HARDWARE ACCELERATOR FOR MULTIPLE SEQUENCE ALIGNMENT OF DNA SEQUENCES

DUVERNEY CORRALES MENDOZA

This thesis has been submitted in fulfilment of the requirements for a:

Master's Degree in Engineering with Emphasis in Electronic Engineering

at Universidad del Valle

Universidad del Valle - Engineering Faculty

Electrical and Electronic Engineering School

Santiago de Cali, December of 2018

HARDWARE ACCELERATOR FOR MULTIPLE SEQUENCE ALIGNMENT OF DNA SEQUENCES

DUVERNEY CORRALES MENDOZA

This thesis has been submitted in fulfillment of the requirements for a:

Master's Degree in Engineering with Emphasis in Electronic Engineering

at Universidad del Valle

Approved:

Vladimir Trujillo Olaya, PhD Professor at the Universidad de San Buenaventura	Juan Manuel Marmolejo, PhD Post-Doctorate Research at the Pontificia Universidad Javeriana
Jaime Velasco Medina, PhD Director	Sandra Esperanza Nope Rodriguez Coordinator of Master's in Engineering with Emphasis in Electronic Engineering

Universidad del Valle - Engineering Faculty

Electrical and Electronic Engineering School

Santiago de Cali, December of 2018

This work is dedicated to:

The mother Nature (Or God?), because it has provided the inspiration and help for reaching this work. The Light, because it always has been for guide me and enlighten me in everything I do, Everything… Also, to the wind, the water, the colors, the sounds and my hyper-noised mind (but sometimes I cannot stand it).

Acknowledgements

Firstly, I want to thank God for allowing my half spermato-Ying to seduce my half ovule-Yang resulting this precious creature, and my parents to provide them. On the whole, they are the basics and foundations of what I am; I love them.

Honestly, the most problematic part of this research work was to fight with my very fluctuating emotional component. It wasted a lot of time (Three well-defined psychotic episodes) and motivation, it was a very Hard for me. However, my family supported me a lot, like Professors Jaime Velasco and Mario Vera, the EIEE directives, Claudia, Juan Manuel, Melusita, Carlos, Jorge D, Jorge G, my UWH team, among others. I am very grateful to all of them.

I want to thank the Universidad del Valle for giving me the opportunity to be a professor (Teaching assistant). Teaching is one of the things that I like the most and that is one of the reasons why I studied my master's research at this university. I learned a lot in these two-years as Professor and I had many bright students (Really); I hope I have contributed my sand bit.

I want to especially thank Professor Jaime Velasco, director of this research, for understand me and support me, for bringing me on the bioinformatics way; I have skills for engineering, but I always dream to be a biologist, and I do both things in this research area, just a little. Also for demanding and challenging me, for trying to get the best of me.

I also want to give special thanks to Professor Mario Vera, my undergraduate director, to him I owe my digital design skills. I also consider him a great friend and one of the best conversationalists that I know.

Finally, I want to thank my friends in general, and mana fountains like Daniela, Nicole, Melusita, May, Estefania, Catalina, Natalia, Jess... who somehow filled my life with magic. Life is always more beautiful when there is magic.

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCCION & MOTIVATION

1.1. Motivation

Multiple Sequence Alignments (MSA) is a process for aligning multiple related sequences, allowing to reveal more biological information than many Pairwise Sequence Alignment (PSA) can, for example, phylogenetic analysis and prediction of protein structures. Theoretically, optimal MSA is possible but is not practical since it has a complexity $O(L^N)$, where L is the average sequence length and N is the number of sequences. PSA is a particular case for two-sequences and has a complexity $O(L^2)$ for same length sequences. In practical, heuristic MSA methods are used based on critical tasks as PSA and profile-profile alignments.

Usually, these MSA tasks require a high computational cost, time processing, and memory resources because they are performed using high-performance platforms. Nowadays, the computational cost for performing both PSA and profile-profile alignments limit the use to short sequences. For example, for aligning 100 sequences using ClustalW, the most popular MSA algorithm, are necessary to perform around 5000 PSA and 100 profile-profile alignments.

Ideally, both tasks require hardware acceleration, considering that the other MSA task can be solved efficiently in software. Then, to reach it, hardware accelerators based on systolic arrays allow increasing the performance. However, they are restricted both sequence length and number of PEs, being necessary to find a balance between them to obtain useful results. Modular arithmetic is an ideal solution to address this issue. However, the comparison “smaller than”, used in NWA and SWA, are not defined in modular space, being an obstacle to reach it. Nevertheless, we found a way to solving this issue by performing the comparisons in modular space when “the maximum difference between the compare data is fixed”. Therefore, this work was addressed to improve the optimal PSA by increasing the performance and the length of the sequence that can be processed and reducing the development costs.

1.2. Contributions

In this work, we design two processors for performing the optimal semi-global pairwise sequence alignment using the K-Band method, for both linear gap (Needleman-Wunsch) and affine gap (Gotoh). Each designed processor uses new systolic arrays and PE architectures. An embedded system was developed to verify the designed processors, and it was integrated on a low-cost SoC-FPGA platform. A new co-design algorithm performs the forward process using the designed processors, while the traceback process was performed using a new software algorithm.

These systolic array designs allow maximizing the PE density, perform pair-sequences of any length, process sequences by packets and simplify the control tasks. To accomplish the above four particular designs are necessary to solve the issues. The first one performs “smaller than” comparisons in modular space when the maximum difference between the data is fixed. The second one limits the processing space to the band using dynamic very close values in the edge of the band which is mainly useful for using modular arithmetic. The third one performs the initialization functions of the NWA. The last one detects the begin and the end of the alignment.

1.3. Organization

This work is composed by other six chapters, organized as follows:

Chapter 2 introduces a theoretical background of bioinformatics, focused on sequence alignment: Pairwise and Multiple sequence alignment.

Chapter 3 exposes a proposed method for processing the Needleman-Wunsch algorithm using arithmetic mod-2^N.

Chapter 4 describe the proposed method for performing the local pairwise sequence alignment, forward and traceback process, using the K-Band method and arithmetic mod-2^N.

Chapter 5 presents hardware design details of the streaming processor and synthesis results.

Chapter 6 presents verification results of the streaming processor using a co-design solution on a low-cost SoC-FPGA.

Chapter 7 exposes the conclusions of this work and the proposed future works.

Chapter 2

THEORETICAL BACKGROUND

2.1. Bioinformatics and Molecular Biology

Bioinformatics is an interdisciplinary field, as shown in Figure 21, that develops and applies computational methods for storing, retrieving, organizing and analyzing large collections of biological data, such as genetic sequences, cell populations or protein samples, which allow making new predictions or discover new biology. The computational methods used include analytical methods, mathematical modeling and simulation.

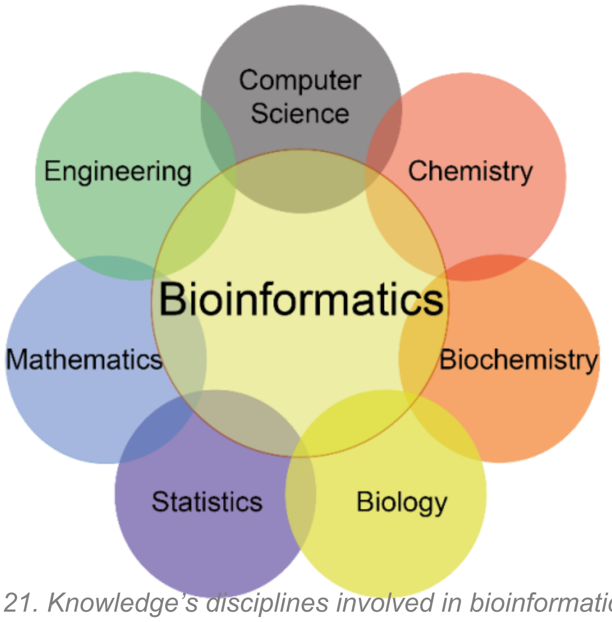


Figure 21. Knowledge's disciplines involved in bioinformatics [1].

The main topics in bioinformatics from a journal perspective are genome analysis, sequence analysis, phylogenetic, gene expression, genetics and populations, systems biology, data and text mining, databases and ontologies, bioimages and structural bioinformatics.

Bioinformatics differs from a related field known as computational biology [2]. Bioinformatics is limited to sequence, structural and functional analysis of genes and genomes and their corresponding products and is often considered computational molecular biology. However, computational biology encompasses all biological areas that involve computation. For example, mathematical modeling of ecosystems, population dynamics, application of the game theory in behavioral studies, and phylogenetic construction using fossil records all employ computational tools but do not necessarily involve biological macromolecules.

2.2. Sequence Alignment (SA)

Sequence Alignment is an essential task for sequence analysis in bioinformatics, and it is the process by which sequences are compared by searching for common character patterns and establishing residue-residue correspondence among related sequences. SA is used to identify regions of similarity between biological sequences (DNA, RNA or proteins), allowing to find functional, structural and evolutionary relationships. Usually, high sequence similarity implies significant functional or structural similarity [3]. SA of two sequences is called Pairwise Sequence Alignment (PSA), and Multiple Sequence Alignment (MSA) for three or more sequences.

2.2.1.Similarity vs Distance

Both similarity and distance are opposed string metrics for measuring the difference between two sequences considering single-symbol edits: insertions, deletions and substitutions. Considering a gapped sequence alignment pair, the distance corresponds to the number of editions for becoming a sequence into the other, while the similarity also uses the sequence length of the gapped alignment. The similarity metric is calculated using Equation 1 and the edit distance (Or Levenshtein), the most basic distance metric, is calculated using Equation 2.

(1)

(2)

2.2.2.Homology vs Similarity

Homology is a qualitative statement: two sequences are homologous or nonhomologous. Two sequences share homology or have homologous relationships when they descend from a common evolutionary origin. On another hand, sequence similarity is the percentage of aligned residues that are similar in physicochemical properties such as size (small, very small and large), charge (negative and positive), hydrophobicity (polar and nonpolar), aromatic and aliphatic. The similarity is useful to determinate the homology. For example, sequences A and B have 85% similarity. Therefore, they are homologous.

2.2.3.Similarity vs Identity

Sequence similarity and sequence identity are synonymous for nucleotide sequences. However, for protein sequences the two concepts are different. Identity refers to the percentage of matches of the “same” amino-acid residues, for example, Tyr is identical to Tyr. Similarity refers to the rate of matches of aligned residues that have similar physicochemical characteristics and can be more readily substituted for each other, for example, Tyr and Phe are similar because both are aromatic amino-acids.

2.2.4.Types of Alignment: Global vs Local

There are two alignment strategies often used: Global Alignment and Local Alignment [2], [3]. Global alignment aligns two sequences assuming that they are generally similarity over their entire length. Alignment is carried out from beginning to end of both sequences. Therefore, this method is more applicable to align two closely related sequences of roughly the same length. Local alignment does not consider the assumption of global similarity. It only finds local regions with the highest level of similarity and aligns these regions without regard for the alignment of the rest of the sequence regions. This approach is used to search conserved patterns in divergent sequences.

2.3. Pairwise Sequence Alignment (PSA)

PSA is the most essential task for sequence analysis and one of the most important in bioinformatics since it allows finding if two sequences are related. PSA is the basis of database similarity searching and MSA and it is used in processes as sequencing, phylogenetic trees and database searching [2], [3]. The overall goal of PSA is to find the best pairing of two sequences, such that there is maximum correspondence among residues. To achieve this goal, one sequence segments are relatively shifted by introducing gaps to maximize the number of matches, where the gaps represent evolution events of insertion/deletion. Alignment algorithms both global and local are fundamentally similar. Both types of algorithms can be based on one of the three methods: Dot Matrix, Dynamic Programming and Words.

2.3.1.Dot Matrix

The dot matrix method is a graphical way to compare two sequences in a two-dimensional matrix. If a residue match is found, a dot is placed within the graph. Diagonal lines of adjacent dots reveal regions of similarity. Parallel diagonal lines represent repetitive regions (…ACTACTACT…), as shown in

Figure 22. Dot matrix is used to identify internal repeat regions, self-complementary (inverted repeats) of DNA sequences, for identifying secondary structures and comparing gene order conservation between two closely related genomes.

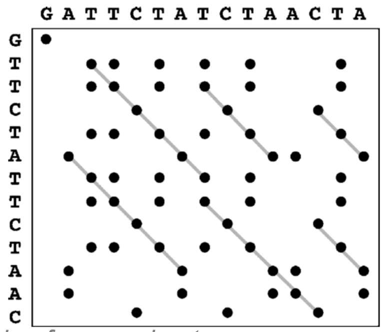


Figure 22. Example of comparing two sequences using dot plots [2].

