithm. The S-W Algorithm is a dynamic programming method that calculates the similarity of sequences of any length and position within any sequence, and assesses the feasibility of achieving an optimal alignment. The Smith-Waterman (SW) alignment is referred to as a local alignment, as defined by Equation 4.

$$H[i.j] = \begin{cases} \max(i-1.j-1) + \ score \\ \max(i.j-1) + Gap \\ \max(i-1.j) + Gap \\ \max(\,0\,) \end{cases} \quad \dots\dots\dots\dots\dots (4)$$

B. Parallel Computing and GPU Architectures

Parallel computing is a computational methodology that simultaneously executes several instructions and can decompose complex problems into smaller, concurrently solvable components [10]. This computational approach reduces the complexity and execution time of numerous algorithms. There exist four categories of parallel computing: Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), and Multiple Instruction Multiple Data (MIMD). Our study concentrates on addressing issues involving extensive data that require uniform processing, hence we employed SIMD parallel computing.

In numerous algorithms, various functions can manipulate data concurrently without impacting other operations or data [11]. Numerous manufacturers, including Intel and AMD, create new hardware that supports parallel computing. Develop new iterations of Control Process Units (CPUs) including multi-core architectures to enhance the performance of contemporary systems; for instance, dual-core processors exhibit a 60% increase in speed compared to single-core counterparts [12].

Graphics Processing Unit (GPU) devices are now supported by NVIDIA. This can work with SIMD because it generates thousands of threads, each of which can process a separate data item using the same command [13]. The GPU and CPU have varying numbers of cores, as seen in Figure 1. By utilizing NVIDIA GPU devices, developers and programmers are able to take advantage of the parallel computing environment known as Compute Unified Device Architecture (CUDA). As illustrated in Figure 2, CUDA threads are structured as sets called warps. A single block is made up of a set of warps, and several blocks can be placed in a single grid [11].

