Journal of Artificial Intelligence and Data Mining (JAIDM), Vol. 12, No. 3, 2024, 435-454.

Shahrood University of Technology

Journal of Artificial Intelligence and Data Mining (JAIDM)
Journal homepage: http://jad.shahroodut.ac.ir

Research paper

PSALR: Parallel Sequence Alignment for long Sequence Read with Hash model

Nasrin Aghaee-Maybodi [1], Amin Nezarat [2], Sima Emadi *[3] and Mohammad Reza Ghaffari [4]

1. Department of Computer Engineering, Islamic Azad University, Yazd Branch, Iran.
2. Department of Computer Engineering, Shiraz University, .Iran
3. Department of Computer Engineering, Islamic Azad University, Yazd Branch, Iran
4. Department of Systems Biology, Agricultural Biotechnology Research Institute of Iran, Agricultural Research, Education, and Extension Organization, Karaj, Tehran, Iran.

Article Info

Article History:
Received 06 May 2024
Revised 02 October 2024
Accepted 24 November 2024

DOI:10.22044/jadm.2024.14462.2554

Keywords:
Indexing, Hash Base, Sequence Alignment, Mapping, MPI.

**Corresponding author: emadi@iauyazd.ac.ir (S. Emadi).*

Abstract

Sequence alignment and genome mapping pose significant challenges, primarily focusing on speed and storage space requirements for mapped sequences. With the ever-increasing volume of DNA sequence data, it becomes imperative to develop efficient alignment methods that not only reduce storage demands but also offer rapid alignment. This study introduces the Parallel Sequence Alignment with a Hash-Based Model (PSALR) algorithm, specifically designed to enhance alignment speed and optimize storage space while maintaining utmost accuracy. In contrast to other algorithms like BLAST, PSALR efficiently indexes data using a hash table, resulting in reduced computational load and processing time, this algorithm utilizes data compression and packetization with conventional bandwidth sizes, distributing data among different nodes to reduce memory and transfer time. Upon receiving compressed data, nodes can seamlessly perform searching and mapping, eliminating the need for unpacking and decoding at the destination. As an additional innovation, PSALR not only divides sequences among processors but also breaks down large sequences into sub-sequences, forwarding them to nodes. This approach eliminates any restrictions on query length sent to nodes, and evaluation results are returned directly to the user without central node involvement. Another notable feature of PSALR is its utilization of overlapping sub-sequences within both query and reference sequences. This ensures that the search and mapping process includes all possible sub-sequences of the target sequence, rather than being limited to a subset. Performance tests indicate that the PSALR algorithm outperforms its counterparts, positioning it as a promising solution for efficient sequence alignment and genome mapping.

1. Introduction

Bioinformatics is the application of computer science, statistics, and probability to molecular biology. Sequence alignment has become a fundamental tool in identifying similarities and differences between sequences as a result of recent advances in molecular biology [1, 2]. Based on dynamic programming, the Needleman-Wunsch algorithm was the first optimal algorithm for aligning two sequences proposed in 1970 [3, 4]. The advent of next-generation sequencing has increased the length of query sequences, making it necessary to develop tools that are faster, more sensitive, and more accurate for mapping short and long queries [5]. To overcome this challenge, various techniques have been developed, such as data compression and parallelism. While multiple alignment tools are available, next-generation sequencing experiments consistently generate exomes or whole-genome sequences for several hundred to several hundred thousand samples within a short period of time, requiring more efficient analysis tools [6].

Most alignment tools use an index-based mapping strategy based on the Burrows-Wheeler transform (BWT) or hash table to address issues regarding storage space and mapping speed [7]. As part of the alignment process, BWT-based tools use the FM-index data structure and the suffix array concept. While it has good speed, it lacks efficient management of mismatches and INDEL/GAPs [8, 9]. The hash table-based method, however, is more accurate, responsive, and efficient in terms of handling non-compliances and INDELs/GAPs. Some BWT-based tools include BWA [9, 10], SSAHA2, SOAP2 [11], Bowtie1 [12], and Bowtie2 [13]. Hash table-based tools such as AGILE [14], SeqAlto [15], Blast [16], PATTERN HUNTER [17], SSAHA [18], and NextGen Map [19] can reduce search times to O(1) and have high speeds despite requiring more memory [7, 20].

Among bioinformatics applications, hashing is widely used for sequence alignment, K-mer calculation, and error correction [21]. However, most hash table-based applications, such as BLAST, use the seed-and-extend strategy, which involves expanding candidate regions (K-mer) on both sides, scoring them, and reporting the best match (K-mer is a substring of length k) [6, 22]. Due to the high number of calculations required for expansion, scoring, and evaluation, this approach consumes 90% of the mapping time [18]. In order to overcome this limitation, researchers have explored various techniques to increase the speed of the process and reduce the memory load on the node, including parallel architectures. Some algorithms use MPI and OpenMP for accurate alignment of sequences [23, 24], while others use CUDA for approximate alignment or multithreading on the GPU for short or long sequence [25, 26]. Alignment tools typically use multithreading for parallelism, with implementations such as Bowtie1, Bowtie2, and BLASR [27] utilizing the Posix Threads technique. There are two versions of the BWA aligner, one utilizing Posix Threads and the other utilizing MPI and ‘distributed memory [28]. SHRiMP [29] and SHRiMP2 [30] are parallel alignments that have a distributed version available [31]. Some alignment tools, such as SOAP3 [32], BarraCUDA [33], and CUSHAW [34], use GPUs to parallelize their process. In some cases, multithreading is associated with SIMD parallelism to use hardware or processor accelerators, including SSE or GPGPUs. To achieve parallelism in indexing and mapping while minimizing data overhead and ensuring safety, this study employs the MPI technique in a fully distributed manner. Unlike GPU-based techniques, multithreading is not suitable due to safety concerns [35].

In this study, we introduce the PSALR algorithm, a significant advancement in sequence alignment technology over our previous research. Our prior work [36] utilized OpenMP for parallel processing, relying heavily on a single processor to manage shared memory for hash table creation and sequence similarity searches. Although effective for smaller datasets, this method struggled with scalability and efficient processing of larger genomic sequences, often resulting in notable overhead and computational bottlenecks.

To overcome these challenges, we developed the PSALR algorithm, which employs MPI to achieve true parallel processing across multiple nodes. This architecture not only speeds up the processing by distributing workload more evenly across nodes but also significantly reduces memory usage. Unlike the OpenMP-based approach, PSALR methodically divides large DNA sequences into smaller, manageable subsequences. These are then distributed across various nodes to prevent any single node from becoming a computational bottleneck.

Moreover, PSALR incorporates advanced data compression and reduction techniques, enhancing the efficiency of sequence alignment processes. This method allows for the immediate initiation of the mapping process upon data arrival at each node, eliminating the necessity for data to be integrated centrally. Each node independently processes parts of the data, thus eliminating delays associated with central data processing and significantly reducing network traffic.

Additionally, results are compiled and returned directly to the user from each individual node, bypassing any central aggregation. This structural refinement not only minimizes the resources and time spent on integrating data but also improves the precision and accuracy of the alignment process. By ensuring comprehensive data processing at each node, PSALR guarantees that no part of the sequence data is overlooked.

The primary objective of this study is to enhance both the accuracy and speed of sequence alignment algorithms while optimizing memory usage. The PSALR algorithm utilizes an MPI-based concurrency framework along with innovative hash-based indexing and compression techniques. Diverging from the traditional seed-and-extend methods prevalent in hash table-based alignment tools, PSALR eliminates unnecessary steps such as expansion, scoring, and evaluation, streamlining the alignment process. However, continuous improvements and modifications are essential to further enhance the algorithm's accuracy, speed, and efficiency. The purpose of this study is to present solutions that address these challenges, which are discussed in more detail below.

- The MPI parallelization technique is used in this study to distribute data and tasks across nodes, resulting in faster processing time.
- In this study, a packing technique is employed to group 32 characters, equivalent to 64 bits, into each packet. These packets are then sent to the nodes in the subsequent phase along with their corresponding packet index. This approach leads to significant enhancements in memory consumption, bandwidth transmission, and data transmission speed.
- To enhance the precision and accuracy of the algorithm, this study allows the user to select the dynamics of K-mers with the desired length. By doing so, the algorithm avoids limited and fixed sensitivity, resulting in more reliable and accurate results.
- The purpose of this study is to create a hash table based on K-mers present within the text rather than all possible K-mers of length k. This results in a smaller hash table, resulting in faster processing times and less memory consumption.
- To enable fully distributed and concurrent indexing and mapping, this study breaks down large sequences in the reference input file into smaller sequences. It allows the sequence to be divided between nodes, resulting in faster processing and better utilization of resources.
- To increase efficiency and reduce latency, this study initiates the mapping process in the nodes upon receipt of the first batch of query sequences. By using this approach, it is not necessary to receive all sequences prior to mapping, which results in faster processing and improved resource utilization.
- To minimize data transfer and improve the efficiency of the algorithm, this study eliminates the need to send results from nodes to a central node for integration and sending to the output. Instead, results are generated locally node and combined on each node, increasing processing speed and reductions in network traffic.

The study is organized as follows: Section 2 reviews the relevant literature in this field. In section 3, the problem is defined, and hypotheses are presented. The proposed solution is presented in section 4, including MPI parallelization, compression and reduction techniques, dynamic selection of K-mers, and a hash table based on K-mers in the text. The results of experiments performed on the proposed algorithm using both overlap and non-overlap techniques are presented in Section 5. The results are compared with three other algorithms regarding time and memory consumption. Lastly, Section 6 concludes and suggests future directions.

2 Literature Review

The first step before mapping is to call the reference or query genome sequences and index them. Some algorithms use reference indexing, others use query sequence indexing, and some use indexing for mapping. The techniques used in alignment are hash tables and BWT. The hash base technique creates a hash table for reference and query sequences. The keys are generated by substrings (K-mer), and their values are a list of the positions of all possible substrings in the sequence [7]. The alignment process with either of the two techniques can be done serially or in parallel. Parallel alignment is more difficult than serial alignment, and the developer must be more careful to solve the problem on multicore platforms by a trade-off between increasing performance and time.