2.3.2.Dynamic Programming (DP)

This method is fundamentally similar to the dot matrix method, aligning the sequences using a two-dimensional matrix (Similarity Matrix). However, it allows finding optimal alignments in a more quantitative way using a scoring scheme, which is composed of a score matrix and gap penalties. For DNA, the simplest score matrix is determinate by the parameters match and mismatch as shown the Equation 3.

(3
)

This method is composed of two processes: forward and traceback. In the forward process, the scores of one row of the similarity matrix are calculated at a time: horizontal, vertical or diagonal row. The scores of each row are calculated taking into account the scores already obtained from the two previous rows. The best score of each cell is selected, and an arrow (diagonal, up or left) indicates the source path. This process is repeated until all cell values will be calculated. In the traceback process, the optimal alignment is done by tracing back the similarity matrix, following the path determinates by the arrows (best score path). If two or more paths have the same highest score, one is chosen arbitrarily to represent the best alignment.

The classical DP-based algorithms for PSA are the Smith-Waterman Algorithm (SWA) [4] for local alignment and the Needleman-Wunsch Algorithm (NWA) [5] for global alignment, presented in the Equation 4 y 5. Figure 23 shows an example of the NWA.

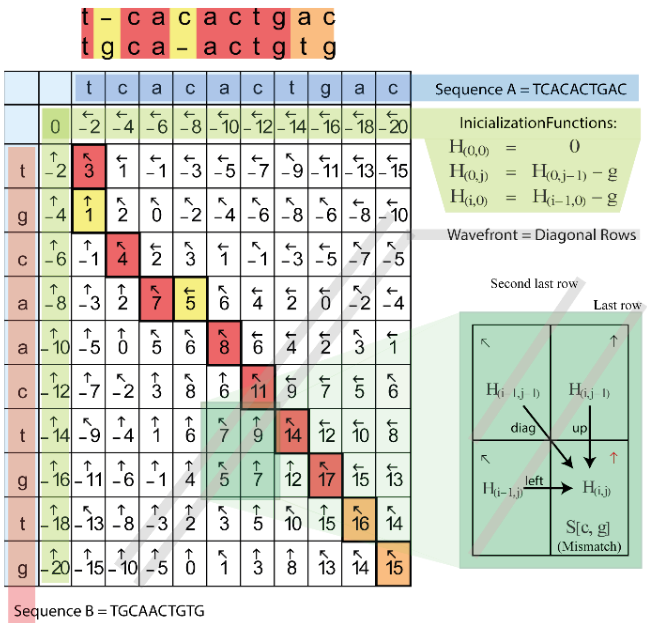


Figure 23. PSA using NWA. Match=3, Mismatch=-1, Gap penalty=-2. [6]

(4)

(5)

These algorithms have been enhanced by using the affine-gap scoring scheme [7], which uses different cost for an open gap or an extended gap to improve the gap distribution on biological PSA. Affine gap uses three matrices {H, G_X, G_Y} (Equation 6), and can be simplified to two matrices {H, G} (Equation 7) for the case when mismatch≥2e [2], [3].

(6)

(7)

2.3.3.Words

This method works by finding short stretches of identical or nearly identical two sequences [2] because the two related sequences must have at least one word in common. By identifying word matches, the full alignment is obtained by extending the similarity regions into de words, as shown in Figure 24. This heuristic method is not guaranteed to find the optimal alignment or true homologous but is 50-100 times faster than DP. Algorithms for database searching as BLAST and FASTA use this method.

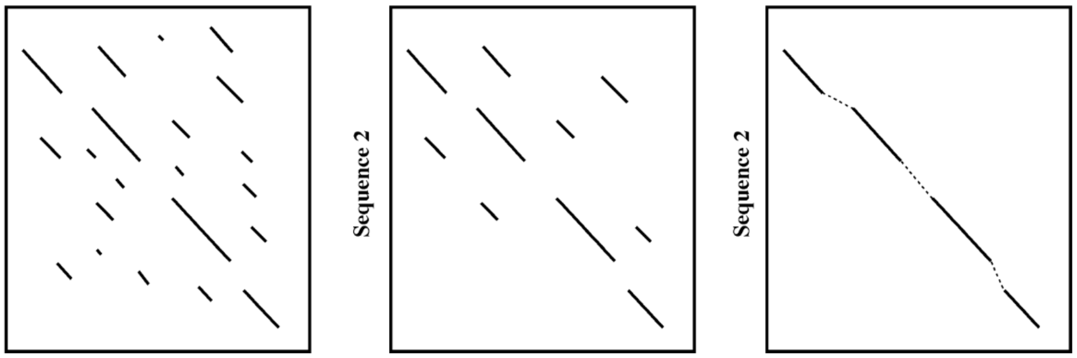


Figure 24. PSA based on K-mer method [2].

2.3.3.1. K-mer

The main PSA algorithm based on words is the k-tuple o k-mer algorithm [8]. The position of all k-tuples matches is related by using a hash table, as shown in Figure 25. The alignment is found considering the diagonals with the most matches.

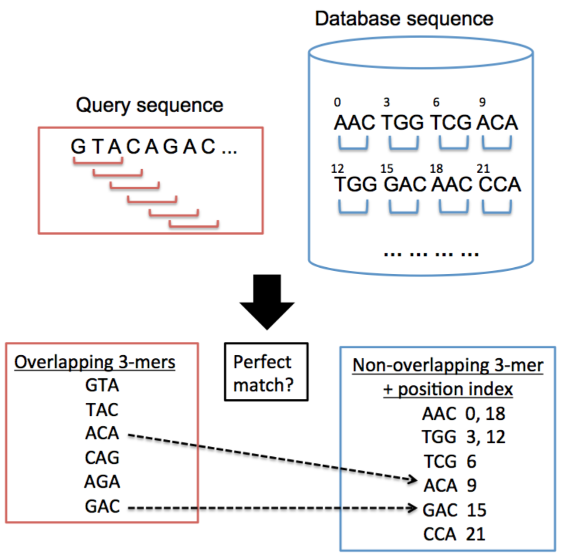


Figure 25. PSA based on K-mer method [9].

2.3.4. Enhanced Methods

The two main methods for increasing the performance of the PSA methods are Divide and Conquer (D&C) and K-Band. Divide and Conquer method is used to reduce the matrix zone for processing the PSA, aligning only into optimal points [10], [11], as shown in Figure 2 6. Optimal points correspond to base-pair alignments which are sure to be included in the alignment, and they are especially useful in NWA, which is based on the optimal point concept: the first and last alignments (left-up and right-down corners). D&C introduces the wave-front symmetries, allowing the parallel process of all cells of a diagonal row (wave-front) for reducing the storage to only last two diagonal rows. K-Band [12], [13] method emerges as a simpler approach to D&C method since does not use optimal points. This method also uses wave-front symmetries, and it computes only a K-width band segment of the matrix H, assuming that the optimal alignment is located into this band [12], [14]–[16], as shown in Figure 2 6.

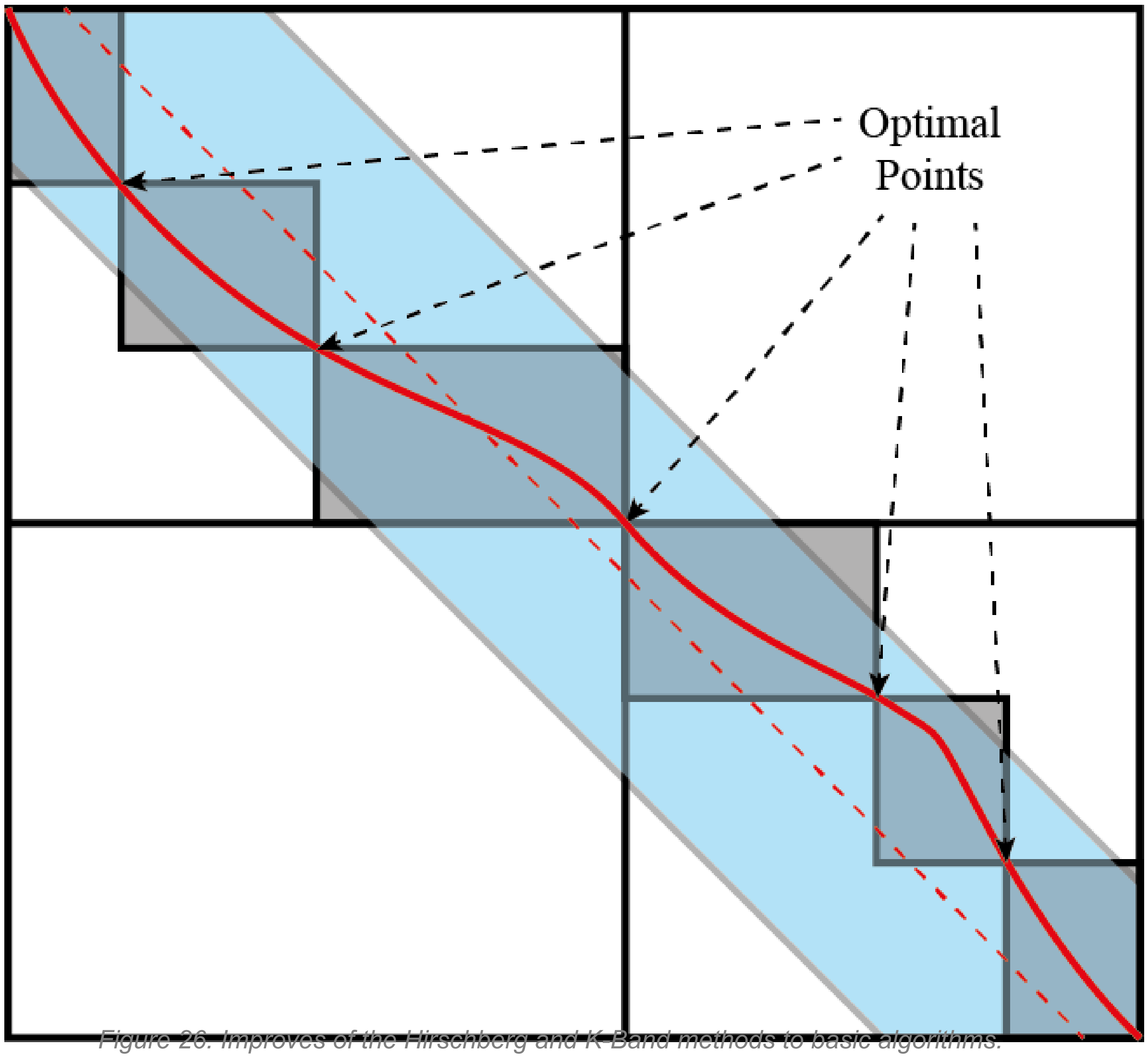


Figure 26. Improves of the Hirschberg and K-Band methods to basic algorithms.

2.4. Multiple Sequence Alignment (MSA)

MSA is a process for aligning multiple related sequences, allowing to reveal more biological information than many PSA can, for example, phylogenetic analysis and prediction of protein structures [2], [3]. Theoretically, optimal MSA is possible but is not practical since it has a complexity $O(L^N)$, where L is the average sequence length, and N is the number of sequences. In practice, heuristic methods are used based on multiple matching PSA, being necessary to convert numerous PSA into a single alignment, arranging the sequences in such a way that evolutionary equivalent positions (Guide tree) across all sequences are matched. Next, the MSA is performed using progressive, iterative and clustering strategies and profile alignment methods.

2.4.1.MSA alignment strategies: progressive, iterative and clustering

2.4.1.1. Progressive

Progressive alignment builds up a final MSA by combining pairwise alignments beginning with the most similar pair and progressing to the most distantly related according to a guide tree. The initial guide tree is determined by using methods such as neighbor-joining or UPGMA. Progressive methods are so strongly dependent on a high-quality initial alignment is the fact that these alignments are always incorporated into the final result, that is, once a sequence has been aligned into the MSA, its alignment is not considered further. This approximation improves efficiency at the cost of accuracy.

2.4.1.2. Iterative

Iterative alignment works similarly to progressive methods but repeatedly realign the initial sequences as well as adding new sequences to the growing MSA. Iterative methods can return to previously calculated pairwise alignments or sub-MSAs incorporating subsets of the query sequences. Iterative methods reduce the dependency respect to the initial alignments, unlike progressive methods.