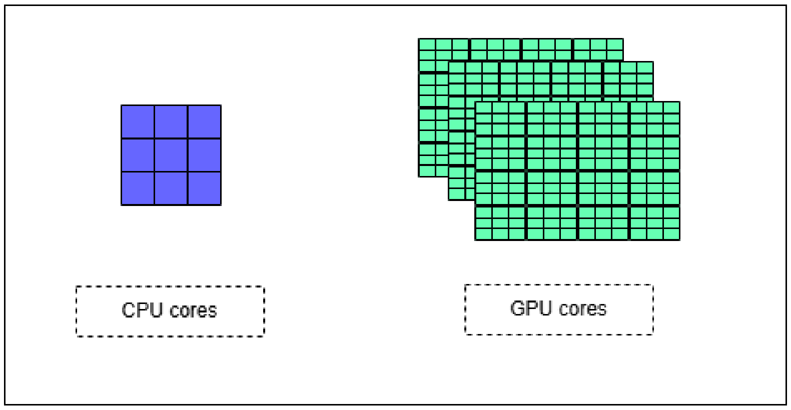


Figure 1 shows the difference number of cores between the GPU and CPU

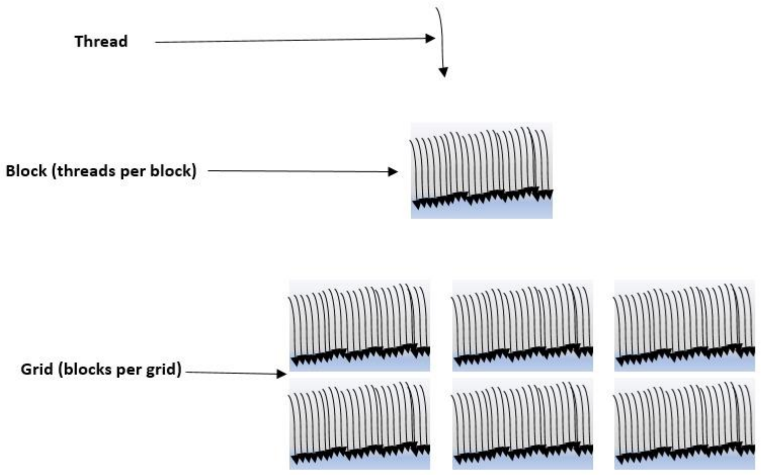


Figure 2 Organization threads in CUDA

However, parallel computing faces various obstacles, including memory management, data transfer (Host to Device or Device to Host), and GPU data type. No double data type, float data type size, etc. We measured the Machine epsilon of our hardware and determined that float in GPU float = 1.000001E-37 and on CPU = 1.401298E-42). For this reason, programmers must transfer double and float data carefully. In addition, GPUs are not always better than CPUs. Data transfer delays (H to D or D to H) can take longer than CPU execution [14]. Some data-dependent functions, including finding the matrix's maximum and minimum values, cannot be done in parallel because they must synchronize threads. Thus, developers must use a CPU-GPU hybrid platform [15]. Recently, several academics have used GPUs to accelerate image processing, bioinformatics, and other computer science applications. GPU hardware is also developed constantly. Integrated memory after this advancement.

In CUDA toolkit 6, invidia supports new technology (Unified memory) to save time spent on CudaMemCpy() and enable programmers write code clearly [16]. Unified memory is shared across the host (CPU) and device (GPU) and available for both CPU and GPU. Managed memory is CPU memory while the thread is executed in the host and GPU memory when it is executed in the GPU [17]. This approach gives CUDA programmers a new perspective on memory. Before unified memory, the programmer must utilize cudaMalloc(), cudaMemCpy(), and cudafree(). First, cudaMalloc allocated GPU memory, and then cudaMemCpy transferred data between host and device. Finally, cudafree() deallocates memory, but when using UM, the programmer needs two functions: cudaMallocManaged() to allocate memory and cudafree() to deallocate memory. To obtain high-speed computing for string matching algorithms, we suggest utilizing new approaches like UM for parallel programming. We then compare the algorithms' performance outcomes using CPU, GPU, and UM GPU.

3. RELATED WORK ON PARALLEL SEQUENCE ALIGNMENT

The primary challenge of sequence alignment is the computational alignment of lengthy and intricate biological sequences. This issue likely originated in 1966 with the concept of edit distance between two strings [18]. Edit distance quantifies the similarity between two strings by three operations: insertions, deletions, and replacements. Because dynamic programming algorithms are computationally expensive and dependent on sequence length, they are often referred to as quadratic algorithms. The advent of big data and next-generation sequencing (NGS) has significantly increased their cost [19]. It is known that algorithms that find the ideal solution after investigating all conceivable possibilities are expensive and time-consuming. Thus, several researchers have lowered execution time using parallel programming and innovative technologies like [20] [21]. In this research, we will employ the same strategy to speed up dynamic programming algorithms for sequence alignments and focus on three popular DNA sequence alignment algorithms: Wagner-Fischer, Needleman-Wunsch, and Smith-Waterman.

Aligning strings and assessing string similarity are common challenges solved by dynamic programming. Distributing a huge problem into subproblems and solving them progressively is a key principle of dynamic programming [22]. Two of the most typical difficulties of dynamic programming techniques are memory fullness and execution time. Thus, many uses greedy algorithms and other methods.

Bioinformatics challenges including sequence alignment, motif identification, RNA structure prediction, and protein-DNA interaction are solved via dynamic programming [22]. Researchers compared DNA and protein sequences using dynamic programming methods. A compilation of prior research is shown in Table 1.

Table1: Different techniques applied by the researchers

Author	Ref.	Technique	Year	Speedup
Bani Baker and et al	[23]	parallel computing to speedup NW algorithm.	2024	4X
Makino and et al	[24]	parallel CPU OpenMP to speedup SW algorithm	2024	1.9X
Balhaf and et al	[25]	Used GPUs parallel implementation to accelerate edit distance algorithm.	2016	11X
Al-Hussien and et al	[26]	Parallel computing to speedup NW pairwise sequencing using CPU threads.	2018	4X
Kurt and et al	[22]	Used GPUs parallel implementation to accelerate Needleman Wunch algorithm.	2022	17X
Schmidt and et al	[27]	(GPU) CUDA parallel implementation to accelerate Smith-Waterman algorithm.	2024	7X
Fakirah and et al	[28]	CUDA parallel implementation to accelerate Needleman-Wunsch global alignment.	2015	4X
Bahig and et al	[29]	parallel CPU OpenMP implementation to accelerate Smith-Waterman algorithm.	2024	3X

4. PROPOSED METHODOLOGY

The first of our four parts deals with sequential implementation using the central processing unit. Second, in order to implement OpenMP in parallel, we used a CPU. Additionally, we proceeded by running CUDA in parallel on a graphics processing unit (GPU). Unified Memory also allows us to take use of the unique design of modern GPUs.