In [16], the BLAST algorithm is introduced as the basis of hash algorithms, with an alignment process that occurs in three stages: input preprocessing, search, and evaluation. In BLAST algorithms, the query sequences are transformed into overlapping K-mers and stored in a hash table. During the search and mapping phases, each K-mer is searched within the reference sequences to identify exact matches. These matches are then expanded on both sides until their score meets or exceeds the specified threshold. Finally, the scores are reviewed for the final evaluation.

In [37], the BLAST algorithm was optimized using an open-source parallelization of BLAST. This optimization aimed to share the database, reduce I/O by storing small utility files, enable parallel I/O on shared files, and implement scalable processing protocols. In this algorithm, the raw reference sequences are formatted, partitioned, and stored in a shared storage space before the search operation. The master node then utilizes a greedy algorithm to assign search sections to worker nodes. Each worker node copies the relevant partition to its local disk and performs the search operation.

Finally, the results from each node are sent to the master for centralized integration, and this process continues until all partitions are completed. Once all the results for the desired query sequence have been received, the master node calls the MPIBlast output function to format and print the results to the output file.

In [38], the parallel implementation of the BLAST algorithm in HPC supercomputers and clusters using thousands of processors is examined. Job distribution and search management are accomplished using a Java library called PCJ. The PCJ-BLAST package is responsible for reading the sequences to be compared, dividing them, and initiating the implementation of multiple NCBI-BLASTs. Additionally, it addresses the issue of parallel I/O by utilizing the PCJ library, aiming to significantly reduce the time required for sequence analysis.

In [39], a simulation-based framework was developed to analyze the scalability and performance of critical optimizations in a parallel genome search program, such as MPIBlast. This algorithm leverages an advanced macro-scale simulator (SST/macro) to enhance the alignment capability.

In [9], an optical parallel processing architecture is utilized. In this algorithm, the query sequence divides DNA into windows by the overlap technique. It then extracts the points in the reference in parallel and, finally, uses a simple algorithm to find the edit distance and analyzes the correlation rate by comparing the window-based DNA sequence using the extracted points and their locations. This algorithm adopts several metamaterial-based optical correlations to implement the proposed parallel architecture. This wave computational architecture completely controls wave and phase transmission using dielectric and plasmonic materials. Although optics provides high-speed processing of alignment results, not every arbitrary algorithm can be implemented effectively using it. So, each algorithm introduced for sequence alignment must consider the limitations and advantages of the nature of parallel processing and the appropriate architecture.

In [40], a BWT-based parallel alignment technique is proposed. This technique utilizes hardware called MPU-BWA to accelerate alignment with minimal modifications to the BWA_MEM software. It integrates seamlessly with PCIe-based infrastructure to achieve significant speed improvements, up to 75 times faster in a clustering environment. The algorithm follows a three-phase approach for the alignment process. First, it performs seed selection, then matches the query sequence, and filters the seeds using a heuristic algorithm. Finally, it expands the remaining seeds. The hardware component is employed specifically in the seed selection and expansion phases.

In [8], clustering algorithms are employed to develop parallel alignment algorithms. This algorithm first identifies regions that can be mapped and then performs the mapping process specifically within those regions. This approach significantly reduces the time required for high-quality alignment when using a local aligner such as BLAST or the Smith-Waterman (SW) algorithm. The algorithm involves a master processor and N-1 worker processors. The master processor and workers collaborate to detect common regions between two strings. Both processors read the input sequences in parallel and determine their lengths. Each processor then extracts and segments the larger sequence overlap. Each segment of the string is read by the Pi processor, which further divides it into overlapping substrings. These substrings are then compared using a binary matrix to calculate the number of matching elements and score the desired segment. Finally, each processor sends its results to the master node for further analysis and processing.

In [41], a parallel aligner utilizing the suffix array technique is proposed. This aligner is designed to rapidly align RNA sequences to servers equipped with multicore processors. The algorithm combines the mapping operation with a suffix array and local alignment to align query sequences using the Smith-Waterman (SW) algorithm. While the suffix array offers faster processing compared to the SW algorithm, it does not inherently support INDEL/GAP acceptance. However, this limitation is overcome by combining the suffix array technique with the local alignment approach. By leveraging this combination, the proposed algorithm achieves high-speed alignment of RNA sequences while accommodating INDEL/GAP acceptance.

In [35], a high-performance parallel K-mer indexing and counting library is introduced. This library is specifically designed for use in distributed memory environments. The library provides a collection of simple and reliable APIs with serial semantics, allowing for flexible and scalable parallel implementations. To ensure safety and minimize data overhead, the algorithm avoids using multithreading techniques and instead utilizes the Message Passing Interface (MPI) technique. By leveraging MPI, the library achieves efficient parallelization without compromising safety or incurring excessive data overhead. Additionally, the algorithm keeps the indexes in memory to reduce the cost associated with accessing the file system when performing operations.

In [26], a parallel alignment algorithm based on the FED algorithm [42] is proposed for accurate sequence alignment. This algorithm utilizes the Message Passing Interface (MPI) technique for parallelization. The FED algorithm employs a general strategy that involves mapping compressed DNA sequences of constant length. Specifically, the algorithm performs alignment without decoding the text by compressing only the reference sequence (text) and generating multiple patterns for the given query sequence. This approach allows for efficient alignment without the need to decode the entire text. However, it's important to note that the FED algorithm is serial-based and may not be suitable for large-scale texts from gene banks. Additionally, the algorithm requires the creation of multiple patterns for mapping.

In [43], an alignment algorithm is proposed that consists of four phases: seed selection, clustering, linking, and scoring. The term "seed" refers to a K-mer that serves as a candidate for mapping between query and reference sequences. The STAR algorithm, introduced in this paper, is claimed to be five times faster than other mappers but requires more memory. For each query sequence, the algorithm searches for the longest sequence that matches exactly with one or more locations in the reference genome. This matching sequence is called the Maximum Mappable Prefix (MMP). In the second phase, the algorithm connects the seeds to form a complete query by clustering adjacent seed bases. This process results in an interconnected set of seeds. Finally, the seeds are selected based on the best alignment for scoring the query, taking into account mismatches and INDEL/GAP information.

In [44], an algorithm is proposed for sequence alignment analysis and comparison using dynamic programming. This algorithm is specifically designed for pairwise alignment within a clustering system in an MPI environment. Notably, the scoring matrix is calculated concurrently in this algorithm. It is important to mention that although this algorithm utilizes dynamic programming for alignment, which is a rigorous and accurate method, it can be slower compared to more recently developed heuristic methods.

In [45], the BFAST algorithm is proposed, which consists of three phases: creating a reference index, finding candidate alignment locations (CALs) using the reference index, and performing local alignment. Local alignment is performed on the possible CAL keys to identify the best possible alignment. The algorithm uses several independent space seeds as a pattern. The seed must match at least one of these patterns.

In [29], the SHRiMP algorithm is introduced, which is capable of handling INDEL/gap variations in addition to mismatches. This algorithm utilizes a mask to generate possible keys for mapping sequences. Based on these masks or patterns, the algorithm does not include some bases in the mapping. The match and mismatch of the bases will not make a difference in the mapping result, and they will be able to control the data polymorphism. They are also allowed to map color-space sequences generated by AB-SoLID. Some tools have recently learned the hash table to improve the alignment process.

In [46], a bit-mapping method is proposed for mapping query sequences to a reference database. This method involves learning the hash algorithm from the transcriptome to generate binary hash codes for sequences. The query sequences are then mapped to the corresponding transcripts based on their hash codes. This algorithm treats the query mapping problem as the nearest neighbor search (NNS) problem in the learning machine, which aims to find the nearest neighbor to the query item by measuring a certain distance.

In [47], a combination of matrix and linked-list data structures is utilized to store sequence information. The matrix represents a two-dimensional grid, where the rows and columns correspond to the hash values generated by two specific hash functions. These hash values act as coordinates for storing and locating sequences within the matrix. By using the hash values as coordinates, the method ensures efficient storage and retrieval of sequences. The matrix provides a structured framework for organizing the sequences based on their hash values, allowing for quick access to the desired sequences. Additionally, linked lists are used within each matrix cell to handle collisions or multiple sequences with the same hash values.

In [48], the authors propose a novel seeding approach that relies on long inexact matches rather than short exact matches. They demonstrate that this approach yields a better trade-off between time and accuracy in settings with up to a 25% mutation rate. To achieve this, they utilize sketches of a subset of graph nodes, which are more robust to indels. These sketches are stored in a k-nearest neighbor index, effectively mitigating the curse of dimensionality. Their approach stands in contrast to existing methods and emphasizes the significant role that sketching in vector space can play in bioinformatics applications. The authors further demonstrate that their method can scale graphs with 1 billion nodes and provide quasi-logarithmic query times for queries with an edit distance of 25%. In fact, for such queries, longer sketch-based seeds result in a 4× increase in recall compared to exact seeds.

In [49], the authors introduced a novel sequence alignment technique called ESA. This algorithm is implemented on the Sunway TaihuLight architecture and is capable of performing both local and global alignment. The algorithm incorporates several advanced features, including cache-aware sequence alignment, capacity-aware load balancing, and bandwidth-aware data transfer. However, one limitation of ESA is its relatively high computational time. Additionally, when the lengths of the sequences being aligned differ significantly, ESA may encounter an issue of asymmetric load distribution among the processors.

In [50], the authors introduced the FMapper algorithm, specifically designed for the TaihuLight supercomputer. This algorithm is optimized to leverage the computing power of the fourth-generation ShenWei multi-core architecture (SW26010). The FMapper algorithm incorporates dynamic task scheduling, synchronous I/O, and data transfer techniques to maximize performance and efficiency. The authors achieved a significant speedup of 6 compared to the naïve implementation. Additionally, when scaling up to 512 compute groups, they observed a strong scaling efficiency of 65%."

In [51], The main objective is to find the maximum alignment region between two sequences and then identify the seeds within that region to increase sensitivity. In this algorithm, artificial intelligence rules are used to find additional seeds with different lengths. Additionally, this algorithm can be used for weighted seeds. The "if-else" rule is a simple expression in AI that is used to determine the length of seeds to be searched for and whether overlapping seeds should be merged or discarded.

In [52] The computational burden of algorithm is alleviated by utilizing the LexicHash method to estimate sequence similarities. To achieve this, the algorithm performs a hash function on each k-mer within the read sequence and stores the minimum hash value. By counting the number of minimum hash matches between pairs of reads, the algorithm can estimate the similarity between two sequences. It is crucial to carefully choose the parameter k when identifying sequences. Increasing the value of k enhances accuracy and precision, but there is a possibility of losing some matches.