2.4.1.3. Block-Based

Blocked MSA is a method for locating sequence motifs in global MSAs. In order to achieve a better MSA, firstly the blocks (conserved regions) are aligned, then the no-conserved regions between successive blocks are aligned until achieving the full alignment. Blocks can be generated from an MSA, or they can be extracted from unaligned sequences using a precalculated set of common motifs previously generated from known gene families.

2.4.2.MSA alignment methods: consensus, profile-profile and HMM

Figure 27 shows an example of MSA using a progressive strategy; the first alignments as S1-S3 and S4-S5 can be performed using PSA methods, and the result is called a profile. Profiles can be aligned with other profiles or with single sequences; for the last case, the profile is converted in a consensus sequence, then the MSA is performed using PSA methods, or the sequence is converted in a profile, then a profile-profile method is used.

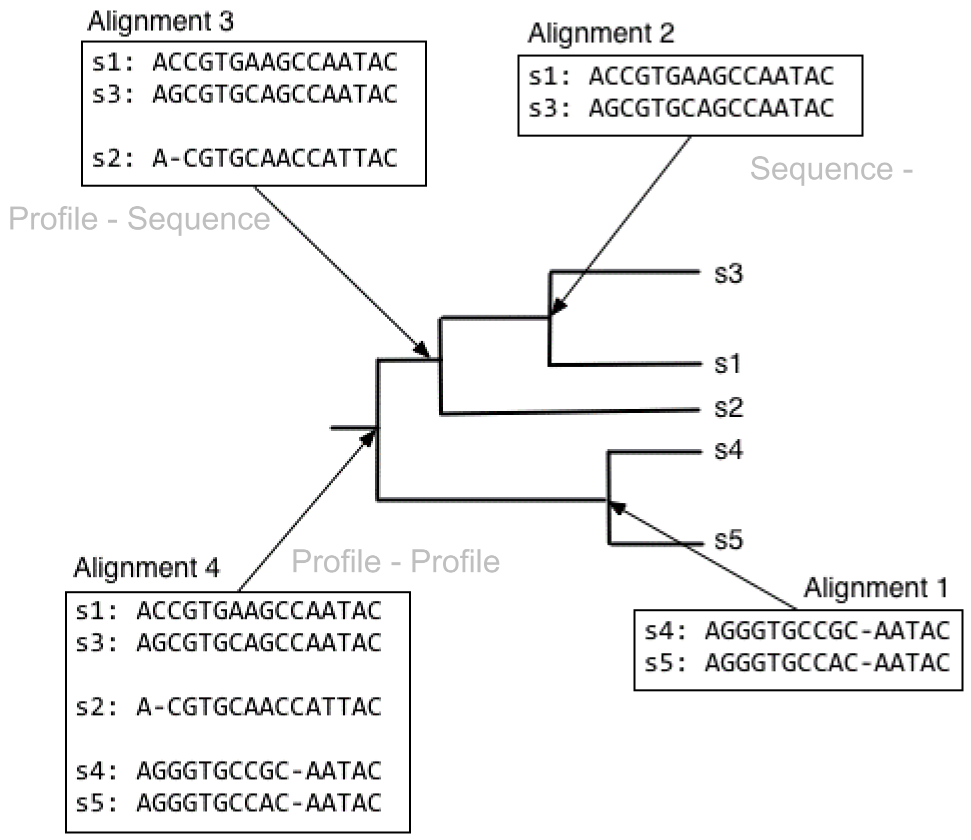


Figure 27. MSA example for 5 sequences [17].

2.4.2.1. Sequence-Sequence alignment: Consensus

The simplest method for performing MSA is using PSA tools. For reaching it first is necessary to obtain a new sequence that corresponds to the consensus between the aligned sequences of a profile, as shown in Figure 28. Then, the consensus sequences are aligned with others one using PSA methods. This method presents loss of information.

Seq 1	A	G	A	A	A	A	T	T	A	T	T	T	T	A	A	A	T	T	T	C	C	T
Seq 2	A	G	A	A	A	A	A	A	A	G	A	T	C	A	A	A	A	A	A	A	T	A
Seq 3	T	C	A	A	A	A	A	A	T	A	T	T	T	T	A	A	A	A	A	C	G	A
Seq 4	C	T	G	A	A	A	A	A	T	T	T	T	G	C	A	A	A	A	A	G	T	T

Consensus	n	n	A	A	A	A T	A T	T	A T	T	T	T	T	n	n	A	A	A	A	n	n	n

Figure 28. Example of a consensus sequence from a profile of 4-sequences.

2.4.2.2. Profile-profile alignment

Methods based on profiles are most sensitives since they do not have loss of information. A profile is built using a frequency table as shown in Figure 29.. Profiles are commonly normalized. Profiles are processed using methods as dot point [18] and Hidden Markov Models (HMM) [5], [19], [20], as shown in Figure 210.

Seq 1	A	G	A	A	A	A	T	T	A	T	T	T	T	A	A	A	T	T	T	C	C	T
Seq 2	A	G	A	A	A	A	A	A	A	G	A	T	C	A	A	A	A	A	A	A	T	A
Seq 3	T	C	A	A	A	A	A	T	A	T	T	T	T	A	A	A	A	A	A	C	G	A
Seq 4	C	T	G	A	A	A	A	A	T	T	T	T	G	C	A	A	A	A	A	G	T	T
A	2	0	3	4	4	4	3	3	2	1	1	0	0	2	4	4	3	3	3	1	0	2
C	1	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	2	1	0
T	1	1	0	0	0	0	1	1	2	2	3	4	2	1	0	0	1	1	1	0	2	0
G	0	2	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	1	1	2
_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29. Building of an MSA-profile

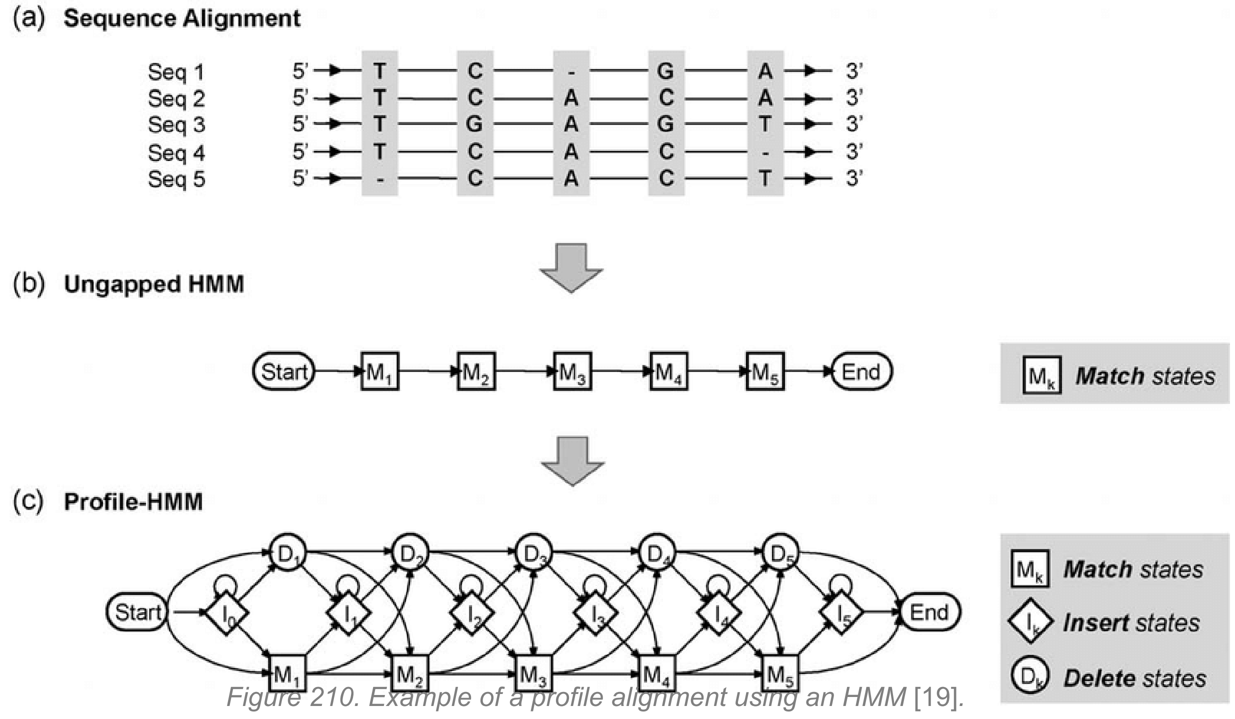


Figure 210. Example of a profile alignment using an HMM [19].

2.4.3. MSA alignment algorithms

In general, all MSA algorithms are based on PSA algorithms [21]–[23], mainly heuristic PSA algorithms because they are faster than optimal PSA. The most popular MSA algorithms are ClustalW [24], Clustal-Omega [25], T-COFFEE[26], MAFFT[27], Kalign[28], MUSCLE[29] and PRRN[2].

2.4.3.1. ClustalW and Clustal Omega

The most popular MSA algorithm is ClustalW (See Figure 211), introduced in 1994, which presented the best alignment quality, sensitivity and speed of its time. Firstly, the algorithm performs global PSAs of all the sequences using the k-tuple algorithm or the NWA. Second, a matrix is built with the similarity or distances of each pair of sequences. Third, a guide tree is built using the Neighbor-Joining algorithm. Finally, the MSA is performed by progressively aligning the most closely related sequences according to the guide tree constructed previously, using a full DP algorithm.

Clustal Omega (See Figure 212) is the latest MSA algorithm of the Clustal family, introduced in 2011. For large sequences, it outperforms other MSA algorithms in terms of processing time and overall alignment quality. Firstly, global PSA is performed using the k-tuple method (NWA is slower). Next, sequences are clustered using the mBed method, which mapping each Nth-sequence in a space n-dimensional, where n is proportional to Log[N]; then each sequence is replaced by a vector, where each element is the distance to one of n “reference sequences”. Next, these vectors are clustering quickly using the k-means++ method, which seeks to minimize the average squared distance between points of the same cluster. Next, the guide tree is built using the UPGMA method. Finally, the MSA is produced by aligning two Hidden Markov Models (HMM) instead of a profile-profile comparison, improving sensitivity and alignment quality significantly.

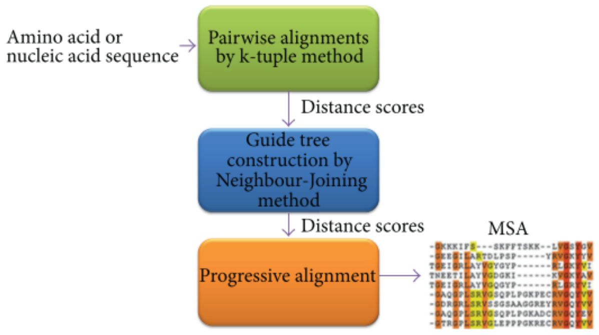

Amino acid or nucleic acid sequence
Pairwise alignments by k-tuple method
Distance scores
Guide tree construction by Neighbour-Joining method
Distance scores
Progressive alignment
MSA

Figure 211. ClustalW [21]

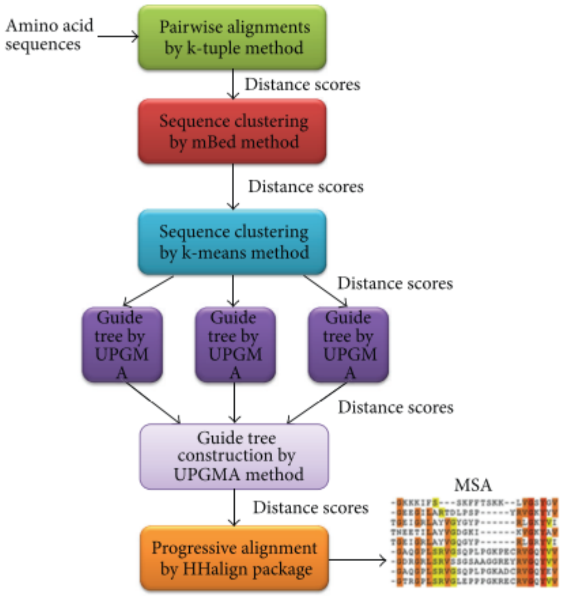

Amino acid sequences
Pairwise alignments by k-tuple method
Distance scores
Sequence clustering by mBed method
Distance scores
Sequence clustering by k-means method
Distance scores
Guide tree by UPGMA
Guide tree by UPGMA
Guide tree by UPGMA
Distance scores
Guide tree construction by UPGMA method
Distance scores
Progressive alignment by HHalign package
MSA

Figure 212. Clustal Omega [21]

2.4.3.2. T-Coffee

The tree-based consistency objective function for alignment evolution (T-Coffee) is an iterative MSA algorithm (See Figure 213) which it can combine signals from heterogeneous sources (e.g. sequence-alignment programs, structure alignments, threading, manual alignment, motifs and specific constraints) into a single consensus MSA. The main difference from traditional progressive alignment methods is that, instead of using a substitution matrix for aligning the sequences, a position-specific scoring scheme is used (the extended library). Thanks to the extension process, the values contained in the library for a given pair of sequences also depend on information from the other sequences in the set. In this way, errors are less likely to occur during early stages of the progressive alignment. T-Coffee increases 10% the accuracy of the MSA in comparison to ClustalW; however, it presents disadvantages such as weak scalability: can only align maximum 100 sequences without loss of accuracy.