a) Sequential CPU Implementation

We perform sequential operations within the CPU. Algorithm 1 delineates our CPU implementation, and we utilized Equation 5 to convert the 2D matrix into a 1D array. Equations 6 and 7 are utilized to convert the one-dimensional matrix into a two-dimensional matrix. The fundamental concern of dynamic algorithms is their dependence on data, as demonstrated in Figure 6. Calculate the H(I,j) cell utilizing the left cell, the upper cell, and the diagonal upper-left cell.

$$Index = Row * n + column \text{ ……… } (5)$$

$$Row = \frac{index}{n} \text{ …….. } (6)$$

$$Column = \text{index} \,\%\, \text{n} \text{ …….. } (7)$$

Algorithm 1 : CPU Sequential Implementation

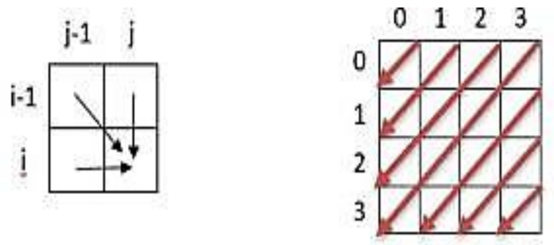
```
1:  $i \leftarrow 0$ , $j \leftarrow 0$
2:  while ($i < n - 1$) do
3:          $H(i, 0) = -i$
4:  end while
5:  while ($j < n - 1$) do
6:          $H(0, j) = -j$
7:  end while
8:  for $i = 0 \rightarrow n - 1$ do
9:          for $j = 0 \rightarrow n - 1$ do
10:             if $Ai - 1 == Bj - 1$ then
11:                 $Score = 1$
12:             else
13:                 $Score = -1$
14:             end if
15:     Calc ulate$Hi, j$ using equations 1 and 2 for the Levenshtein algorithm
16:     Calculate$Hi, j$ using equation 3 for the Needleman-Wunsch algorithm
17:     Calculate$Hi, j$ using equation 4 for the Smith-Waterman algorithm
18:         end for
19: end for
```

b) Parallel CPU Implementation Using OpenMP

The principal challenge in utilizing parallel computing with single-instruction multiple-data (SIMD) is data dependency. To address this issue, we utilized a diagonal approach in calculating the matrix of the sequence alignment algorithms displayed in figure 3. In our second way, we utilized an OpenMP strategy to reduce the extended execution time, especially when the chain size is considerably huge. The measurable diameter concurrently increases. This notion is termed data independence. In concurrent OpenMP programming, we employed all available CPU resources, particularly 8 threads, according to our hardware specifications. The CPU computes the quantity of elements to be processed concurrently for the current iteration and adjusts the memory block sizes. The block size is determined using the slide ID through the subsequent by equations 8 and 9.

$$blocksize = slideID - 2 * Z + 1 \text{ ……… } (8)$$

$$H[i.j] = \begin{cases} 0 & if\ slideID < n \\ slideID - n + 1 & otherwise \end{cases} \text{ …… } (9)$$



(a) (b)

Figure 3: Address dependency problem. (a) data dependency problem. (b) diagonal technique in matrix

Algorithm 2: CPU OpenMP Implementation

```
1:  i ← 0 , j ← 0
2:  while (i < n − 1) do
3:       H( i , 0 ) = 0
4:  end while
5:  while (j < n − 1) do
6:       H( 0 , j ) =0
7:  end while
8:  for slide = 0 → n ∗ 2 − 1 do
9:       if slide < n then
10:      z = 0
11:      else
12:      Z = Slide − n + 1
13:      end if
14:      blocksize = slide − 2 ∗ Z + 1
15:    #pragma OpenMP Calculate Hi, j equations 1 and 2 for the Levenshtein algorithm
16:    #pragma OpenMP Calculate Hi, j equation 3 for the Needleman-Wunsch algorithm
17:  #pragma OpenMP Calculate Hi, j equation 4 for the Smith-Waterman algorithm
 16: end for
```

c) GPU-Based Implementation Using CUDA

This section will employ a GPU as the hardware device because of its multitude of cores relative to CPU cores. Figure 1 depicts the architecture of the CPU and GPU. The Compute Unified Device Architecture (CUDA) is a parallel computing architecture that allows developers to utilize NVIDIA GPU devices. In CUDA, threads are organized into warps, numerous warps create a block, and several blocks constitute a grid.