In [36], the authors employ the OpenMP parallelization method and shared memory to enhance performance. The method involves dividing sequences amongst processors, with each processor dividing its reference sequences into completely overlapping k-mers. A shared hash table is then created with the assistance of other processors. In the subsequent step, each processor receives a query sequence and, using the shared hash table checks the percentage of similarity between the query sequence and the sequences in the hash table. The result is then returned to the user. Although this method offers several advantages, such as ease of implementation and reduced overhead due to the use of shared memory, it can only run on a single node and is not distributed. Additionally, if the reference sequences are few but lengthy, a few processors may have to handle a substantial workload. Therefore, to enhance this method, distribution can be increased, and other methods can be utilized to manage the load if necessary.

Most of the algorithms mentioned in the literature require special hardware platforms or the addition of special software and algorithms to enable parallelism in the alignment process. However, these approaches often result in overheads, and in many cases, only the overlap technique is used in one of the input sequences, typically the query. This approach leads to only a portion of the reference sequences being placed in the hash table and subsequently used in the search and mapping phases, potentially resulting in reduced accuracy in the output results. A detailed comparison of the above algorithms is provided in Table 1.

3 Methodology

This algorithm employs a novel approach compared to many hash table-based applications, such as the BLAST family, which utilizes the seed-and-extend strategy. Instead, this algorithm uses an SSAHA-based method, dynamically selecting the K-mer size and utilizing the overlap technique to extract them from both input sequences. The overlap technique can improve accuracy by up to 100% regardless of INDEL/GAP and mismatch.

To optimize time and memory management, this algorithm utilizes parallelism, with a master node and N-1 worker nodes responsible for compressing and dividing data, creating a hash table, and mapping query sequences.

The algorithm breaks down large reference sequences into smaller ones and distributes them to different nodes, allowing for fully distributed and concurrent indexing.

Table 1. Comparison of parallel alignment algorithms.

Reference	**Year**	**Algorithm**	**Alignment technique**	**Parallelism MPI/Openmp/ GPU**	**INDEL/Gap acceptance**	**The use of hard/soft ware**	**Techniques for memory optimization**
Altschul, S.F., et al	1990	BLAST	Hash-Base	--	yes	No	**No**
Lin, H., et al.	2005	MPIBlast	Hash-Base	MPI	yes	Greedy Algorithm	**No**
Nowicki, M., et al.	2018	Parallel-Blast	Hash-Base	MPI	yes	PCJ-lib	**No**
Dechev, D., et al.	2013	Parallel-Blast	Hash-Base	MPI	yes	SST/macro	**No**
Mozafari, F., et al.	2018	WOC	Hash-Base	NA	yes	Optics	**No**
Dobin, A., et al.	2013	STAR	BWT-Base	MPI	Yes	---	**No**
Vijayaraghavan, T., et al.	2018	MPU-BWA	BWT-Base	--	Limited number	MPU	**No**
Bandyopadhyay, S, et al.	2009	RPAlign	Hash-Base	MPI	Yes	---	**No**
Martinez, H, et al.	2015	HPG Aligner SA	Suffix Array & Hash-Base	---	yes	----	**No**
Pan, T., et al.	2019	Kmerind	Hash-Base	MPI	Yes	---	**NO**
Q. Xue et al.	2014	Fast Matching Method	Hash-Base	MPI	Yes	Multi-Pattern	**Compress**
Chen, Y, et al.	2006	Parallel Pairwise	Hash-Base	MPI	Yes	---	**NO**
Yu, X, et al.	2020	Learning hash-table	Hash-Base	---	NA	-----	**Hash-code**
Homer, N, et al.	2009	BFAST	Hash-Base	POSIX	Yes		**Multi-level-index**
Rumble, S.M., et al.	2009	SHRiMP	Hash-Base	OpenMP	Yes	------	
Esmat, A., et al.	2022	Parallel-Alignment	Hash-Base	OpenMP	Yes	-------	**Yes**
Peng, F., et al.	2022	Matrix-LinkedList	Hash-Base	-------	NA		**Yes**
Canzar, S., et al.	2023	Graph Alignment		----	Yes	------	-----
Muhammad,U. et al.	2023	scalable parallel algorithm	Needleman-Wunsch	GPU	Yes	NA	**No**

Additionally, nodes do not need to return their results to the primary node, and each node puts its results in the output file.
An evaluation method is used to assess the accuracy, precision, and sensitivity of the algorithm, which demonstrates that selecting sequence overlapping bases not only increases accuracy but also does not reduce sensitivity. Overall, this algorithm provides a more efficient and accurate approach to sequence alignment without the need for specialized hardware or software

3.1 Problem Definition

In this study, we aim to optimize the alignment of genomic sequences by improving the storage process, managing memory, and increasing execution speed. To achieve this goal, we propose a novel algorithm that utilizes various techniques, including parallelism, compression, reduction, and hash-based indexing. These techniques enable efficient memory management and faster processing without compromising accuracy or sensitivity. PSALR Algorithm dynamically selects K-mers with desired lengths and enables user-defined overlap to enhance precision and accuracy. Additionally, PSALR breaks down large reference sequences into smaller ones and distributes them across nodes for fully distributed and concurrent indexing. This algorithm eliminates the need to return results to a central node, reducing network traffic and improving efficiency. The purpose of this study is to investigate the alignment of genomic sequences, improve the storage process and sparse execution, and at the same time the indexing and mapping steps to manage memory and increase speed, definitions are needed that are detailed in [36]. But it is briefly described below.

3.1.1. Definition 1

In sequence alignment, the reference sequences are known sequences that are stored in a database, while the query sequences are unknown sequences that are compared to the reference sequences to identify and predict their structure. The goal of sequence alignment is to identify regions of similarity between the query and reference sequences, which can provide insights into the evolutionary relationships, functional domains, and other important features of biological molecules such as DNA, RNA, and proteins.

$$\mathrm{Re}f = (seq_1, seq_2, ..., seq_m) \ , m \geq 1$$
$$Query = (seq_1, seq_2, ..., seq_n) \ , n \geq 1$$

3.1.2. Definition 2

In DNA sequencing, each DNA string is represented by the four nucleotide bases: A (adenine), C (cytosine), G (guanine), and T (thymine). Depending on the length of the DNA string, these nucleotides combine to form the sequence. To optimize memory consumption in data storage, binary numbers are used instead of characters. This means that only two bits are needed to represent each nucleotide base instead of the standard eight bits used to represent a single character. By compressing the data in this way, it can reduce the memory footprint of the DNA sequences and improve the efficiency of the alignment process.

$$F(A) = (00)2 \qquad F(C) = (01)2$$
$$F(G) = (10)2 \qquad F(T) = (11)2$$

3.1.3. Definition 3

A K-mer is a sub-string of length k that is a continuous sequence of DNA bases within an input sequence. The number of K-mers within a string is obtained from the relation $N + K - 1$ if the K-mers overlap or from the relation N / K if they do not overlap. According to Definition 2, each K-mer can be represented as a unique number with 2k bits, which is referred to as the mer index. The mer index can be created using Equation 1.

$$E(w) = \sum_{i=1}^{k} 4^{i-1} f(bi) \quad i = 1, 2, ..., k \qquad (1)$$

3.1.4. Definition 4

The hash table is defined as a triple (w, E(w), Position), where w is a K-mer, E(w) is its index, and position is an array of positions of w within the reference file. This hash table allows for efficient indexing and searching of K-mers within the reference file, enabling the alignment algorithm to quickly identify regions of similarity between the query and reference sequences. When a query sequence is received, the algorithm uses the hash table to locate the K-mers within the query sequence and then searches for matching K-mers within the reference file. The positions of these matching K-mers are stored in the Position array, allowing the algorithm to identify potential regions of similarity between the query and reference sequences.

3.1.5. Definition 5

To search for all query sequence hits within the reference sequences, the PSALR algorithm scrolls through the query sequence from base zero to (l-k), where l represents the length of the query sequence. For each base t, it obtains the list of positions r, which represents the occurrence of the K-mer $w_t(Q)$ within the query sequence, from the hash table. It then extracts the list of K-mer positions and place them in a table, which will be used for mapping in the next phase. Finally, the algorithm calculates the list of hits using Equation 2, as described in Ning, Cox, and Mullikin (2001).

$$H = (i_1.j_{1-t}.j_1)(i_2.j_{2-t}.j_2)...(i_r.j_{r-t}.j_r) \quad (2)$$

The value t represents the distance of the K-mer from the beginning of the input sequence. The collision list contains three elements: index (i_r), shift (j_{r-t}), and offset (j_r), which are used to identify the locations of the K-mer match within the reference sequence. The collision list is sorted first by index and then by shift, allowing for efficient mapping of the query sequence to the reference sequence. By sorting the collision list in this way, it can quickly identify regions of high similarity between the query and reference sequences and accurately align the sequences.

In the final step of this algorithm, the list of hits based on index, shift, and offset it sorted. Then, the algorithm performs a scan to identify hits that have the same index and shift, which allows us to determine the corresponding bases between the query and reference sequences. If the algorithm is allowed to accept INDELs, there may be differences between the positions of the hits that are equal to the number of INDELs present. In this case, closely matched areas can be combined to create larger regions for GAP acceptance. By using this approach, this algorithm can accurately align the query and reference sequences, even in the presence of INDELs or other types of variations.

4. The Proposed Technique

Algorithm PSALR utilizes MPI parallelism to divide the operation process into two parts. The master or zero node performs certain operations such as receiving, preprocessing, compressing, and sending data, as shown in Figure1. The worker nodes are responsible for receiving data, extracting K-mers, creating hash tables, and searching and mapping query sequences concurrently.

In the master section, a node receives the sequences from the input files and preprocesses them. Each character (base) in the sequence is converted into two binary bits to reduce their size, and the compressed data is placed in 8-byte packets before being sent to the worker nodes.

In the worker section, each node receives its sequences, extracts the overlapping K-mers with a window length of one from the received reference sequences, and creates a hash table from them. The worker nodes then perform search and mapping with the query sequences received in the hash table and print the mapping results of their sequences in the output file.

By utilizing MPI parallelism in this way, the PSALR algorithm can efficiently process large amounts of data and speed up the alignment process, making it a valuable tool for genomic research and related fields.