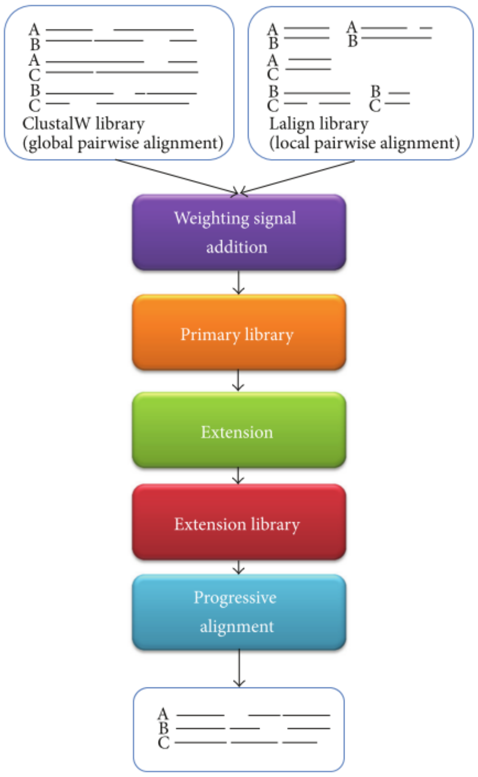


Figure 213. Block diagram of T-Coffee. [21]

2.4.3.3. MAFFT

MAFFT is an MSA algorithm based on the Fast Fourier Transform (FFT) for identifying homologous regions. In this method, the amino-acids are converted to a volume and polarity values, and a simplified scoring system to reduce the CPU time and increases the accuracy of alignments. MAFFT use the FFT-NS-2 method for calculating all pairwise distances quickly, building a provisional MSA and calculate refined distances from the MSA. Later, it uses FFT-NS-i method for refining the MSA which it is faster and less accurate than FFT-NS-2. This method allows scalability.

2.4.3.4. Kalign

Kaling is yet a good quality MSA algorithm. This follows a similar strategy to the standard progressive methods. The guide tree is built using UPGMA or Neighbor-Joining, based on a distance matrix obtained from PSA. In contrast, this algorithm differs in the use of Wu-Manber approximate string-matching algorithm used to align the profiles for calculating the distance, and in the DP based algorithm for profile aligning.

2.4.3.5. MUSCLE

The multiple sequence comparison by log-expectation (MUSCLE) uses two distance measures: k-mer distance for unaligned pairs of sequences and the Kimura distance for aligned pairs. Guide trees are produced using the UPGMA method. An initial MSA is produced by progressive alignment based on the guide tree. The initial guide tree is re-estimated using the UPGMA method and Kimura distance, which is more accurate but require of a previous alignment, producing a second guide tree. A second MSA is generated based on the second guide tree. Finally, the MSA is performed from the profiles of the first and second MSA.

2.4.3.6. PRRN

This method is a randomized iterative strategy for MSA based on hill-climbing strategies which use pairwise profile alignment to improve the overall weighted sum-of-pairs score each iterative step, where the pair weights are introduced to correct for uneven representations of the sequences to be aligned. The weights are fixed in the earlier (RIW) method, whereas the doubly-nested randomized iterative (DNR) method described in [8] tries to make mutually consistent the alignment, phylogenetic tree and pair weights. These two strategies can be selected by setting different options.

Chapter 3

Modular comparison: only under special conditions

Nowadays, sequence alignment processors designed using systolic array are limited for both Processor Elements (PE) density and sequence's length because the similarity matrix H (and gap matrix G for the affine gap) can reach very large values when large sequences are performed. Large values require a higher number of bits to represent them. For example, 25-bits are necessary to codify the signed value ±10'000.000, being necessary more logic resources to implement each PE. Modular arithmetic is an ideal solution to address this issue. However, the comparisons A<B used in NWA and SWA are not defined in modular space, preventing its use.

Previous work had tried to apply modular arithmetic by using a delta encoding [30] to compress the data and storing only the difference between them, but finally, the data are decompressed and processed using full bits. Nevertheless, in the algorithms based on Dynamic Programming (DP) as NWA and SWA, "the maximum difference between the compare data is fixed, it is known". To the best of our knowledge, there are not reported hardware designs entirely based on modular arithmetic for PSA, possibly because the A<B comparison is not defined in modular arithmetic space. However, A<B modular comparison can be employed without ambiguity when the maximum difference D between the two compared data (D=|A-B|) is fixed [31], which is one of the properties of these DP's algorithms. Considering that arithmetic mod-2^N is the ideal case for hardware-based designs (FPGA, ASIC), A<B comparison can be performed using two's complement subtractors of N-bits when |A-B|<2^(N-1) and indicated by the MSB, that is, A<B = MSB(A-B)[32].

To understand this, consider two large values "a" and "b" with a priori maximum difference D between them on a continuous infinite space, for instance, Reals or Integers. Next, consider two values "A" and "B" that correspond to their mapped values (A← a, B← b) in a continuous finite space mod-2^N. The signed difference between the original values and between the modular values is the same (Equation 8) when N is higher to the number of bits necessary to represent any difference between them (#D), which is shown in the Equation 9.

(8
)
(9

)

In NWA and SWA, the distance D between the H inputs values {H_{DIAG}, H_{UP}, H_{LEFT}, H_{EDGE}} only depend on the selected scoring scheme, and it requires few bits because D is small. The models for linear gap and affine gap are described in Equations 10 and 11.

(10
)

(11
)

Figure 31 shows an example for N=5 (D<16). Large integer values a=3234 and b=3224. Then, their mapped values to modular space mod-32 (N=5, then D<16) are A=2 and B=24, that is, A and B correspond to the N-LSB of a and b. Then, A<B=False considering that the MSB of the two's complement subtraction is '0' (A-B="01010"). Therefore, two's complement adders can be used on hardware-oriented designs to implement modular comparators.

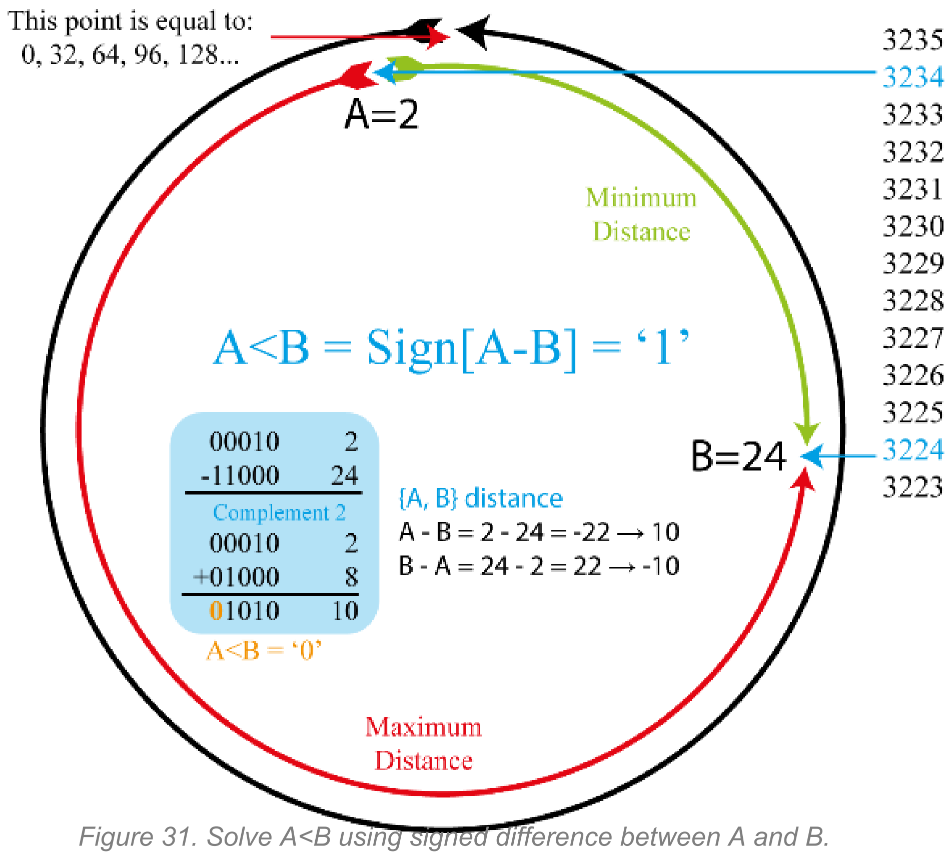


Figure 31. Solve A<B using signed difference between A and B.

Chapter 4
Proposed Approach for performing PSA using the NWA, K-Band method and arithmetic mod-2^N

- The proposed design for KBand based global PSA is a modification of the design presented in [14], which is oriented to perform the similarity using the SWA. For global PSA, also it is necessary to perform the initialization functions (See Figure 41) and traceback (See Figure 42). Summarizing, our design has the following features and conditions:
- Use arithmetic mod 2^N for performing the NWA.
- Process DNA sequences of any length due to arithmetic mod 2^N.
- The length difference between DNA sequences must be less than K, restriction of the KBand method.
- High PE density for a specific scoring scheme due to arithmetic mod 2^N.
- Use a simple accumulative method for performing the NWA initialization functions.
- Allow obtaining several optimal PSAs since that additionally codify the two arrows cases H_{DIAG}-H_{UP} and H_{DIAG}-H_{LEFT}.
- Center of K-Band is shiftable by adding zeros to a sequence.
- Detect the alignment zone dynamically.
- Processing by packets of the forward and traceback process.

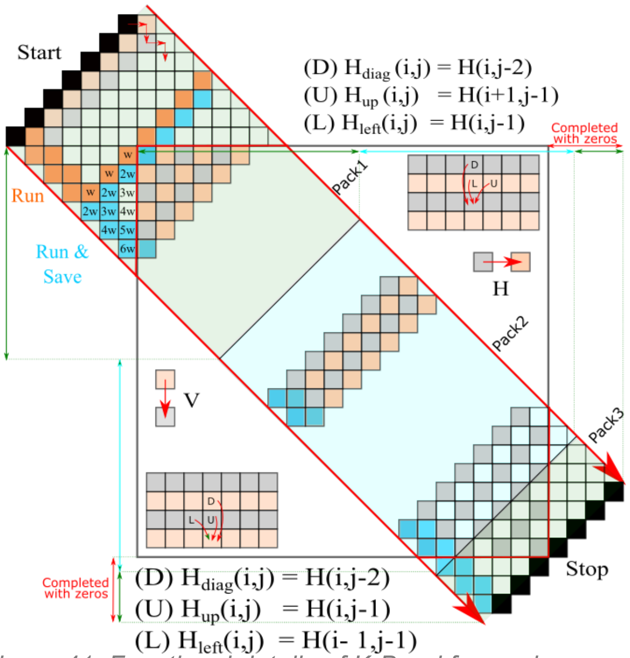

Start
(D) H_diag (i,j) = H(i,j-2)
(U) H_up (i,j) = H(i+1,j-1)
(L) H_left(i,j) = H(i,j-1)
Completed with zeros
Run
Run & Save
Pack1
Pack2
Pack3
H
V
D
L
U
Stop
Completed with zeros
(D) H_diag(i,j) = H(i,j-2)
(U) H_up(i,j) = H(i,j-1)
(L) H_left(i,j) = H(i- 1,j-1)

Figure 41. Functional details of K-Band forward process.