The principal obstacle in employing parallel computing for single structure numerous data is data dependency. To address this issue, we utilized an antidiagonal method in the computation of the matrix for the sequence alignment algorithms. Figure 6 demonstrates that. Additionally, algorithm3 and algorithm4 illustrate our parallel implementation on the GPU employing CUDA programming tools.

Algorithm 3: GPU parallel Implementation

```
1:  i ← 0 ,  j ← 0
2:  while (i < n − 1) do
3:       H( i , 0 ) = - i
4:  end while
5:  while (j < n − 1) do
6:       H(0 , j ) = - j
7:  end while
8:  for slide = 0 → n ∗ 2 − 1 do
9:       if slide < n then
10:      z = 0
11:      else
12:      Z = Slide − n + 1
13:      end if
14:      blocksize = slide − 2 ∗ Z + 1
15:      CUDA Sequence alignment algorithm <<< blocksize, 256 >>>
16:  end for
```

Algorithm 4: CUDA Kernel algorithm

```
1:  Calculate thread ID
2:  if Z <= 0 then
3:       startindex = slide
4:  else
5:       startindex = increment ∗ Z + slide
6:  end if
7:  Grid = startindex + (ID ∗ increment)
8:  row = equation 6
9:  column = equation 7
10:  index = equation 5
11:   Calculate Hi, j using equations 1 and 2 for the Levenshtein algorithm
12:  Calculate Hi, j using equation 3 for the Needleman-Wunsch algorithm
13:  Calculate Hi, j using equation 4 for the Smith-Waterman algorithm
```

d) GPU Implementation with CUDA Unified Memory

With the introduction of Unified Memory in CUDA Toolkit 6, NVIDIA hopes to reduce the amount of time the cudaMemcpy() function takes and make programming easier and more understandable. In a Unified Memory setup, the host CPU and the device GPU share memory. When working on a host computer, managed memory is known as CPU memory; when running on a device, it is known as GPU memory. A new way of looking at memory in CUDA code is given to the developer by this method. The three procedures cudaMalloc(), cudaMemcpy(), and cudaFree() were used by the programmer before unified memory was implemented . The first step was to use cudaMalloc to allocate GPU memory, and then cudaMemcpy to transfer data from the host to the device. utilize cudaFree() to deallocate memory. However, when using unified memory (UM), the programmer needs to utilize two functions: cudaMallocManaged() to allocate memory and cudaFree() to deallocate it. That is seen in Figure 4.

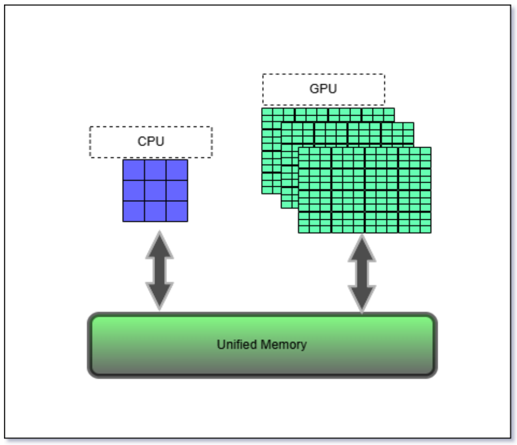


Figure 4 CUDA Unified Memory

5. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

This section presents and discusses the results achieved from the acceleration of three algorithms: the Levenshtein algorithm, the Needleman-Wunsch method, and the Smith-Waterman algorithm. This section delineates the results of our concurrent implementations. The aim is to determine the extent of speedup for scalable data sizes. DNA sequences varying in length from 15 KB to a maximum of 45 KB. Five unique tests are conducted for each size, and the averages are recorded. The acceleration is ascertained utilizing Equation 10. Experiments demonstrate that dynamic method computing is affected by the increase in sequence sizes during the sequential execution of sequence alignment methods. Thus, parallel computing with OpenMP reduces execution time, especially when employing several CPU cores. The following table presents the appropriate number of cores compatible with the device utilized in this study.

$$Speedup = \frac{CPU\ time}{GPU\ time} \qquad (10)$$

Table 2: shows execution time for different number of OpenMP threads.