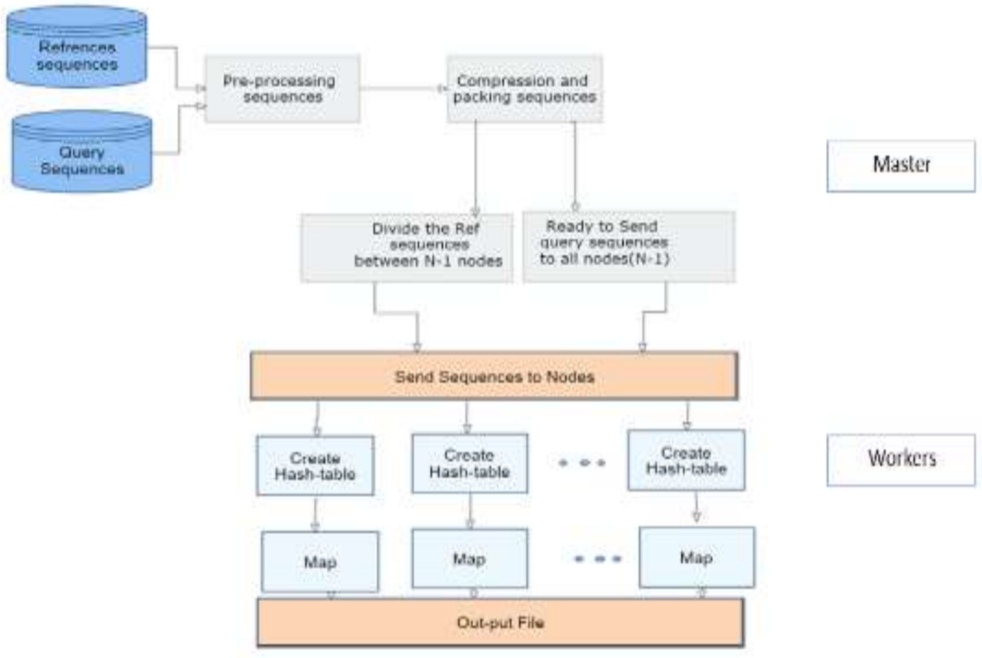

Figure1. The PSALR framework

Problem: The task is to find encrypted pattern(s) P' in encrypted sequence(s) T' without decoding, using n nodes for concurrent processing. The reference file may contain one or several sequences, while the query file may contain one or several patterns.
To address this problem, we propose an algorithm that operates in both zero and non-zero nodes. Algorithm 1 presents an example of parallel execution in these two operational nodes. The master node performs operations in lines 1-39, while the worker node performs operations in lines 40-43. Details and code for both nodes are provided in the following sections.

Algorithm 1: Query search in the Hash table and extract their positions in each node

```
1.  IF Node == 0 THEN
2.    For each sequence (seq) in the dataset, do
3.      IF (seq is large) THEN
4.        Initialize start_index
5.        WHILE (start_index + 180 < seq.size()) do
6.          Add 180 characters from start_index to Section_data
7.          Increment start_index by (180 - k)
8.        END WHILE
9.        Dispatch(Section_data) // Continue processing sectioned data
10.     ELSE
11.       For i <- 1 to seq.size(), i += 32 do
12.         Add 32 characters from position i to Section_seq
13.       END FOR
14.       IF (mod exists) THEN
15.         Add remaining characters (mod) to Section_seq
16.       END IF
17.     END IF
18.   END FOR
19.   Count <- seq_count / Number_threads
20.   For i <- 1 to Count, do
21.     MPI_Send(&ready, 1, MPI::BOOL, executer_id, READY_TAG, MPI_COMM_WORLD)
22.     MPI_Send(&k, 1, MPI_UNSIGNED, executer, GLOBAL_K_TAG, MPI_COMM_WORLD)  // K-mer size
23.     For each data in Section_seq, do
24.       Orgin = compress(data)
25.       MPI_Send(&orgin, 1, MPI_UNSIGNED_LONG, executer_id, ORIGIN_TAG, MPI_COMM_WORLD)
26.     END FOR
27.     MPI_Send(&dataset_index, 1, MPI_UNSIGNED, executer_id, DATASET_INDEX_TAG, MPI_COMM_WORLD)
28.     MPI_Send(&mer_index, 1, MPI_UNSIGNED, executer_id, DATASET_INDEX_TAG, MPI_COMM_WORLD)
29.     MPI_Send(&section_seq_size, 1, MPI_UNSIGNED, executer_id, SECTION_SIZE_TAG, MPI_COMM_WORLD)
30.   END FOR
31. END IF
32. ELSE IF Node != 0 THEN
33.   For i <- 1 to Count, do
34.     MPI_Recv(&ready_for_seq, 1, MPI::BOOL, 0, READY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
35.     MPI_Recv(&origin, 1, MPI_UNSIGNED_LONG, 0, ORIGIN_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
36.     MPI_Recv(&sectioned_seq_size, 1, MPI_UNSIGNED, 0, SECTION_SIZE_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
37.     MPI_Recv(&debug_f_l2, 1, MPI::BOOL, 0, DEBUG_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
38.     MPI_Recv(&big_seq, 1, MPI::BOOL, 0, DEBUG_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
39.     MPI_Recv(&k, 1, MPI_UNSIGNED, 0, GLOBAL_K_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE)
40.   END FOR
41.   Extract_Kmer_from_origin()
42.   Create_Hash_table()
43.   Extract_Query_Position_in_Hash_table()
44.   Perform_Mapping()
45. END IF
```

The zero or master nodes execute three main phases, namely input reception, encoding, and sending. These phases are explained in detail below:

- **Input Reception**

DNA sequences consist of two complementary strands, which are represented as separate files during preprocessing. The query sequences are typically provided in .fastq format, while the reference sequences are in .fasta format. The input sequences may contain the character N instead of one of the four main bases, which is often randomly replaced with one of the bases in other algorithms. However, in PSALR, N characters are eliminated to prevent incorrect events and improve accuracy, as the algorithm considers overlapping bases and calculates both mismatches and gaps.

- **Encoding**

Parallel and network algorithms face challenges such as file transfer and bandwidth occupancy. The query and reference files contain characters that require significant memory and bandwidth to transmit since each character occupies 8 bits. To address this issue, this algorithm employs compression techniques that reduce each character's size to 2 bits, allowing the transmission of four characters with a single byte. The nodes can continue searching and aligning by receiving the transmitted encoded characters without decryption, which saves time and memory compared to some existing algorithms.

During this phase, the string is encrypted and packaged into 8-byte packets, as outlined in Pseudo-code 1. This algorithm packs 32 characters, or 64 bits, into each packet and then sends the packet index to the nodes in the next phase. The encryption and packaging process is performed for all query and reference sequences, significantly improving memory consumption, bandwidth transmission, and data transmission speed.

To encode each character as an unmarked integer with two bits in each nucleotide, binary numbers are used instead of characters. Since the string length may not be a multiple of 32, the last packet may not be complete. Thus, the final package size is calculated as follows:

Last_Pack = String.size() % 32

After sending the string data packets, an eight-byte status packet is sent to indicate if the final packet contains several characters.

- **Sending**

This section details the three types of data that need to be transmitted in this algorithm. The first type comprises compressed packets of reference sequences, which must be divided among the nodes to concurrently create a hash table. The second type is the user-specified K-mer size, which must be sent to all nodes. The final type consists of compressed packages of query sequences, which must be sent to all nodes for searching and mapping based on the hash table created in subsequent phases.

When using MPI parallelism in algorithm, the sequences are divided among the nodes. If N is the number of nodes and M is the number of reference sequences, each node is assigned approximately M/N sequences. Thus, the zero nodes must send the encoded packets of each sequence to one node, which extracts K-mers from the received packets and creates a hash table. If a reference sequence is large, it can be broken into subsequences and distributed among the nodes to avoid overloading any one node. For instance, if the master node is W_M and the other nodes are W_S, a large sequence S_L can be divided into segments, with each node assigned a segment except the master. If the K-mer size is k, the size of each segment F_i, i = 1, 2, ..., n is calculated as follows:

$$\frac{SL}{N-1}+K$$

Where N is the number of processors, the beginning and end of each segment can be calculated as follows:

$$start[i]=(i-1)*\frac{SL}{N-1}+1$$

and

$$End[i]=i*\frac{SL}{N-1}+K$$

Next, operations performed on non-zero or worker nodes, including subsequence extraction, hash table creation, query sequence search, and mapping, are described.

- **K-mer Extraction**

Each node receives the K-mer size (k) and compressed packets of reference sequences, which are extracted using the overlap technique without decoding the packets in sequential shifts. At each time step, the node receives a query sequence from the zero nodes and generates a list of K-mers from it.

Many hash-based aligners only index non-overlapping K-mers of the reference database to preserve memory. This means that they only include 1/k of the database locations in the index table for K-mers of length k. For example, if the K-mer size is five and a query or reference sequence is as follows, a window of five characters is drawn on the sequence using the overlap technique. This approach returns all possible subsequences of the string as K-mers. Thus, the number of K-mers obtained from the sequence is calculated as follows:

Number (k_mer) = (n - K_mer+ 1)

Where n is the length of the strings or the number of bases. In the following example, the number of K-mers will be equal to 11.

S1= CGTCACTCTGAGGAT

K-mers is: GTCA, GTCAC, TCACT, CACTC, ACTCT, CTCTG, TCTGA, CTGAG, TGAGG, GAGGA, AGGAT

Regardless of the overlap technique, the number of K-mers in the same string that reaches the search and mapping phases is only part of all of the string subsequences shown below.

K-mers is: CGTCA, CTCTG, AGGAT

In other words, the number of K-mers that reach the main phase is obtained by dividing the string length by the size of K-mer, which will be only three K-mers in the same example.
In this small example, the difference in the number of K-mers that reach the search and mapping phases can be seen. This difference in datasets with millions of bass characters can significantly reduce the output accuracy. Many algorithms convert only one of their sequences to overlap and the other to non-overlap and send it to the search and mapping phases so that they can maintain some accuracy because selecting K-mers with the overlap technique increases the amount of memory several times. Some algorithms even use the non-overlap technique for both query and reference sequences, sacrificing performance and accuracy for speed and memory. However, this algorithm uses the overlap technique for both query and reference sequences and tries to manage memory and time using techniques that will be discussed later so that they do not increase dramatically and even improve in many cases, and bring its accuracy closer to 100 by considering two mismatches and two gaps per K-mer.