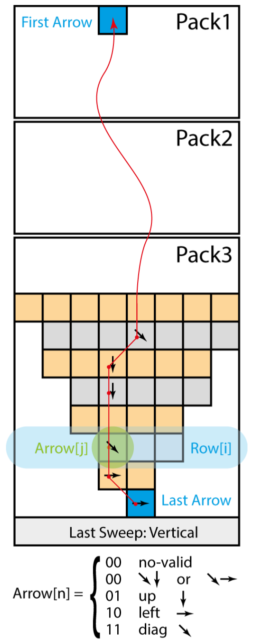


Figure 42. Traceback process.

4.1. Forward process

4.1.1. Diagonal wave-front: H/V sweep

The data dependency of the NWA and SWA allows the parallel processing of all cells of the same antidiagonal. Commonly, a horizontal or vertical sweep is used to process the similarity matrix. However, a diagonal wave-front sweep effect can be reached by alternating the direction sweep between horizontal and vertical sweep (H/V sweep). Considering a systolic array architecture, the Table 41 shows that horizontal sweeps are obtained by left-load of A[i] symbols whereas vertical sweeps by right-load of B[i] symbols.

Table 41. Start of alignment between A=ACTGTCAAT... and B=CTGATCACT...

Flag	#Cycle	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6	PE_7
START	0	0	0	0	0	0	0	0
RUN	1	A	0	0	0	0	0	0
	2	A	0	0	0	0	0	C
	3	C	A	0	0	0	0	C
	4	C	A	0	0	0	C	T
	5	T	C	A	0	0	C	T
	6	T	C	A	0	C	T	G
	7	G	T	C	A	C	T	G
R&S	8	G	T	C	AC	T	G	A
	9	T	G	T	CC	AT	G	A
	10	T	G	TC	CT	AG	A	T

4.1.2. Valid Alignment Cycles

Considering a parallel array processor, an alignment cycle is valid if at least one cell of the parallel processor presents a valid alignment, that is, both A[i] and B[i] sequences symbols correspond to valid symbols: {A, T, G, C, U, _,} for nucleotide sequences.

4.1.3.Band position: Shift with zeros

By default, the first cell of the similarity matrix H is aligned with the center of the K-Band, as shown the Table 1. However, the K-Band can be shifted to the left or the right by adding Zb zeros (number of shifts positions) at the beginning of the sequence A (Left-Shift) or the sequence B (Right-Shift). This feature gives flexibility for using the streaming processor, mainly when the sequences to align does not have the same length.

4.1.4.Alignment zone: Start and Finish

Assuming that the systolic array is empty at the initial state, the beginning of the alignment zone of the similarity matrix is determined by the first valid alignment cycle, which occurs later of K+Zb cycles; the example presented in Table 41 for the case K=7 and Zb=0 shows that the alignment zone starts in cycle 8. The finishing of the alignment zone is determined by the first cycle after the last valid alignment cycle, as it is shown in Table 42. The number of rows in the alignment zone is Length[A] + Length[B] – 1. The parallel processor registers that correspond to H matrix only are updated in the alignment zone. G matrix registers are always updated.

Table 42. End of alignment between A=…ACTGTCAAT and B=…CTGATCACT

Flag	#Cycle	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6	PE_7
	-5	A	TT	AC	AA	CC	TT	G
	-4	A	T	TC	AA	AC	CT	T
	-3	T	C	TA	AC	AT	C	T
	-2	T	C	A	TC	AT	A	C
	-1	C	A	C	TT	A	A	C
STOP	0	C	A	C	T	T	A	A

4.1.5.Data management: Packing

Sequences are processed by packets for reducing the cache memory requirements for allocating the I/O data of the parallel processor: segments of the A and B input sequences and segments of the arrow matrix. Parallel processor always waits for the next sequence segment, manipulating them as a single continuous sequence until detecting the alignment zone's finish.

4.1.6.Arrows encoding and storage

Arrows only are generated in the alignment zone by all parallel processor cells, both valid and no-valid cells. An arrow indicates the source cell associated with the maximum selected value of each H cell input, being possible cases with multiples arrows. For these cases, selecting a single arrow restring the solution to a single alignment, a single path, reducing the quality and flexibility of the traceback process. Usually, one arrow can be coded using 2-bits, being necessary more bits for two or three arrows. For this case, no-valid arrows are coded by “00” and valid one-arrows Diag, Up and Left by the codes “11”, “01” and “10”, respectively, and the two arrow cases Diag-Up and Diag-Left by also using the code “00”, as shown in Figure 42. Arrows storage can be divided into multiple files for reducing the memory requirements, since these files are interpreted as a single continuous file.

4.1.7.Data dependency: H sweep vs V sweep

For alternated H/V sweep, the data dependency change in each cycle, depending on the last direction sweep. The data dependency determines the data flow of the parallel processor, interconnections for systolic array case. Table 43 presents the data dependency for each processor cell, where the H inputs {H_{DIAG}, H_{UP}, H_{LEFT}} are connected from the H output of the one cell taking to account two parameters: the first parameter corresponds to the cell identifier and the second parameter corresponds to the number of delays applied to the H output.

Table 43. Data dependency for H matrix. Same data dependency applies for G matrix.

H input	H sweep	V sweep
H_{DIAG} (I , J)	**H (I , J-2)**	**H (I , J-2)**
H_{UP} (I , J)	**H (I+1 , J-1)**	**H (I , J-1)**
H_{LEFT} (I , J)	**H (I , J-1)**	**H (I-1 , J-1)**

4.1.8.Initialization function: Hedge

The initialization function generates the boundaries values necessary for processing the similarity matrix. For SWA these values always are zero. However, for NWA these values are defined in Equations 5 and 6. These equations are performed in the alignment zone by only all cells with no-valid alignments, allowing to obtain the H_{EDGE} values from any-cell outside of the alignment zone, where the most uncomplicated and general case is from the same cell.

4.1.9.Systolic array edge connections for using modular arithmetic

In K-Band method is necessary to solve the data dependency for the first and last cell of the parallel processor, since these edge connections are not defined for the cells 0 and K+1, presenting missing connections. According to the literature, these missing connections should be connected to -INF to prevent that these cases will be chosen, limiting the processing space to the band. However, the same result is obtained by connecting the missing inputs from a just lower value as it is indicates in Equation 12, generating dynamic negative-infinite values very close to the other score inputs values {H_{DIAG}, H_{UP}, H_{LEFT}, H_{EDGE}}, minimizing the difference between them and optimizing the use of modular arithmetic.

(12
)

4.2. Traceback process

4.2.1. Last direction sweep

Traceback process is performed based on the arrows generated by the forward process. The last direction sweep is necessary for a successful arrow decoding, considering that the data dependency is different for both H sweep and V sweep. The last direction sweep is codified in the last row of the last arrow packet by "0101…" for H sweep and "1010…" for V sweep.

4.2.2. Start alignment: Last arrow and H/V sweep

The traceback process starts in the last arrow. The next step is seeking the absolute position of this first arrow, which is located in the second last row of the last arrow packet and it is the single arrow of that row.

4.2.3. Decoding arrows: dual arrow

Each arrow indicates the relative position of the next arrow. The code "00" is used to codify the no-valid, Diag-Up and Diag-Left arrow cases. No-valid cases are not present in the optimal path because the K-Band limits the processing space to the band dynamically. The two-arrow cases are firstly pre-decoded in a one-arrow (tagArrow) taking to account the last alignment symbols, assigning diagonal-code when A[i]=B[j] or assigning the last tagArrow-code for the other cases.

In affine gap, other particular case is presented: the algorithm can ignore the first symbols of a matched segment. For delete this issue, diagonal-code is assigned to the pre-decoded arrow when the last tagArrow-code is equal to diagonal (match block) and A[i]=B[j].

4.2.4. Data dependency: H sweep vs V sweep

In each step, the absolute position of the next processed arrow is sought considering the current arrow and its associated direction sweep. Firstly, the relative position is obtained from the last arrow, determinate by the relative row position and the relative arrow position in the row. Then, the absolute position is compensated by adding it. For the arrow cases Up and Left, the arrow is sought one row next, and the direction sweep is changed. For Diagonal-arrow, the arrow is sought two rows next and the direction sweep is held. The data dependency was presented in Table 3.3.

4.2.5.Synchronization of multiple packages

When an arrow packet is processed completely, the absolute position of the next arrow is outside of the packet, that is, the arrow is in the next arrow packet. Therefore, firstly the current arrow's packet is closed, then the next arrow packet is opened, then the absolute row-position is compensated considering the outside-absolute row-position, taking into account that the arrows matrix is segmented in continuous packets.

4.2.6.Finish alignment: posA = posB = 0

Traceback process starts with the last symbols of both input sequences. The indexes i and j, associated with the sequences A and B, are initialized with the length of these sequences and indicate the sequences symbols to align. For Up and Left arrows, only one-index decreases, while for diagonal arrows both-indexes decrease. A[i] or B[j] symbols are added when the index decreases, while a gap is added when does not it. If all symbols of the one sequence have been used (i=0 or j=0), the other sequence is completed by decreasing its index in each step. The alignment finalizes when both indexes are equal to 0.

Chapter 5
Streaming processor designs: Linear gap and Affine gap

We develop two processors for performing the KBand-based global PSA, for both linear gap and affine gap. Each designed processor is based on a systolic array architecture, composed by: Two shift registers, for loading the sequences to align; K Processing Elements (PEs), for performing the NWA simultaneously; K Score modules, for solving the score matrix for each PE; An alignment zone detector, and synchronization and interconnection logic. Systolic array design is enhanced by using arithmetic mod 2^N. Their main features and conditions are:

- Process DNA sequences of any length.
- Use a simple accumulative method for performing the NWA initialization functions.
- High PE density for a specific scoring scheme.
- Allow obtaining several optimal PSAs since that additionally codify the two arrows cases H_{DIAG}-H_{UP} and H_{DIAG}-H_{LEFT}
- Center of K-Band is shiftable by adding zeros to a sequence.
- Detect the alignment zone dynamically.
- Processing by packets of the forward and traceback process.

The streaming processor also is composed by other components as it is shown in Figure 51. Two-FIFO are used to synchronize the input sequences loading, one-FIFO to synchronize the sending of the output arrows and a control unit for managing and synchronizing all components of the streaming processor.

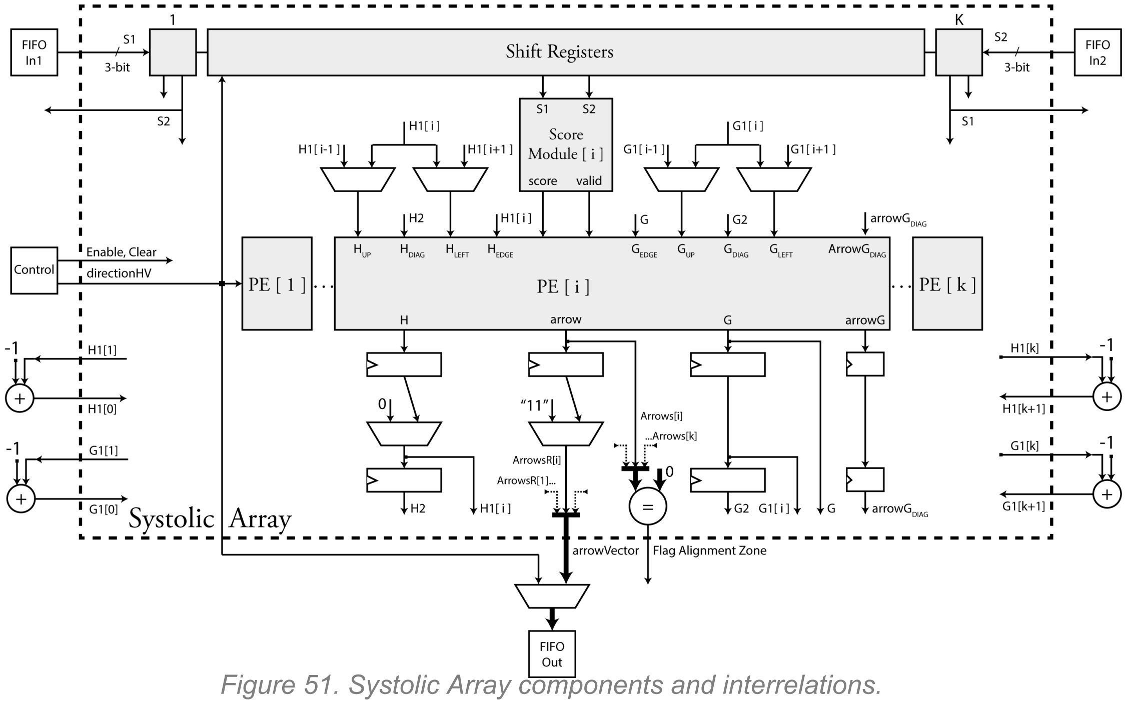
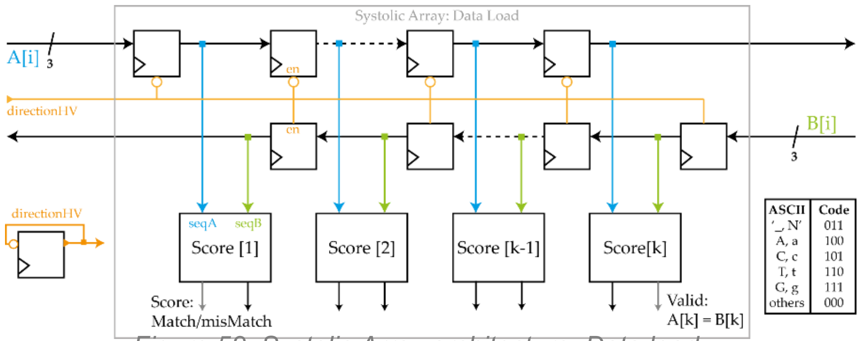


Figure 51. Systolic Array components and interrelations.