OpenMP			Sequence size= 8KB		
# of threads	**2**	**4**	**8**	**16**	**32**
Time execution	**1.165**	**0.708**	**0.604**	**0.799**	**0.943**

Figure 5 illustrates the outcomes of our investigation. The efficacy of the CPU implementation declines with increasing sequence length. In contrast, the GPU does not exert an effect. Furthermore, Figure 6 depicts the duration of data transmission between the host (CPU) and the device (GPU). We utilized UM to reduce execution time. Figure 5 illustrates our results on the execution durations of CPU, GPU, and GPU with Unified Memory (UM). Figure 7 demonstrates the improvements attained through the use of GPU and GPU with UM for the Smith-Waterman algorithm, and, Figure 8 and Figure 9 show the execution time and performance improvements as results of Needleman-Wunch. Also. Figure 10 and 11 show the results of Wagner-Fischer algorithm.

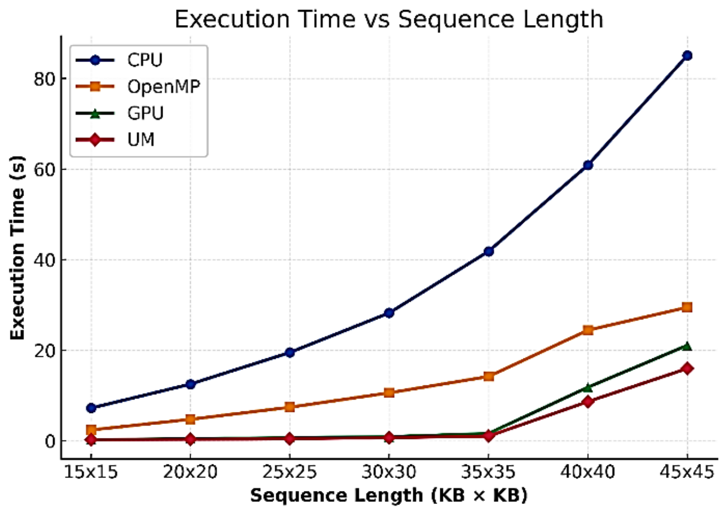


Figure 5 Execution time for Smith-Waterman algorithm

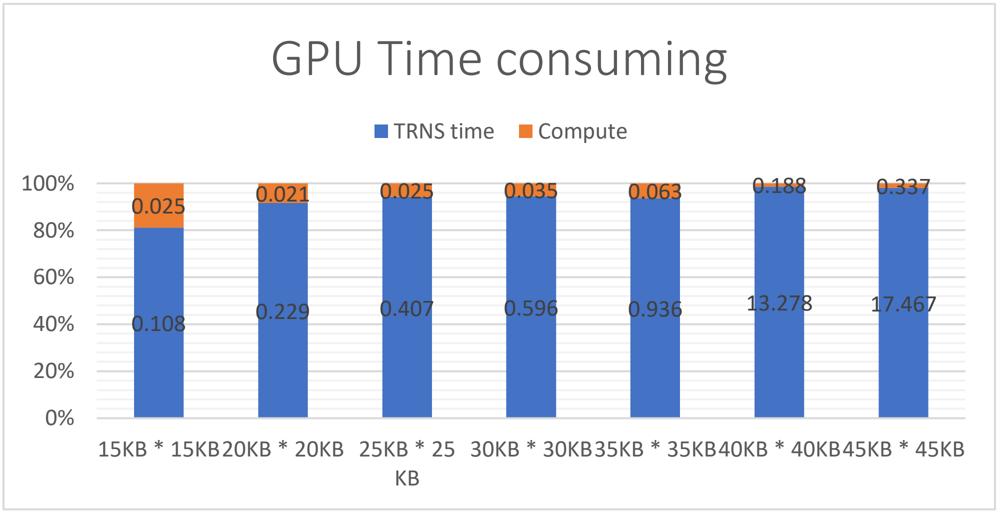


Figure 6: Time of Data Transfer and time of computation

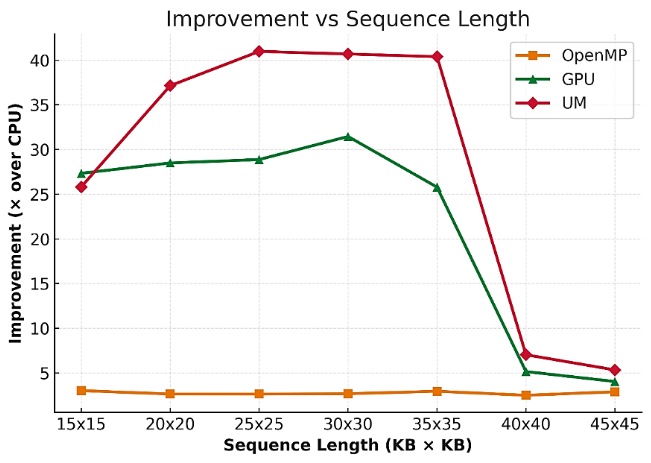


Figure 7: Improvement Compared to CPU Implementation for Smith-Waterman

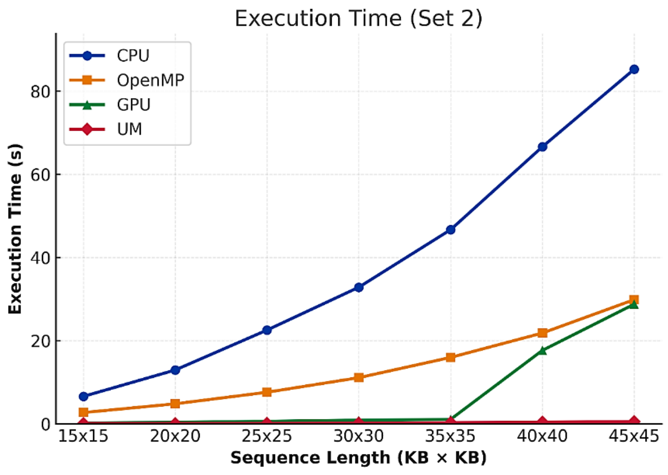


Figure 8: Execution time for Needleman-Wunch algorithm

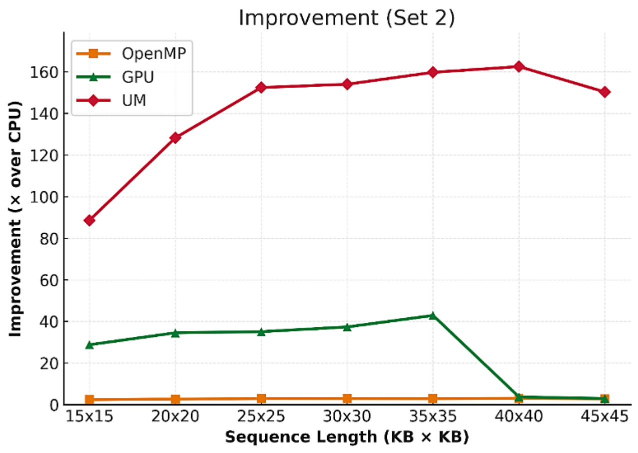


Figure 9: Improvement Compared to CPU Implementation for Needleman-Wunch

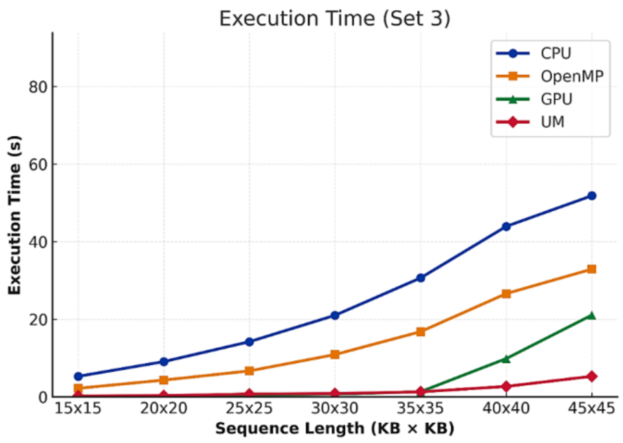


Figure 8: Execution time for Wagner-Fischer algorithm

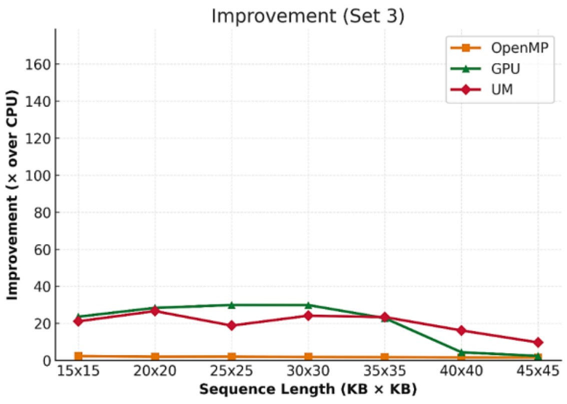


Figure 11: Improvement Compared to CPU Implementation for Wagner-Fischer algorithm.