- **Hash Table Creation**

The first process in alignment is to create a hash table for the reference sequence. The indexing process begins after the nodes receive the reference sequences. As mentioned earlier, not all possible states of K-mer are included in this table, and only the K-mers in the sequence are placed in the hash table by moving over the desired sequence. Their position is then recorded in the table. According to definition 4, the hash table consists of three parts: w, E (w), and position. In SSAHA, two data structures are used to create a hash table: a list of positions and an array of pointers to the list. Since this algorithm puts all possible states of K-mer in the table, 4^k pointers are required. Pointers in position E (w) point to the entry in the list of desired K-mer positions. However, in this algorithm, not all possible states of K-mer are entered, and K-mers are placed in the hash table that is in the reference sequences. So, it helps to reduce the hash table.
Another problem is using two passes to create a hash table in the SSAHA algorithm. In the first pass, all non-overlapping events are counted in each of the 4^k possible states, and in the second pass, the event information of that K-mer is placed in the reference list in the reference sequences.

Algorithm 2: Create a Hash-table for every node

Input: A set of sequences in the Reference file
Output: A hash table with k-K-mers of the reference file and their position

```
01 Initialize a hash table: Map <unsigned long, vector<unsigned int>> table1
02 For each sequence in Ref_file do
03    For each K-mer in the sequence do
04       Split the K-mer into substrings (K-mer, k, mersvector)  // Store K-mers in mersvector
05    End For
06 End For
07 For each K-mer in mersvector do
08    If K-mer exists in table1 (insertion.second is false) then
09       Add dataset_index and mer_index to the existing entry in the hash table
10    Else
11       Create a new entry in the hash table with dataset_index and mer_index
12    End If
13 End For
17 End For
```

In the PSALR algorithm, all overlapping K-mers can be completed with a one-pass hash table. In other words, the K-mer positions are placed in the position list in the order of their passage, passing through the beginning of the reference sequences. Algorithm 2 displays pseudo-code to create a hash table.

5. Evaluation

In this section, the PSALR algorithm is evaluated with overlap and non-overlap techniques and compared with the other three algorithms in terms of memory and time consumption in the indexing section. Experiments are performed on datasets with different numbers and lengths based on Table 2, the results of which are analyzed in the next section. Implementations and evaluations are done on a machine with 128 cores and 256 GB of memory. STAR, BFAST, and SHRiMP algorithms are used to evaluate and compare the proposed algorithm. Some aligners argue that using all sequence bases for indexing and mapping reduces the sensitivity of the algorithm, and if a base mutates in K-mer, it will be rejected in the mapping phase.

Table 2. Datasets used in experiments.

DB	Organism	Source		Name	Size	Sequence-number
DB	Fragaria vesca f.alba	Genomic	Query	SRR072029.fastq	3.1GB	**2,727,589**
	H. sapiens	Genomic	Ref	GRCH37.p13 CHROMOSOME 1	741MB	**1**
DB2	H. sapiens	1000 Genome HG00096, NCBI	Query	SRR077487.fastq	2.6GB	**7,757,821**
	H. sapiens	1000 Genome reference GRch37	Ref	GRCH37.p13 CHOROMOSOME x	208MB	**1**
DB3	H. sapiens	Genomic	Query	SRR494099.fastq	2.6GB	**14,166,619**
	H. sapiens	NCBI Nucleotide CM0004 63.1	Ref	GRCH38-Genome CHROMOSOME 2	439MB	**1**

Therefore, they use Space Seed technique to extract K-mers from sequences that will reduce accuracy. The Space Seed technique in bioinformatics is a method used to enhance the sensitivity of sequence mapping by selectively considering specific seeds of a sequence rather than analyzing the entire sequence. This technique aims to improve the detection of similarities or patterns between sequences while allowing for some degree of mismatch. However, this study shows that not only accuracy but also sensitivity will be increased by selecting all bases and involving them in indexing and mapping and that the False Negative problem will be prevented by considering mismatch and indels in K-mers. The details will be explained below.

The first item in the experiments is size K. As mentioned in the literature review, the K-mer size can be considered differently. Selecting a small K-mer size increases sensitivity but increases false positives (FP). Instead, selecting a large K-mer reduces sensitivity, speeds up the process, and reduces FP. Selecting longer K-mers in indexing will result in less FP. In this way, a better alignment will be obtained [53]. Most algorithms, such as the BLAST family, use a small k-size, or algorithms, such as BFAST, use K-mers longer than 14 to manage memory at two-level or higher indexes. Some algorithms, such as ELAND, MAQ, BFAST, and SHRiMP use the space seed technique so that not all bases are considered for mapping, and only the part specified in the template is selected to increase the sensitivity and accept some mismatch by defining such patterns. For example, if a pattern is considered to be 11101001, the size of K-mer is 8, but its weight is 5, and only 1,2,3,5,8 K-mer bases are used to map two sequences. These algorithms generally use fixed lengths and weights by default to perform the alignment process. Space seeds and their shifted samples map fewer positions for adaptation and do not include INDEL [7, 20]. In a group of these algorithms, one or more templates are defined independently so that at the time of mapping, the K-mer must match at least one of the templates. However, another group follows the seed-and-vote technique, in which several seeds jointly identify a candidate region. In space seed-based algorithms, K-mers are first extracted from the query based on the defined template and placed in the index table (in the case of reference-indexed algorithms, this is done for reference sequences). The search and alignment then take place in another sequence [54]. However, the proposed algorithm uses larger k sizes from the BLAST family and other common algorithms by selecting sequential and overlapping bases to reduce false positives in addition to improving accuracy.

Using the overlap technique in query and reference sequences, the PSALR algorithm tries to complete the process accuracy, which significantly increases memory consumption, especially during indexing. Therefore, two groups of comparisons are used in the next section to evaluate the algorithm. The first group is the comparison of memory and time consumed to index the data in the proposed algorithm with overlap and non-overlap techniques, and the second group is the comparison of memory and time with three other algorithms with a different number of processors at the time of data indexing.

5.1 Comparison of the proposed method with overlap and non-overlap techniques

In this section, the memory and time required to index the proposed algorithm with overlap and non-overlap techniques are reviewed and compared.

In the first mode, like most aligners, query sequences are considered with the overlap technique, reference sequences are considered with the non-overlap technique, and indexing and mapping are performed on the three datasets mentioned. In the second mode, K-mers of query and reference sequences are extracted by the overlap technique, and indexing and mapping are performed. As mentioned earlier, hashing algorithms consume the most memory and time in indexing. In the PSALR algorithm, after indexing, the queries are entered sequentially and aligned. They then go out and free up memory so that the amount of memory consumed does not increase. The implementation of the algorithm on three datasets by overlap and non-overlap techniques can be seen in Figure2. As can be seen in the figure, the amount of memory consumed by the overlap technique is several times higher than the non-overlap one due to the increase in the number of K-mers. For example, the amount of memory consumption in the K-mer with a length of 17 in DB1 is significantly reduced from 38.4GB to 3.53 GB. Therefore, most aligners use the same technique and extract only query sequences by the overlap technique so that they can maintain some accuracy. Algorithms such as SSAHA, SNAP, and BSSHA are examples of this.

The time required to index each of the DBs by overlap and non-overlap techniques is shown in Figure 3. For example, in the same DB1 and K-mer with a length of 17, the consumption time increases from 1159 to 86 seconds because, in the overlap technique, all sequence K-mers must be extracted and placed in a table.

However, in the non-overlap technique, only 1/k is extracted from the subsequences and placed in the table, so it greatly reduces memory and time but lacks the necessary accuracy for the reasons stated earlier. In this study, 2,4,8 and 16 processors are used to evaluate the algorithm and execute it in parallel. In the following, the amount of memory and time consumed by the algorithm with two techniques and execution on 2 and 16 processors with k of different lengths are shown. The amount of memory consumption of datasets is compared by the overlap technique with 2 and 16 processors in Figure 4.

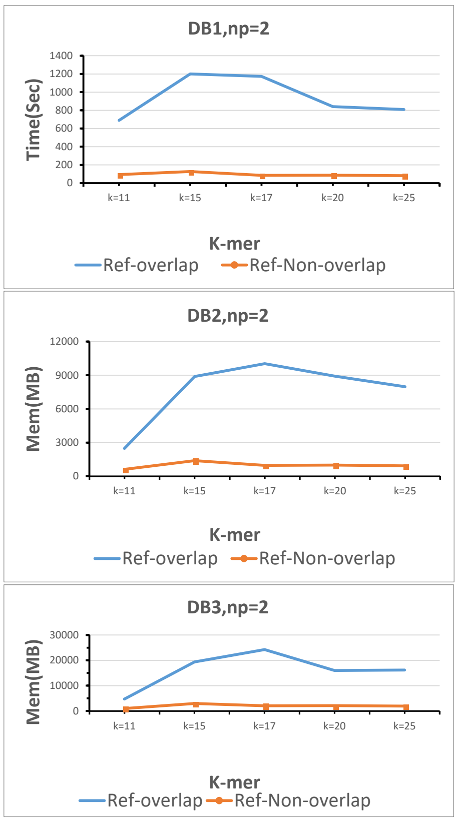

Figure 2. Comparison of memory consumption in three datasets by overlap and non-overlap techniques with two processors.

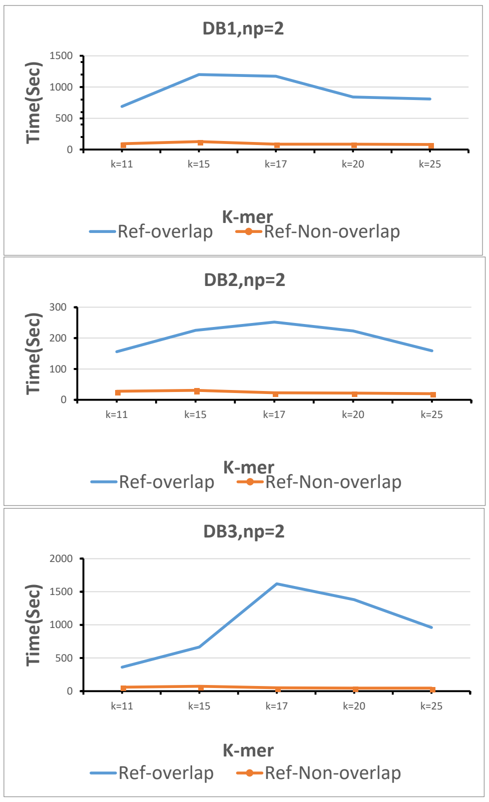

Figure 3. The time required for indexing by overlap and non-overlap techniques with two processors.