5.1. Systolic Array Design

5.1.1. Data load: score and valid

The 8-bits symbols of the input sequences A and B are codified in 3-bits, as shown in Figure 52. Then, they are loaded in the systolic array through two shift registers of 3-bits, one for each sequence. The sequences symbols are charged alternately in the shift registers but with an opposite data flow. A[i] symbols are loaded by the left when the sweep type is H, while B[i] symbols are loaded by the right when the sweep type is V, as shown in Figure 52. The score output value of the score scheme module is equal to “match” when the seqA and seqB inputs are equals, and it is equal to “mismatch” when they are different. The valid output indicates if the aligned symbols are both valid symbols, that is, both seqA and seqB are different to the code “000”.



ASCII	Code
‘_, N’	011
A, a	100
C, c	101
T, t	110
G, g	111
others	000

Figure 52. Systolic Array architecture. Data load.

5.1.2.Data dependency: H matrix, G matrix and edge connections

Data dependency changes according to the current sweep type: H or V sweep. Muxes are used in H_{UP} and H_{LEFT} PE inputs to control the data dependency, as shown in Figure 53. For affine gap, the same structure is used to the gap PE inputs G_{UP} and G_{LEFT}, but a mux is added between the H registers, which is explained in the Section 5.1.4. The H_{EDGE} input is connected from the H or G outputs from the same PE: H output with 1-delay for the linear gap and G output without delays for the affine gap. Adders are connected in the edges of the systolic array for preventing that these cases will be selected, according to explained in the Section 4.1.9.

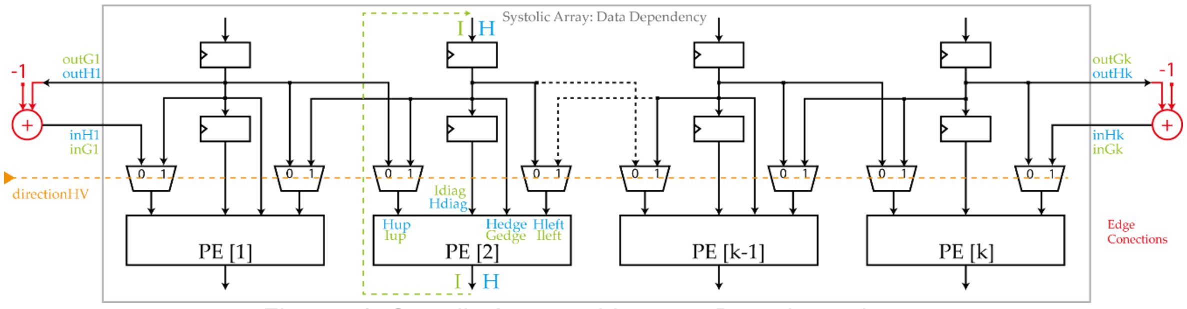


Figure 53. Systolic Array architecture. Data dependency.

5.1.3.Alignment zone

A flag indicates when the output Valid is equal to zero in all score cells, that is, all cells have A[i] or B[i] invalid symbols, which occurs when the parallel processor is outside of the alignment zone. Therefore, this flag allows detecting the beginning and the end of the alignment zone dynamically.

5.1.4.Especial cases: First H matrix value and Last arrow

When the first H-register is disabled outside of the alignment zone, this just failed for the first values of the initialization function of the H-matrix, from the point of view of the H_{DIAG} PE input, since the second H-register is updated with zero, failing for this case. To solve this issue, the first H-register is enabled outside of the alignment zone, but a mux is added between the H registers for ignoring its output, allowing to update the second H-register correctly. Similarly, in the traceback process, the last arrow can be sought because the last row has just a single arrow. However, both two-arrow and no-valid cases are codified by “00”. This issue is solved by adding a mux after the arrow register for giving priority to the diagonal arrow in the last row when the two-arrow case occurs.

5.1.5.Score scheme by FPGA reconfiguration

Considering that the produced alignment depends on the used score scheme, the score values (score matrix and gap penalties) are selected considering the sequences properties. These score values can be updated in two ways: software reprogramming and hardware reconfiguration. When software reprogramming is used, the score values are updated on registers, because the PE architecture must be general for supporting all input cases. Hardware reconfiguration is used to reduce the FPGA logic requirements since the FPGA design is synthesized for each score scheme parametrization (fixed values) which allow synthesizing processing units optimized for each one of the parametrization cases. Then, a binary programming file is generated for each case and the score values are updated by FPGA reconfiguration.

5.2. PE design for the linear gap

The core of the streaming processor for the linear gap is a systolic array with its respective PE, which is shown in Figure 54. The PE processing is based on arithmetic mod-2^N, for which we design modular comparators using C2 subtractor. The PE uses a single adder to perform the gap penalty based on a modified scoring scheme and a mux for performing the initialization function.

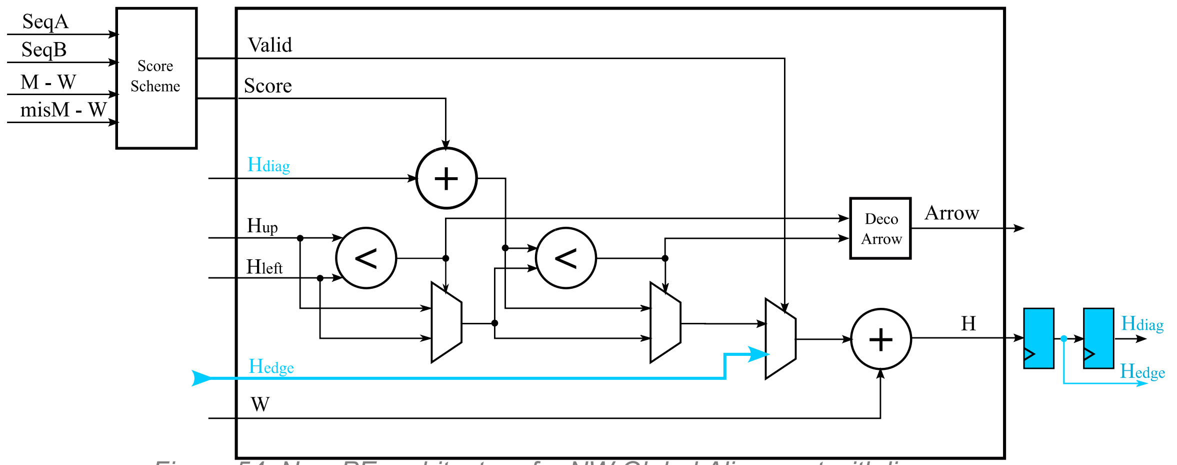


Figure 54. New PE architecture for NW Global Alignment with linear gap.

5.2.1.Modular comparators: C2 Subtractors

The design of modular comparators has been an obstacle for performing the sequence alignment using modular arithmetic. Considering what is presented in Chapter 3, modular comparators are designed using two's complement subtractors.

5.2.2.Modified scoring scheme: apply gap penalty just one time

In NWA, the gap penalty is applied to H_{UP}, H_{LEFT} and H_{EDGE} inputs, being necessary three adders. Two adders can be used by applying the gap before of the comparison Max(H_{UP}, H_{LEFT}). We use a new method for reducing the adders to a one-adder by applying the gap penalty just one time. For reach it, the gap penalty is added to H_{DIAG} case and compensated in the scoring scheme, specifically in the match and mismatch values, generalizing the gap penalty for all cases.

5.2.3.Initialization function

The mux-3 allows applying the initialization function to all PEs outside of the similarity matrix. Considering what is presented in section 4.1.8, H_{EDGE} values are obtained from the H output of the same PE with 1-delay. Table 51 shows the values presented in the H_{EDGE} input, which are equal to zero for all processed rows before the alignment zone because the systolic array registers are not updated in this zone. Table 52 shows that the initialization function is successfully performed in the H output, compared with Figure 41.

Table 51. H_{EDGE} values, or H save values for alignment presented in the Table 1.

#Cycle	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6	PE_7
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0
8	w	w	w	AC	w	w	w
9	2w	2w	2w	CC	AT	2w	2w
10	3w	3w	TC	CT	AG	3w	3w

Table 52. Initialization Function: (Hedge – w) or H values for alignment of Table 1

#CYCLE	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6	PE_7
6	w	w	w	w	w	w	w
7	w	w	w	**W**	**W**	w	w
8	2w	2w	**2W**	AC	**2W**	2w	2w
9	3w	3w	**3W**	CC	AT	**3W**	3w
10	4w	4w	TC	CT	AG	4w	4w

5.3. PE design for the affine gap

The core of the streaming processor for the affine gap is a systolic array with its respective PEs, which is shown in Figure 55. The PE processing is based on arithmetic mod-2^N, for which we design modular comparators using C2 subtractors. The G matrix and its respective arrow are calculated for they are used two cycles next for processing the H matrix. A mux is used to perform the initialization function.

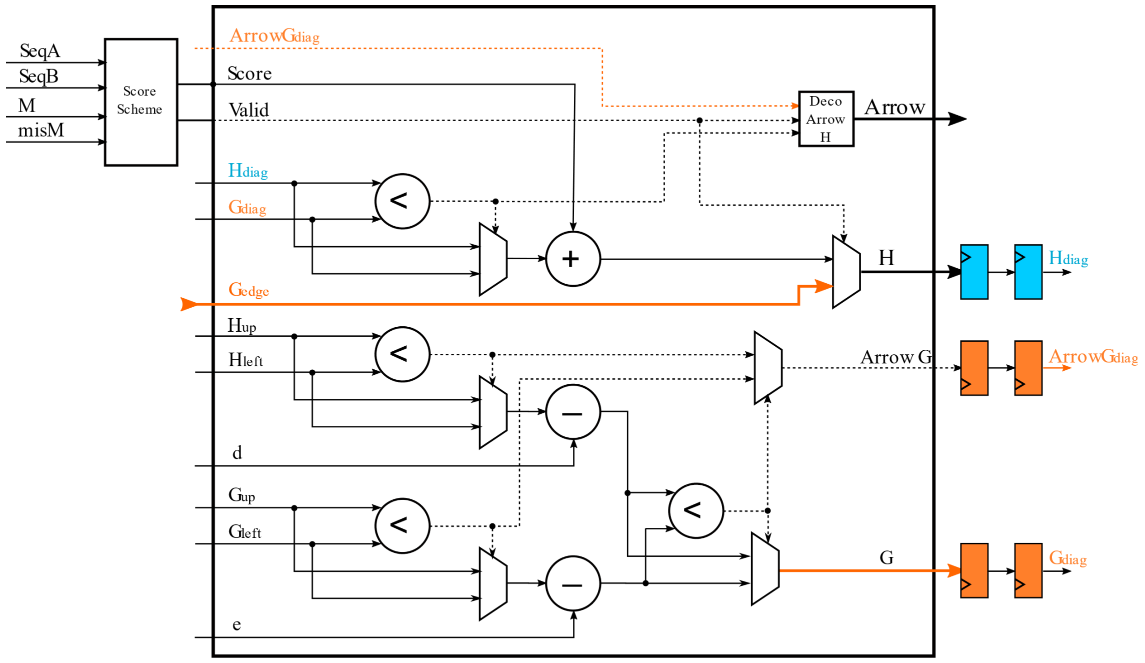


Figure 55. New PE architecture for a simplification of affine gap.