The results indicate that memory was influenced by both the size of the graphics card memory and the larger chain size, as well as the unified memory space, which is approximately 8 GB in this device, whereas the memory size of the graphics processing units (GPUs) is only 6 GB, significantly affecting the percentage of improvements.

6. CONCLUSIONS AND FUTURE WORK

Bioinformatics is a discipline intrinsically linked to people, with DNA sequence alignment serving as a pivotal topic within the field. This work employed three established techniques for calculating the similarity and alignment of DNA sequences: the Needleman-Wunsch algorithm, the Smith-Waterman algorithm, and the Wagner-Fischer algorithm. The growing sophistication of technology has significantly enhanced our advantages, especially in computer hardware, characterized by an increasing number of cores in CPUs, GPUs, and shared memory. The CUDA programming language was employed as a parallel programming tool to enhance the computation of the previously discussed methods.

The outcomes were extraordinary, with rates escalating up to three times when using parallel programming, utilizing the cores of contemporary CPUs. This value increased by almost 30 times when employing parallel programming, leveraging the GPU's core count. Moreover, the efficiency improved by over 100-fold relative to sequential execution by utilizing shared memory and obviating the necessity for data transmission between the CPU and GPU. Future work will focus on extending the proposed approach to multiple GPU configurations and exploring its applicability to RNA and protein sequence alignment.

References

[1]. S. A. Shehab, A. Keshk and H. Mahgoub, "Fast Dynamic Algorithm for Sequence Alignment," International Journal of Computer Applications, vol. 37, no. 7, p. 0975 – 8887, 2012.

[2]. M. K. Das and H.-K. Dai, "A survey of DNA motif finding algorithms," in Fourth Annual MCBIOS Conference. Computational Frontiers in Biomedicine, New Orleans, LA, USA, 2007.

[3]. H. Li و N. Homer ،"A survey of sequence alignment algorithms for next-generation sequencing،" BRIEFINGS IN BIOINFORMATICS. vol.5, no. 1, pp. 473- 4832010 .

[4]. S. O. Ouda and M., "Next generation sequencing technologies and challenges in sequence assembly," SPRINGER BRIEFS IN SYSTEMS BIOLOGY, 2014, p. 16–25.

[5]. W. Haque, A. Aravind and B. Reddy, "Pairwise Sequence Alignment Algorithms – A Survey," in ISTA '09' information science Technology and Applications, Kuwait, 2009.

[6]. V. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals.," Soviet Physics Doklady, vol. 10, p. 707–710, 1966.

[7]. S. Needleman and C. Wijnch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," JMol Biol, vols. 48, pp. 443-453, 1970.

[8]. T. F. SMITH and M. S. WATERMAN, "Identification of Common Molecular Subsequences," vol. 147, pp. 195-197, 1981.

[9]. K. M. M. Aung and N. Htwe, "Comparison of Levenshtein Distance Algorithm and Needleman-Wunsch Distance Algorithm for String Matching," vol. 1, no. 1, pp. 209-2013, 2019.

[10]. G. S. Almasi and A. Gottlieb, Highly parallel computing, USA: ACM, 1989.

[11]. C. S, "CUDA Programming: A Developer's Guide to Parallel Computing with GPUs", USA, Elsevire, 2012.

[12]. D. M. Pase and M. A. Eckl, "A Comparison of Single-Core and Dual-Core Opteron Processor Performance for HPC," IBM xSeries Performance Development and Analysis., 2005.

[13]. C. Navarro, N. H. Kahler and L. Mateu, "A Survey on Parallel Computing and its Applications in Data-Parallel Problems Using GPU Architectures," vol. 15, no. 2, pp. 285-329, 2014.

[14]. M. Pharr and R. Fernando, GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (Gpu Gems), ACM, 2005.

[15]. A. Eklund, P. Dufort, D. Forsberg and S. M. LaConte, "Medical image processing on the GPU - past, present and future.," Medical Image Analysis, vol. 17, no. 8, p. 22, 2013.