When the concurrency technique is used, each node receives a segment of the data sequence and generates its index table. In this way, the amount of memory consumed is reduced in proportion to the number of processors. However, in 2-processor mode, only one node is responsible for creating the hash table, which increases memory and time consumption. Therefore, creating a distributed hash table can help to consume memory several times.

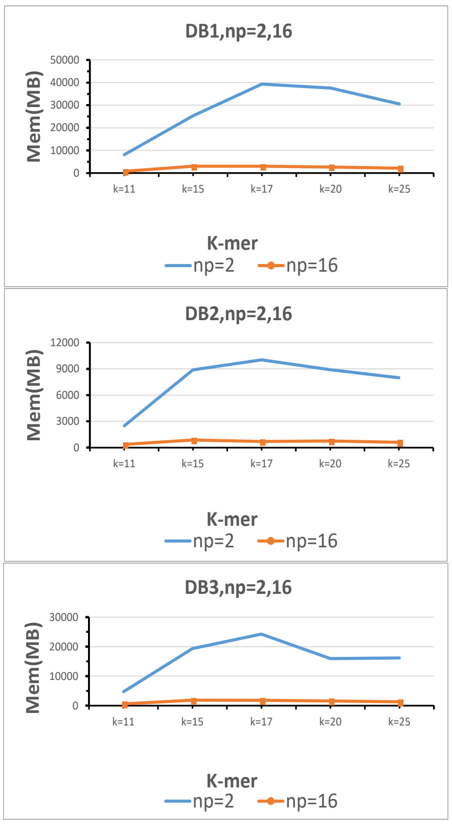

Figure 4. Consumption memory for indexing datasets with 2 and 16 processors.

Figure 5 shows the time required to index the reference sequence in different datasets with this number of processors. As the number of processors increases, the time required for indexing is greatly reduced. For example, the time required to index a DB1 by 2 processors in K-mers with a length of 15 is 1800 seconds but is reduced to 100 seconds with 16 processors. Therefore, creating a distributed index table can reduce the speed by several times in addition to memory reduction.

Based on the above, it can be concluded that the proposed algorithm in K-mer with a length of 17 has the highest memory and time consumption and that by increasing the length of k again, memory and time consumption decrease because the number of K-mer in the overlap technique is obtained based on the formula $N-K+1$, and increasing the length of k reduces their number.

5.2 Comparison of the proposed algorithm with other algorithms with different processors

In this section, the PSALR algorithm is compared with the other three algorithms. The overlap technique of query and reference sequences is used to extract K-mers and align the sequences to evaluate and compare the PSALR algorithm. The version used of the three compared algorithms can be seen in Table 3.

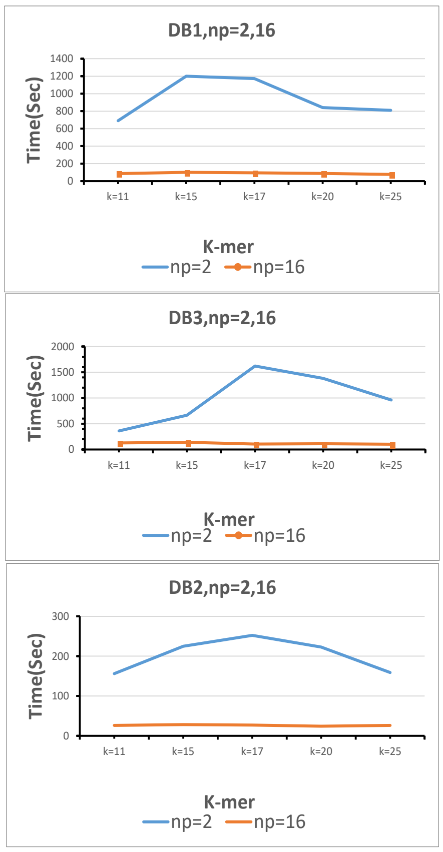

Figure 5. Consumption time for indexing datasets with 2 and 16 processors.

The three STW algorithms, which are based on BWT, and the BFAST and SHRiMP algorithms, which are both hash-based, are compared with the proposed algorithm. Since BWT algorithms do not use K-mers for alignment and space seed algorithms, typically use fixed K-mers with different weights to extract K-mers, in this section, K-mer with lengths of 11 and 20 are used by default of other algorithms to evaluate the PSALR algorithm.

The memory and time consumption of the PSALR algorithm using K-mer with length 11 is compared with other algorithms in Figure 6 and 7.

Table 3. Alignment tools.

Tool	version	Technology
STAR	2.7.5.a	**MPI**
BFAST	0-7-0	**POSIX**
SHRiMP	2-2-2	**OpenMP**

As shown in Figure 6, we have the highest amount of memory at np = 2 because the distribution reaches zero, and only one processor is involved in creating the table. Although the PSALR algorithm uses the overlap technique in extracting K-mers, it manages memory consumption using the mentioned techniques and performs better than other algorithms. In this algorithm, the hash table is created even with a large, fully distributed sequence. Therefore, as the number of processor increases, memory consumption decreases.

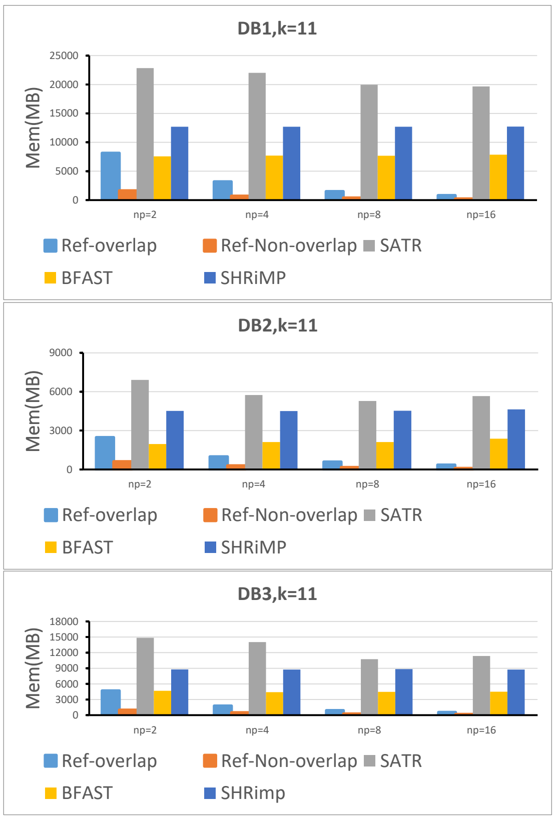

Figure 6. Comparison of memory consumption in the PSALR method with K-mer = 11.

However, in other algorithms, the memory consumption at different np is not much different from each other. The memory and time consumption of the PSALR algorithm with a different number of processors is compared with other algorithms in Table 4.

The time required to index the reference table in the proposed algorithm with overlap and non-overlap techniques and its comparison with other algorithms can be seen in Figure 7. Although the reduction in time consumption can be seen as the number of processors increases in all algorithms, the proposed algorithm performs better with both techniques than the other algorithms. According to the figure, SHRiMP has the most time consumption among algorithms. The same comparisons with K-mer = 20 can be seen in Figure 8 and 9.

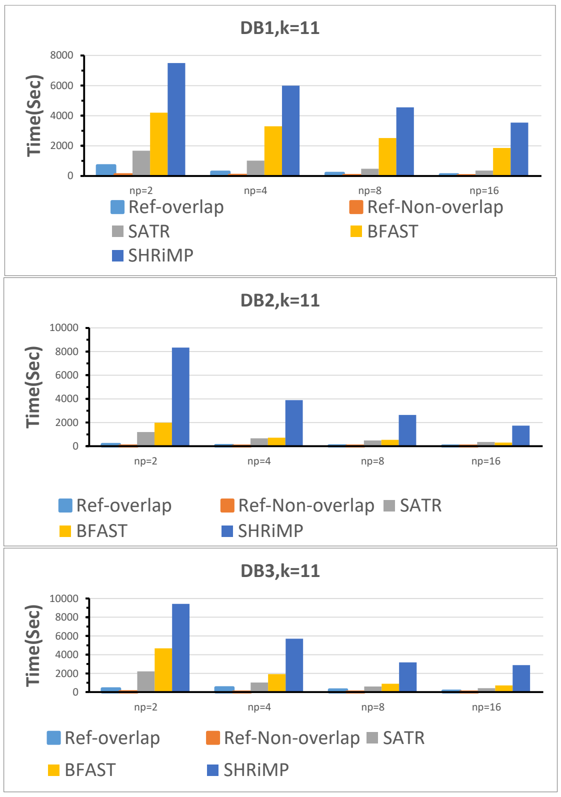

Figure 7. Comparison of time consumption of the PSALR algorithm with K-mer = 11.

When the overlap technique is used and all possible K-mers are extracted from the sequence, high time and memory are required to extract and index them. As shown in Figure 8, the length of K-mers has increased compared to the previous experiment. The PSALR algorithm does not work well compared to other algorithms in the overlap technique and np = 2, which is without distribution, and only one processor is responsible for extracting all possible K-mers from a large sequence and, then recording their positions from the sequence in the table, but improves at higher np as the number of processors increases and the sequence indexing is divided between more nodes. However, it easily performs better than other algorithms in the non-overlap technique used in many alignments.

Similarly, Figure 9 shows a comparison of the time consumed to index the data for the PSALR algorithm with the other three algorithms, which is better in both modes.

Table 4. Comparison of memory and time consumption of the proposed algorithm with other algorithms with different numbers of processors.