5.3.1. Modular comparators: C2 Subtractors

The design of modular comparators has been an obstacle for performing the sequence alignment using modular arithmetic. Considering what is presented in Chapter 3, modular comparators are designed using two's complement subtractors.

5.3.2. Initialization function

The mux-6 allows applying the initialization function to all PEs outside of the similarity matrix. Considering what is presented in section 4.1.8, H_{EDGE} values are obtained from the G output of the same PE, without delays. Table 53 shows the values presented in the H_{EDGE} input (G output), validating that the initialization function is successfully performed, which are equal to open gap for all processed rows before the alignment zone, because H_{UP} and H_{LEFT} are zero in this zone (First H register is not zero) and G_{UP} and G_{LEFT} are equal to d+e in this zone.

Table 53. Initialization Function: (Hedge – w) or H values for alignment of Table 1

#CYCLE	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6	PE_7
6	d	d	d	d	d	d	d
7	d	d	d	**D**	**D**	d	d
8	d+e	d+e	**D+E**	AC	**D+E**	d+e	d+e
9	d+2e	d+2e	**D+2E**	CC	AT	**D+2E**	d+2e
10	d+3e	**D+3E**	TC	CT	AG	**D+3E**	d+3e

5.3.3. Delayed diagonal arrow

When H matrix is performed and G_{DIAG} input is chosen, is necessary to know the source associated with G_{DIAG}: Left or Up. This is reached by using an arrow associated to G matrix which is codified by 1-bit.

5.4. FIFOs: Bottleneck

The streaming processor uses two 8-bits streaming inputs and one 128-bits streaming output. However, it generates 2-K arrows by clock cycle. For instance, 2048-bits are generated using 1024-PEs and are necessary 16-clocks to read the arrows through a 128-bits channel. Then, this is the bottleneck of the streaming processor. For reducing this issue, the streaming processor was designed to work using two clock domains: a low-frequency clock to operate the systolic array and a high-frequency clock to operate at the system level, that is, external components as processors, memory among others. For reach it, a FIFO memory is used to synchronize both clock domains.

5.5. Control Unit

The control unit synchronizes all other components of the streaming processor: systolic array and input/output streaming interfaces. The core is an FSM of 7-states, as shown in Figure 5.6.

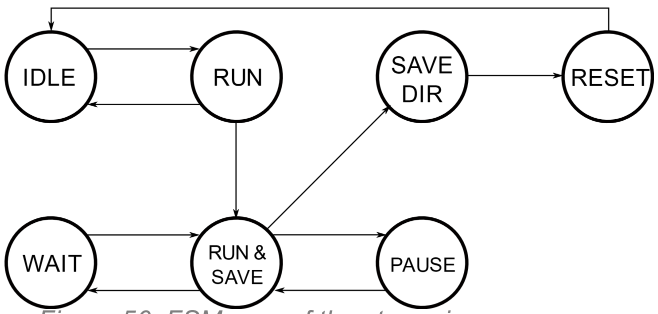


Figure 56. FSM core of the streaming processor.

FSM start in the IDLE state, which is a state of “wait for packets”. FSM changes to the RUN state when a new packet is read. The RUN state is an initialization state, where the sequences segments are progressively charged in the shift registers of the systolic array until the first valid alignment is detected, then change to the RUN&SAVE state. If a packet is processed completely, the FSM returns to the IDLE state, waiting for a next packet. In the RUN&SAVE state is carried out the alignment itself, so is necessary to store the generate arrows for the systolic array. If a packet is finished, the FSM changes to the WAIT state, waiting for a next packet. The WAIT state is equal to the IDLE state, but it is used when the alignment is already started, and the systolic array is generating valid arrows. When the output buffer is full, it is necessary to pause the systolic processing for avoiding the loss of data, changing the FSM to the PAUSE state. When the forward process finishes, the FSM changes to the SAVEDIR state for storing the direction of the last sweep, then it continues whit the RESET state for restarting the IP for a next alignment.

5.6. Results of Synthesis

The streaming processor was synthesized using two FPGA-based platforms: DE10-nano and Cyclone 10 GX. DE10-nano is based on a low-cost Intel Cyclone-V SoC-FPGA (5CSEBA6U23I7NDK), with 42K-ALMs operating to maximum 300 MHz, and an ARM dual-core hard processor operating to maximum 800MHz. Cyclone 10 GX is based on an Intel FPGA Cyclone-10 (10CX220YF780E5G), which has more logic resources (80K-ALMs) and performance (max 600MHz).

Design was synthesized for configurations with 2^N PEs, using two score-system setup modes: software reprogramming (inputs) and hardware reconfiguration (parameters), only for the specific case: match = 1, mismatch = -1, gap-linear = -2, open-gap = -2 and extension-gap = -1. The reprogrammable mode allows modifying the scoring scheme via software, that is, this mode uses a single FPGA binary programming file for all scoring schemes. In the reconfigurable mode, the scoring scheme is modified by FPGA reconfiguration, that is, an FPGA binary programming file is necessary for each scoring scheme configuration. These synthesis results only correspond to the streaming processors; the results can change when these components are integrated on an embedded system, because of the changes presented in the fitter stage (Place & Route).

5.6.1. Synthesis results for a DE10-nano

DE10-nano is a low-cost platform based on an Intel Cyclone-V SoC-FPGA, device reference 5CSEBA6U23I7NDK. The FPGA component has 42K-ALMs operating to maximum 250 MHz, and the HPS component has an ARM dual-core hard processor operating to maximum 800MHz. Table 54 presents the synthesis results for reprogramming mode, reaching 1024 PEs of 3-bits, for both linear and affine gap. Table 55 presents the synthesis results for reconfiguration mode, reaching 2048 PEs of 3-bits for the linear gap, and 1024 PEs of 4-bits for the affine gap. Both tables show that the design for the affine gap needs more logic resources than design for the linear gap, while the frequency is similar. Comparing Table 54 and Table 55, it is observed that the PE logic requirements are reduced when the scoring scheme is fixed in compilation time, which increase the PE density.

Table 54. Streaming processor for reprogramming mode. Maximum number of H-bits (#bitsH) by number of PEs.

#PEs	#BitsH	ALMs Linear	Freq (Mhz)	#BitsH	ALMs Affine	Freq (Mhz)
1024	3	34133	87/236	3	41758	91/237
512	8	36293	77/252	7	40084	62/181
256	24	37653	62/228	16	37698	70/273

Table 55. Streaming processor for reconfiguration mode. Maximum number of H-bits (#bitsH) by number of PEs.

#PEs	**#BitsH**	**ALMs** Linear	**Freq** (Mhz)	**#PEs**	**ALMs** Affine	**Freq** (Mhz)
2048	3	40996	123/206	-	-	-
1024	5	34829	83/241	4	40077	88/229
512	12	35897	76/275	8	34528	78/241
256	28	37099	68/275	16	32385	74/256

5.6.2.Synthesis results for a Cyclone 10 GX

Cyclone 10 GX development kit is a low-cost and high-performance platform based on an Intel Cyclone-10 FPGA, device reference 10CX220YF780E5G. This FPGA has 80K-ALMs operating to maximum 600MHz, similar features to Stratix and Arria FPGAs. Table 56 presents the synthesis results for reprogramming mode, reaching 2048 PEs of 3-bits for the linear gap, and 2048 PEs of 3-bit for the affine gap.

Table 57 presents the synthesis results for reconfiguration mode reaching 4096 PEs of 3-bits for the linear gap, and 2048 PEs of 4-bits for the affine gap. Comparing Table 56 and Table 57, it is observed that the PE logic requirements are reduced when the scoring scheme is fixed in compilation time, which increase the PE density. Comparing Table 54 and Table 55 with Table 56 and Table 57, the streaming processor reaches a higher PE density and PE frequency using high-end FPGA and SoC-FPGA based platforms.

Table 56. Streaming processor for reprogramming mode. Maximum number of PEs for a Cyclone 10 GX FPGA development kit.

#PEs	**#BitsH**	**ALMs** Linear	**Freq** (Mhz)	**#BitsH**	**ALMs** Affine	**Freq** (Mhz)
2048	6	68801	72/336	3	62492	92/326
1024	16	77539	63/441	10	76778	82/446
512	32	74370	62/525	20	73694	81/599
256	64	72897	60/599	40	72221	72/599

Table 57. Streaming processor for reconfiguration mode. Maximum number of PEs for a Cyclone 10 GX FPGA development kit.

#PEs	**#BitsH**	**ALMs Linear**	**Freq (Mhz)**	**#BitsH**	**ALMs Affine**	**Freq (Mhz)**
4096	3	73139	71/307	-	-	-
2048	7	77555	68/304	4	78413	80/306
1024	16	77592	57/440	9	61068	80/452
512	32	74829	61/520	20	74037	76/524
256	64	71455	78/599	40	71455	78/599

5.6.3.Comparison with others works

Table 58 reports synthesis results of similar designs based on systolic array architecture. The previous K-Band designs reach until 128-PEs of 12-bits, while our streaming processor reaches 2048 PEs for the linear gap and 1024 PEs for the affine gap on a CycloneV SoC-FPGA, and it reaches 4096 PEs for the linear gap and 2048 PEs for the affine gap on a Cyclone10 FPGA. Previous works are limited by the sequence length, while the proposed design can process sequences of any length. Considering that the related works are mainly for amino-acids, the score module block of the streaming processors can be modified for processing similarity matrix of amino-acids as blosum62 or PAM250 using PEs with a number of bits between 6 and 8-bits.

Table 58. Comparison with other similar systolic array-based designs.

Design	Device	#PEs (K)	Frequency	#BitsH (word)	Length	K-Band
[14], [30]	Virtex-II 6000-6	75	66-MHz	11	2-K	SW Affine
[33]	EP2S180F1508C5	350	42-MHz	-	350	No
[34]	EP1S30	80	82-MHz	20	80-K	No
[35]	XC5VLX110	195	162-MHz	11	192	No
[15]	EP4SGX230KF40C2	128	125MHz	12	10-K	SW Affine
Our	5CSEBA6U23I7NDK	2048	123/206-MHz	3	Any	NW Linear
Our	5CSEBA6U23I7NDK	1024	88/229-MHz	4	Any	NW Affine
Our	10CX220YF780E5G	4096	71/307-MHz	3	Any	NW Linear
Our	10CX220YF780E5G	2048	80/306-MHz	4	Any	NW Affine

Chapter 6

Performing global PSA on a Low-Cost SoC-FPGA Platform

In order to verify the streaming processor (IP), we design an embedded system for Intel SoC-FPGA platforms, which integrates the streaming processor, as shown in Figure 61. The block diagram is shown in Figure 10. For this case, we use a low-cost SoC-FPGA platform DE10-nano. The streaming processor is used to accelerate the forward process of the K-Band global PSA based on the NWA, and it is managed by using a co-design application. Traceback process does not need to accelerate, so it is performing by using a full-software application. The SoC-FPGA embedded system was verified using pairs of similar DNA-sequences taken from NCBI as proteins, genomes and chromosomes, which are reported in Table 61. Results were compared with it obtained by software implementations developed in Wolfram Mathematica.

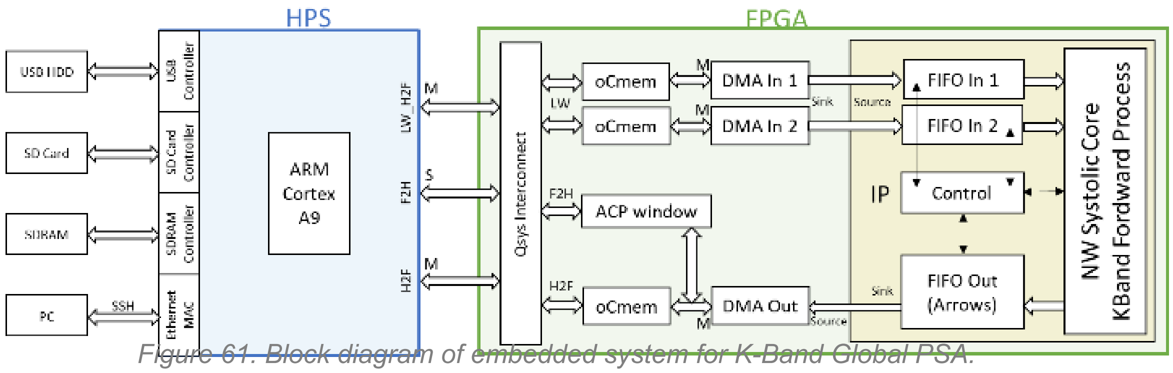


Figure 61. Block diagram of embedded system for K-Band Global PSA.