[16]. D. Negrut, R. Serban, A. Li and A. Seidl, "Unified Memory in CUDA 6: A Brief Overview and Related Data Access/Transfer Issues," TR-2014-09, June 27, 2014.

[17]. K. H. Balhaf and M. F. Abdullah, "GPU-Driven Optimization of the Needleman-Wunsch Algorithm for Fast DNA Sequence Alignment," in 11th International Conference on Optimization and Applications (ICOA), Kenitra, Morocco, 2025.

[18]. S. Batzoglou, "The many faces of sequence alignment," Briefings in bioinformatics, vol. 6, no. 1, pp. 6-22, 2005.

[19]. G. E. Sims, S.-R. Jun, G. A. Wu and S.-H. Kim, "Alignment-free genome comparison with feature frequency profiles (FFP) and optimal resolutions," PNAS, vol. 106, no. 8, p. 2677–2682, 2009.

[20]. M. Yano, H. Mori, Y. Akiyama, T. Yamada and K. Kurokawa, "CLAST: CUDA implemented large-scale alignment search tool," BMC Bioinformatics, vol. 15, no. 406, pp. 1-14, 2014.

[21]. Q. Aguado-puig, S. Marco-sola, J. C. Moure, D. Castells-rufas, L. Aavrez, A. Espinosa and M. Moreto, "Accelerating Edit-Distance Sequence Alignment on GPU Using the Wavefront Algorithm," IEEE Access, vol. 10, pp. 63782-63796, 2022.

[22]. Z. Zhou and Z.-w. Chen, "Dynamic Programming for Protein Sequence Alignment," International Journal of Bio-Science and Bio-Technology, vol. 5, no. 2, pp. 141-150, 2013.

[23]. Q. Bani Baker, . R. Al-Hussien and M. Al-Ayyoub, "Accelerating multiple sequence alignments using parallel computing," Computation, p. 32, 2024.

[24]. J. Makino, T. Ebisuzaki, , R. Himeno, and Y. Hayashizaki, , "Fast and accurate short-read alignment with hybrid hash-tree data structure," Genomics \& Informatics, p. 19, 2024.

[25]. K. Balhaf, M. A. Shehab, W. T, M. Al-Ayyoub, . M. Al-Saleh and. Y. Jararweh, "Using GPUs to speed-up levenshtein edit distance computation," in 7th International Conference on Information and Communication Systems (ICICS), 2016.

[26]. A.-H. Ruba, Q. B. Baker and. M. Al-Ayyoub, "Fast exact sequence alignment using parallel computing," in 9th International Conference on Information and Communication Systems (ICICS), 2018.

[27]. B. Schmidt, F. Kallenborn, A. Chacon and C. Hundt, "CUDASW++ 4.0: ultra-fast GPU-based Smith--Waterman protein sequence database search," BMC bioinformatics, p. 342, 25 2024.

[28]. M. Fakirah, M. Shehab, Y. Jararweh and M. Al-Ayyoub, , "Accelerating Needleman-Wunsch global alignment algorithm with GPUs," in IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA), 2015.

[29]. H. Bahig, M. Hazber, and T. Kenawy, , "Optimized RNA structure alignment algorithm based on longest arc-preserving common subsequence," AIMS Mathematics, pp. 11212--11227, 2024.

BIOGRAPHIES OF AUTHORS

	Professor. Mohammed Fadhl Abdullah is currently a professor of computer engineering in the Faculty of Engineering at Aden University in Yemen. He received his Master's and Ph.D. degrees in computer engineering from the Indian Institute of Technology, Delhi, India, in 1993, and 1998. He was the editor-in-chief of Aden University Journal of Information Technology (AUJIT). He is a founding member of the International Center for Scientific Research and Studies (ICSRS). His main research interests are in the fields of machine learning, parallel algorithms, and cybersecurity. He can be contacted at email: m.albadwi@ust.edu, or al_badwi@hotmail.com.
	Khaled Hassan Balhaf is a PhD student in Information Technology at the University of Science and Technology in Aden. He got his Master's degree in Computer Science from the Jordan University of Science and Technology in 2018. He got his Bachelor's degree in Computer Information Systems from Mu'tah University in Jordan in 2013. He is interested in bioinformatics, artificial intelligence, and parallel programming. He can be contacted at email: khaledbalhaf2021@gmail.com