Input	Algorithm	Memory (MB)				Time(sec)			
		Np=2	Np=4	Np=8	Np=16	Np=2	Np=4	Np=8	**Np=16**
DB1	Overlap K=11	8135	3189	1495	811	690	260	180	**86**
	Non-overlap K=11	1638	697	390	243	93	49	40	**23**
	Overlap K=20	37580	13004	5680	2690	840	320	140	**87**
	Non-overlap K=20	3535	1208	530	252	87	48	45	**39**
	STAR	22835	22016	19968	19660	1680	1020	480	**360**
	BFAST	7579	7688	7688	7870	4200	3300	2520	**1860**
	SHRiMP	12697	12697	12700	12710	7500	6000	4560	**3540**
DB2	Overlap K=11	2475	985	586	358	156	60	43	**26**
	Non-overlap K=11	608	290	160	89	28	13	11	**11**
	Overlap K=20	8908	3396	1573	751	223	65	34	**24**
	Non-overlap K=20	998	344	152	76	22	15	12	**12**
	STAR	6907	5741	5288	5658	1200	660	480	**360**
	BFAST	1965	2115	2113	2372	1980	720	540	**300**
	SHRiMP	4517	4508	4532	4633	8340	3900	2640	**1740**
DB3	Overlap K=11	4695	1795	916	556	360	480	248	**128**
	Non-overlap K=11	982	483	267	161	60	30	25	**24**
	Overlap K=20	15974	7773	2808	1600	1380	454	216	**110**
	Non-overlap K=20	2101	742	299	150	46	30	24	**24**
	STAR	14848	14028	10752	11366	2220	1020	600	**420**
	BFAST	4681	4425	4478	4409	4680	1920	1900	**720**
	SHRiMP	8766	8750	8749	8744	9420	5700	3180	**2880**

According to Table 4, the PSALR algorithm does not perform best in terms of memory consumption for the entire length of K-mers due to the increase in length and number of K-mers. As mentioned before, other algorithms try to improve it by keeping the K-mer length constant by default or creating multi-level indexes.

The proposed algorithm does not work better without these techniques only when fewer processors are used, and work is divided between fewer processors, but it works better in terms of time consumed in all modes.

One of the most important criteria for aligners is the accuracy of mapping, which can be greatly increased by using the overlap technique. However, this technique requires a lot of memory, and one of the problems with the hash method is the amount of memory consumed during the indexing phase. Therefore, in this study, attempts were made to achieve improvements in memory and speed by providing solutions in the processes of indexing and mapping of alignment so that precision and accuracy are not lost but increased. In this technique, three datasets with different sizes and short and long queries are used, and the proposed algorithm is evaluated and compared with the overlap and technique and overlapping queries. It is found that memory and time consumption are reduced several times less in the second mode, but accuracy is reduced for the reasons stated. The PSALR algorithm is also compared and evaluated with the other three algorithms, and it is shown that it performs much better even with the overlap technique with a high number of processors and does not perform better only in some cases, such as k = 17 with a low number of processors that the work is divided little.

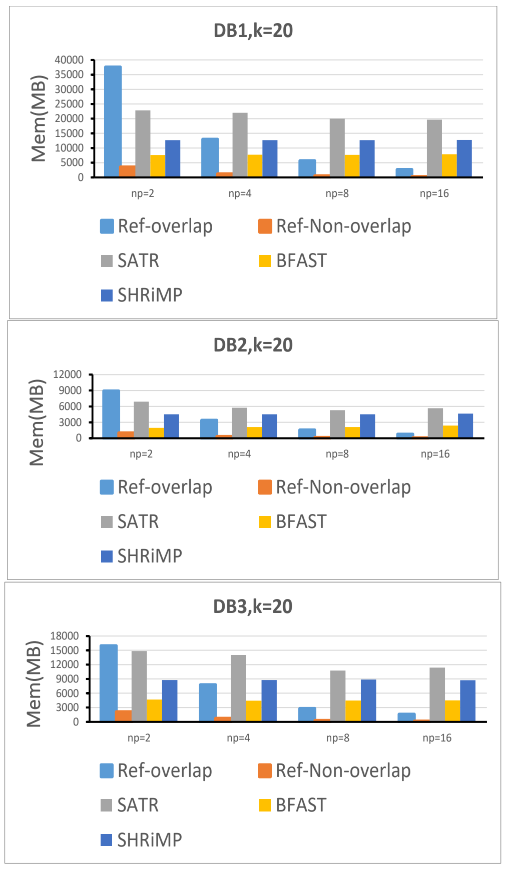

Figure 8. Comparison of memory consumption of the PSALR.

6. Conclusion and Feature Work

Bioinformatics, through the analysis of biological data such as genetic and protein sequences, plays a key role in identifying the genetic factors of diseases, developing personalized treatments, and accelerating processes of diagnosis and prevention[55]. Due to the exponential growth of DNA sequences, the process of searching for and storing sequences in databases has become increasingly time and memory-intensive. As a result, there is a pressing need for efficient algorithms that can accelerate database searches while minimizing memory usage. The SSAHA algorithm, renowned for its efficiency and speed when compared to seed-and-extend-based techniques like BLAST, achieves rapid searches in large databases by eliminating the expansion and evaluation steps commonly found in hash-based methods. The primary objective of this study is to enhance the SSAHA algorithm by addressing memory management, indexing, and leveraging parallelism for multiple processors, all while improving accuracy and performance. To establish a comprehensive distributed system, PSALR incorporates parallelization techniques during the indexing and mapping phases, breaking down large sequences into shorter segments distributed across multiple nodes. Additionally, nodes autonomously relay their results to users without necessitating users to retrieve results, which significantly enhances both time and bandwidth efficiency.

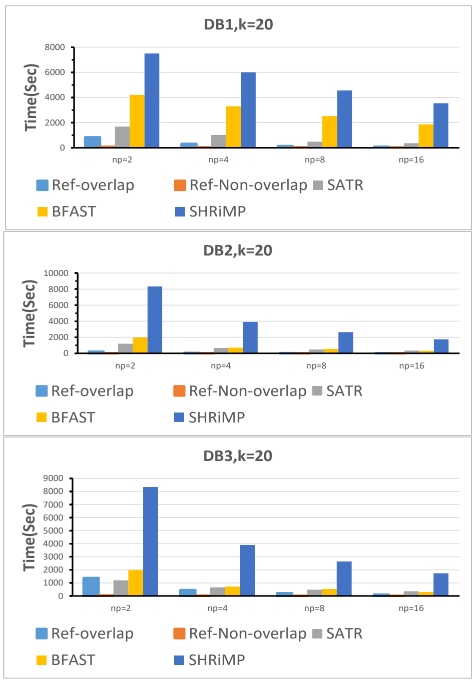

Figure 9. Comparison of time consumption of the PSALR algorithm with K-mer = 20.

To boost the algorithm's accuracy, two key techniques are employed. Firstly, the extraction of all K-mers from sequences with a window of length 1 ensures the inclusion of all K-mers in the alignment. Secondly, a hash table is constructed based on the K-mers in the text, as opposed to all possible K-mers of length k. The dynamic selection of K-mers of desired length by users reduces false positives and increases true positives while maintaining sensitivity. In the evaluation process, we compared the proposed algorithm to BWTs and Hash-based algorithms. Our findings indicate that it outperforms them in memory consumption under most circumstances, except when the distribution is zero or the number of processors is small. Nonetheless, there is still room for optimization in query sequence alignment. This algorithm necessitates sending all query sequences to each node, with each node aligning all sequences in the query file based on its index table and subsequently transmitting them to the output. While this approach doesn't increase memory consumption during alignment, future work should focus on further optimizing alignment to enhance overall performance.

References

[1] L. Hasan, Z. Al-Ars, and S. Vassiliadis, "Hardware acceleration of sequence alignment algorithms - an overview," in *Proc. of the International Conference on Design & Technology of Integrated Systems in Nanoscale Era (DTIS)*, 2007, pp. 92-97, IEEE.

[2] P. Bawono et al., "Multiple sequence alignment," in *Bioinformatics*, Springer, 2017, pp. 167-189.

[3] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443-453, 1970.

[4] J. D. G. De Herve et al., "A perceptual hash function to store and retrieve large scale DNA sequences," *arXiv preprint arXiv:1412.5517*, 2014.

[5] W. J. Wilbur and D. J. Lipman, "Rapid similarity searches of nucleic acid and protein data banks," *Proceedings of the National Academy of Sciences*, vol. 80, no. 3, pp. 726-730, 1983.

[6] J. Choi et al., "HIA: a genome mapper using hybrid index-based sequence alignment," *Algorithms for Molecular Biology*, vol. 10, no. 1, pp. 1-9, 2015.

[7] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing," *Briefings in Bioinformatics*, vol. 11, no. 5, pp. 473-483, 2010.

[8] S. Bandyopadhyay and R. Mitra, "A parallel pairwise local sequence alignment algorithm," *IEEE Transactions on NanoBioscience*, vol. 8, no. 2, pp. 139-146, 2009.

[9] F. Mozafari et al., "Speeding up DNA sequence alignment by optical correlator," *Optics & Laser Technology*, vol. 108, pp. 124-135, 2018.

[10] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754-1760, 2009.

[11] R. Li et al., "SOAP: short oligonucleotide alignment program," *Bioinformatics*, vol. 24, no. 5, pp. 713-714, 2008.

[12] 1 B. Langmead, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, article R25, 2009.

[13] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2," *Nature Methods*, vol. 9, no. 4, pp. 357-359, 2012.

[14] S. Misra et al., "Anatomy of a hash-based long read sequence mapping algorithm for next generation DNA sequencing," *Bioinformatics*, vol. 27, no. 2, pp. 189-195, 2010.

[15] J. C. Mu et al., "Fast and accurate read alignment for resequencing," *Bioinformatics*, vol. 28, no. 18, pp. 2366-2373, 2012.

[16] S. F. Altschul et al., "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403-410, 1990.

[17] B. Ma, J. Tromp, and M. Li, "PatternHunter: faster and more sensitive homology search," *Bioinformatics*, vol. 18, no. 3, pp. 440-445, 2002.

[18] Z. Ning, A. J. Cox, and J. C. Mullikin, "SSAHA: a fast search method for large DNA databases," *Genome Research*, vol. 11, no. 10, pp. 1725-1729, 2001.

[19] F. J. Sedlazeck, P. Rescheneder, and A. Von Haeseler, "NextGenMap: fast and accurate read mapping in highly polymorphic genomes," *Bioinformatics*, vol. 29, no. 21, pp. 2790-2791, 2013.

[20] S. Canzar and S. L. Salzberg, "Short read mapping: An algorithmic tour," *Proceedings of the IEEE*, vol. 105, no. 3, pp. 436-458, 2017.

[21] H. Mohamadi et al., "ntHash: recursive nucleotide hashing," *Bioinformatics*, vol. 32, no. 22, pp. 3492-3494, 2016.

[22] T. D. Wu, "Bitpacking techniques for indexing genomes: II. Enhanced suffix arrays," *Algorithms for Molecular Biology*, vol. 11, pp. 1-16, 2016.

[23] D. Geng et al., "The implementation of KMP algorithm based on MPI+ OpenMP," in *Proc. of the 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2012, IEEE.

[24] C. S. Kouzinopoulos, P. D. Michailidis, and K. G. Margaritis, "Performance study of parallel hybrid multiple pattern matching algorithms for biological sequences," in *Proc. of the International Conference on Bioinformatics Models, Methods and Algorithms*, 2012, SCITEPRESS.

[25] H. Li et al., "A fast CUDA implementation of agrep algorithm for approximate nucleotide sequence matching," in *Proc. IEEE 9th Symposium on Application Specific Processors (SASP)*, 2011, IEEE.

[26] Q. Xue, J. Xie, and J. S., "A parallel algorithm," in *Proc. 2014 International Conference on Information Science, Electronics and Electrical Engineering*, 2014.

[27] M. J. Chaisson and G. Tesler, "Mapping single molecule sequencing reads using basic local alignment with successive refinement (BLASR): application and theory," *BMC Bioinformatics*, vol. 13, p. 238, 2012.

[28] D. Peters, K. Qiu, and P. Liang, "Faster short DNA sequence alignment with parallel BWA," in *AIP Conference Proceedings*, 2011, American Institute of Physics.

[29] S. M. Rumble et al., "SHRiMP: accurate mapping of short color-space reads," *PLoS Comput Biol*, vol. 5, no. 5, e1000386, 2009.

[30] M. David et al., "SHRiMP2: sensitive yet practical short read mapping," *Bioinformatics*, vol. 27, no. 7, pp. 1011-1012, 2011.

[31] R. AlSaad, Q. Malluhi, and M. Abouelhoda, "Efficient parallel implementation of the SHRiMP sequence alignment tool using MapReduce," in *Qatar Foundation Annual Research Forum Volume 2012 Issue 1*, 2012, Hamad bin Khalifa University Press (HBKU Press).

[32] C.-M. Liu et al., "SOAP3: ultra-fast GPU-based parallel alignment tool for short reads," *Bioinformatics*, vol. 28, no. 6, pp. 878-879, 2012.

[33] P. Klus et al., "BarraCUDA—a fast short read sequence aligner using graphics processing units," *BMC Research Notes*, vol. 5, no. 1, p. 27, 2012.

[34] Y. Liu, B. Schmidt, and D. L. Maskell, "CUSHAW: a CUDA compatible short read aligner to large genomes based on the Burrows–Wheeler transform," *Bioinformatics*, vol. 28, no. 14, pp. 1830-1837, 2012.

[35] T. Pan et al., "Kmerind: A flexible parallel library for K-mer indexing of biological sequences on distributed memory systems," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 16, no. 4, pp. 1117-1131, 2019.

[36] A. M. Esmat et al., "A parallel hash-based method for local sequence alignment," *Concurrency and Computation: Practice and Experience*, vol. 2021, article e6568, 2021.

[37] H. Lin et al., "Efficient data access for parallel BLAST," in *Proc. 19th IEEE International Parallel and Distributed Processing Symposium*, 2005, IEEE.

[38] M. Nowicki, D. Bzhalava, and P. BaŁa, "Massively parallel implementation of sequence alignment with basic local alignment search tool using parallel computing in Java library," *Journal of Computational Biology*, vol. 25, no. 8, pp. 871-881, 2018.

[39] D. Dechev and A. Tae-Hyuk, "Using SST/Macro for effective analysis of MPI-based applications: Evaluating large-scale genomic sequence search," *IEEE Access*, vol. 1, pp. 428-435, 2013.

[40] T. Vijayaraghavan, A. Rajesh, and K. Sankaralingam, "MPU-BWM: Accelerating sequence alignment," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 179-182, 2018.

[41] H. Martinez et al., "Concurrent and accurate short read mapping on multicore processors," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 12, no. 5, pp. 995-1007, 2015.

[42] J. W. Kim, E. Kim, and K. Park, "Fast matching method for DNA sequences," in *Proc. International Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, 2007, Springer.

[43] A. Dobin et al., "STAR: ultrafast universal RNA-seq aligner," *Bioinformatics*, vol. 29, no. 1, pp. 15-21, 2013.

[44] Y. Chen, S. Yu, and M. Leng, "Parallel sequence alignment algorithm for clustering system," in *Proc. International Conference on Programming Languages for Manufacturing*, 2006, Springer.

[45] N. Homer, B. Merriman, and S. F. Nelson, "BFAST: an alignment tool for large scale genome resequencing," *PloS One*, vol. 4, no. 11, e7767, 2009.

[46] X. Yu and X. Liu, "Mapping RNA-seq reads to transcriptomes efficiently based on learning to hash method," *Computers in Biology and Medicine*, vol. 116, p. 103539, 2020.

[47] F. Peng et al., "New hash-based sequence alignment algorithm," in *Proc. 2nd International Conference on Bioinformatics and Intelligent Computing*, 2022.

[48] A. Joudaki et al., "Aligning distant sequences to graphs using long seed sketches," *Genome Research*, 2023, article gr.277659.123.

[49] H. Zhang et al., "ESA: An efficient sequence alignment algorithm for biological database search on Sunway TaihuLight," *Parallel Computing*, vol. 117, p. 103043, 2023.

[50] K. Xu, X. D. Kai, A. Müller, R. Kobus, B. Schmidt, and W. Liu, "FMapper: Scalable read mapper based on succinct hash index on SunWay TaihuLight," *Journal of Parallel and Distributed Computing*, vol. 161, p. 11, 2022.

[51] S. Suchindra, "New sequence alignment algorithm using AI rules and dynamic seeds," *Bioscience & Engineering: An International Journal (BIOEJ)*, vol. 10, no. 1/2, 2023.

[52] G. Greenberg, A. N. Ravi, and I. Shomorony, "LexicHash: sequence similarity estimation via lexicographic comparison of hashes," *Bioinformatics*, 2023, article btad652.

[53] M. Zaharia et al., "Faster and more accurate sequence alignment with SNAP," *arXiv preprint arXiv:1111.5572*, 2011.

[54] S. Canzar and S. L. Salzberg, "Short read mapping: An algorithmic tour," *Proceedings of the IEEE*, vol. 105, no. 3, pp. 436-458, 2015.

[55] M. Shamsollahi, A. Badiee, and M. Ghazanfari, "Using combined descriptive and predictive methods of data mining for coronary artery disease prediction: A case study approach," *Journal of AI and Data Mining*, vol. 7, no. 1, pp. 47-58, 2019.

ترازبندی موازی توالی برای خوانش‌های طولانی با مدل مبتنی بر هش(PSALR)

نسرین آقایی میبدی[۱]، امین نظارات[۲]، سیما عمادی[۲،*] و محمدرضا غفاری[۴]

[۱] گروه کامپیوتر ، واحد یزد، دانشگاه آزاد اسلامی، یزد ، ایران .

[۲] گروه کامپیوتر ، دانشگاه شیراز ، شیراز ، ایران .

[۳] گروه کامپیوتر ، واحد یزد، دانشگاه آزاد اسلامی، یزد ، ایران .

[۴] گروه زیست‌شناسی سامانه‌ای، مؤسسه تحقیقات بیوتکنولوژی کشاورزی ایران، سازمان تحقیقات، آموزش و ترویج کشاورزی، کرج، تهران، ایران.

ارسال ۲۰۲۴/۰۵/۰۶؛ بازنگری ۲۰۲۴/۱۰/۰۲؛ پذیرش ۲۰۲۴/۱۱/۲۴

چکیده:

ترازبندی توالی و نگاشت ژنوم با چالش‌های متعددی مواجه است که عمدتاً بر سرعت و نیاز به فضای ذخیره‌سازی توالی‌های نگاشته‌شده متمرکزند. با افزایش حجم داده‌های توالی‌یابی DNA، توسعه روش‌های ترازبندی کارآمد که ضمن کاهش نیازهای ذخیره‌سازی، امکان ترازبندی سریع را نیز فراهم کنند، ضروری است. در این پژوهش، الگوریتم ترازبندی موازی توالی با مدل مبتنی بر هش (PSALR) معرفی می‌شود که به‌طور ویژه برای افزایش سرعت ترازبندی، بهینه‌سازی فضای ذخیره‌سازی و حفظ حداکثر دقت طراحی شده است. در مقایسه با الگوریتم‌هایی نظیر BLAST، PSALR با استفاده از جداول هش، داده‌ها را کارآمدتر نمایه‌سازی کرده و در نتیجه بار محاسباتی و زمان پردازش را کاهش می‌دهد. این الگوریتم با فشرده‌سازی داده، بسته‌بندی آن در ابعاد پهنای‌باند مرسوم و توزیع داده میان گره‌های مختلف، زمان انتقال و حافظه مورد نیاز را کم می‌کند. پس از دریافت داده‌های فشرده، گره‌ها بدون نیاز به بازگشایی و رمزگشایی قادر به جستجو و نقشه‌برداری هستند. نوآوری دیگر PSALR در تقسیم توالی‌های بزرگ به زیرتوالی‌ها و ارسال آن‌ها به گره‌هاست؛ رویکردی که محدودیتی در طول کوئری ایجاد نکرده و نتایج ارزیابی را بدون نیاز به گره مرکزی، مستقیماً به کاربر بازمی‌گرداند. همچنین، استفاده از زیرتوالی‌های همپوشان در توالی کوئری و مرجع تضمین می‌کند که تمامی زیرتوالی‌های ممکن توالی هدف مورد جستجو و نگاشت قرار گیرند. آزمون‌های عملکردی نشان می‌دهد PSALR نسبت به همتایان خود کارایی بهتری داشته و راهکاری امیدبخش برای ترازبندی کارآمد توالی و نگاشت ژنوم است.

کلمات کلیدی: نمایه‌سازی، مبتنی بر هش، ترازبندی توالی، نگاشت، MPI.