Table 61. Pair of sequences used to test the design. Uncoded symbols ("N") have been omitted in the alignments.

ID	ACCESSION NUMBER (NCBI)	LENGTH
1-A	NM_001354943.1	1,236
1-B	NM_001354944.1	1,062
2-A	NM_001354925.1	3,368
2-B	NM_001354924.1	3,431
3-A	NM_001011645.3	10,257
3-B	NM_000044.4	10,070
4-A	PSXO01000091.1	30,485
4-B	PSXO01000097.1	30,316
5-A	PSWL01000013.1	100,451
5-B	PSWJ01000010.1	100,053
6-A	NZ_PQKM01000006.1	300,261
6-B	NZ_PQKN01000004.1	300,339
7-A	NZ_KN849241.1	1,000,480
7-B	NZ_LJXA01000025.1	1,000,381
8-A	NZ_CP007019.1	3,000,273
8-B	NC_017545.1	3,000,464

9-A	CM000111.5	10,323,212
9-B	KZ626786.1	10,323,212
10-A	NW_004929415.1	30,985,389
10-B	KE150199.1	30,985,389

6.1. Embedded system design

The SoC-FPGA embedded system is composed of a Hard Processor System (HPS) as a fix component and an FPGA as a configurable component. HPS and FPGA are interconnected using HPS AXI interfaces. The core of the HPS is an ARM Cortex A9, a general-purpose processor which allows running software applications on an operating system, Linux OS for this case. The core of the FPGA component is the streaming processor (or IP). Two DMA controllers are used to send packets of sequences to the streaming processor for processing they, and a single DMA controller is used to read packets of arrows generated by the streaming processor. Dedicated FPGA on-chip memory is used as shared-memory between the HPS and each one DMA controller. An F2H channel allows memory access from the FPGA to HPS memory space using an ACP window.

In order to verify the streaming processor, it was configured with 1024-PEs of 4-bits. For this case, the design uses 29K-ALMs (70%) for the linear gap and 38K-ALMs (90%) for the affine gap, operating to a maximum internal/external frequency of 57MHz/104MHz for the linear gap and 56MHz/111MHz for the affine gap. Therefore, the embedded system was configurated to 50MHz internal frequency (Systolic Array) and 100MHz external frequency (System level). 320KB (47%) of cache on-chip memory was used, distributed by 32KB to each input sequence and 256KB to the arrows.

6.2. Co-design application for performing the forward process

A co-design application for performing the forward process was developed for testing the streaming processor using codesign techniques for managing it. Figure 62 illustrates their tasks and data flow. First, the inputs sequences are loaded from files in a FASTA single-line format. Next, zeros are added to the shorter sequence to make their lengths equal and for adjusting the center of the K-Band. Next, sequences are divided into packets. Next, each packet is sent to the streaming processor using two 8-bits dedicated DMA streaming interfaces. Next, the packets of arrows generated are read using a 128-bits DMA streaming interface. Next, packets of arrows are stored using files of maximum 100MB for being used in the traceback process.

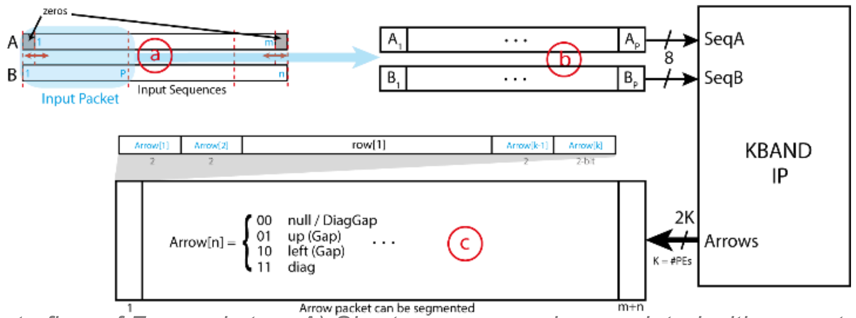


Figure 62. Data flow of Forward step. A) Shorter sequence is completed with zeros to make their lengths equal. Next, they are segmented in packets. B) Structure of the input packets C) Structure of the output packets.

6.2.1. The format of the input sequences

The co-design application only processes DNA-sequences in a text-based FASTA-single-line format. In this format, the file is composed by only two lines. The first line is used as a header and it starts with a ">" symbol; this line is used to describe the sequence. The second line is used to codify the sequence. For this case, only the nucleotide-symbols {A, T, C, G, U, _, N} are supported, that is, the sequence cannot have symbols as space, tab and any other unsupported ASCII symbol. Sequence length is the same length as the second line.

6.2.2. Band position: Shift with zeros

When two sequences have different lengths, zeros are added to the short sequence to make their lengths equal. These zeros can be added to the beginning (left) or/and at the end (right). When all zeros are added at the end, the center of the K-Band is aligned with the beginning of the sequences. Conversely, when all zeros are added at the beginning, the center of the K-Band is aligned with the end of the sequences. For this case, by default are added zeros evenly on both sides of the sequence, aligning the center of the K-Band with the a priori center of the alignment, as shown in Figure 63. The user can modify the K-Band center by modifying the zeros distribution.

A) K-Band aligned with the sequence-start	A	C	T	G	A	C	T	G	A	C
	A	C	T	A	A	C	T			
B) K-Band aligned with the sequence-end	A	C	T	G	A	C	T	G	A	C
				A	C	T	A	A	C	T
C) K-Band aligned with the a-priori center of the alignment	A	C	T	G	A	C	T	G	A	C
		A	C	T	A	A	C	T		

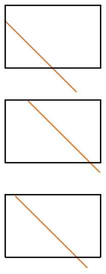


Figure 63. Band position depends of the added zeros distribution.

6.2.3. Processing by packets: DMA streaming interfaces

The streaming processor can perform the full PSA at once (The best performance) but using a lot of memory. For example, for processing pair-sequences of 1Mbp using1024 PEs are necessary 500MB for store the arrows. For reducing the memory requirements, sequences can be processed by packets. We use packets of K/2 symbols by sequence since it is the simplest packet that can be processed and only requires of 2MB of memory to save the arrows for 1024 PEs, considering that the minimum latency zone is equal to K (number of PEs). the first packet does not generate arrows (latency) and one packet of zeros is used at the end to complete the alignment (offset by latency). The arrows generated by each packet are stored on an SD-Card or an HDD connected to the DE10-nano through a USB 2.0 OTG port.

6.3. Software application for performing the traceback process

Traceback process is the second step of the PSA based on dynamic programming. In this step are processed the packets of arrows generated in the forward process, starting by the last arrow and following an optimal path indicated by the arrows, until the first arrow. The Algorithm 1 was designed for performing it taking into account that the arrows are sorted by packets and decoded considering the direction sweep associated to each row of arrows.

6.3.1.Initial values of the alignment symbol indexes

In the traceback process, the symbols are processed in reverse order. Therefore, the indexes I and J used for selecting the sequence symbols to align are initialized with the sequence lengths to start the alignment by the last symbol.

6.3.2.Number of rows of arrows by packet

When a packet of arrows is loaded, the first step is calculating the number of rows of arrows stored in it. Then, to reach it is necessary to know the number of PEs used to perform the forward process. Considering that one-arrow is coded using 2-bits, the number of rows of the packet is equal to 8*#BytesByPacket divided by 2*#PEs.

6.3.3.Seek the last direction sweep

The last direction sweep is coded in the last row of the last packet of arrows. When it is H sweep, the last row has the pattern “0101…” while for V sweep, the pattern is “1010…”. This last row also is useful for detecting synchronization issues in the forward process.

6.3.4.Seek the last arrow

The last arrow is in the second last row. For seeking it, this row is divided into segments of two-bits, and the single arrow is in the position with a code different to "00". This position corresponds to the current absolute position of the arrow in the row.

6.3.5.Get the next arrow and direction sweep

In each iteration of the traceback process, the arrow is sought using the absolute row position in the packet of arrows and the absolute arrow position in the row of arrows. This pointer defines the arrow position completely.

6.3.6.Decode arrows

Arrows are decoded taking into account four special cases. First, the index I is equal to zero when the sequence A has aligned completely, then the alignment is completed with the remaining symbols of the sequence B. Similarly, the index J is equal to zero when the sequence B is fully aligned, then the alignment is completed with the remaining symbols of the sequence A. Third, if the current arrow is not diagonal but the last processed arrow (arrowTag) if it is and the current symbols to align are equals (A[i]=B[j]), then the arrow is replaced by a diagonal arrow to prevent that the first symbols of a matched segment will be ignored. Fourth, the two-arrow cases are interpreted as diagonal arrows when A[i]=B[j], otherwise the arrow is processed using the last arrowTag value.

6.3.7.Perform alignment: Get and add align symbols

The symbols to align are specified by the index I and J. For Up and Left arrows, only one-index decreases, while for diagonal arrows both-indexes decrease. A[i] or B[j] symbols are added when the index decreases but gaps are added when does not it.

6.3.8.Perform similarity and distance

The similarity is calculated by increasing in one for each pair matching cases of aligned symbols and by decreasing for other cases: mismatch and gaps. The distance is calculated by only increasing for mismatch and gaps cases.

6.3.9.Update align indexes and arrow position

The I and J indexes decrease when the arrow is processed as Diagonal, but only I or J decreases when the arrow is processed as Up or Left, respectively. The relative arrow position is determined considering the data dependency for each direction sweep. Next, the absolute arrow position, that is, the arrow position in the row and the row position in the packet, is updated by adding the relative arrow position.

6.3.10. Change of the packet arrow file

The next arrow is in the next packet when the row position pointer is negative. For this case, firstly the current packet of arrows is closed, then the next arrow packet is opened. Next, the row pointer is initialized with the last row position and compensated by adding its previous negative value, emulating a single continuous packet of arrows. Finally, the arrow is located in the recently opened packet.

6.4. Results

In order to verify the streaming processor, it was integrated into a SoC-FPGA embedded system. Software simulations were developed on Wolfram Mathematica, running on a workstation Dell 7910 which it has two 2.2GHz Intel® Xeon® E5-2630 v4 processors of 10 cores each one and 64GB of RAM. Both embedded and PC platforms were tested using sequences-pairs distributed logarithmically in a range from 1-Kbp to 30-Kbp.

6.4.1.Performance Results

The performance of the co-design solution used for testing the streaming processor was compared with their respective software simulation. The bottleneck of the co-design solution is the arrow storage, using an HDD connected via USB 2.0. The software simulation only was performed until 1-Mbp because the arrows only are stored on RAM, which needs high memory resources. The results are reported in Table 62. In average, forward is performing 58x faster to the linear gap and 427x to the affine gap using the co-design solution, and traceback is performing 6.9x faster.

Table 62. Performance comparison between software and SoC-FPGA implementations

	Software [s]			SoC FPGA [s]		Ratio Soft/SoC		
Seqs A-B	**Forward Linear**	**Forward Affine**	**Traceback**	**Forward Both**	**Traceback**	**Forward Linear**	**Forward Affine**	**Traceback**
1	6.718 e0	4.211 e1	0.078	1.07 e-1	1.11 e-2	62.78	393.5	7.03
2	1.587 e1	1.223 e2	0.110	2.84 e-1	2.85 e-2	55.88	430.6	3.86
3	4.446 e1	3.627 e2	0.312	7.86 e-1	8.16 e-2	56.56	461.4	3.82
4	1.337 e2	8.212 e2	1.797	2.311 e0	2.50 e-1	57.85	355.3	7.19
5	4.543 e2	3.056 e3	6.953	7.591 e0	8.33 e-1	59.85	402.6	8.35
6	1.351 e3	1.075 e4	20.265	2.290 e1	2.266 e0	59.00	469.4	8.94
7	4.219 e3	3.676 e4	71.324	7.693 e1	7.864 e0	54.84	477.8	9.07
8	-	-	-	2.236 e2	2.566 e1	-	-	-
9	-	-	-	7.725 e2	8.723 e1	-	-	-
10	-	-	-	2.312 e3	2.625 e2	-	-	-

The performance of the co-design solution (K-Band PSA) also was compared with the performance of the built-in function SequenceAlignment of Wolfram Mathematica (Full PSA). The results are reported in Table 63. The processing-time increases exponentially for the built-in function and increase linearly for the co-design solution, reaching the best performance for short sequences greater than 3Kbp.

Table 63. Performance comparison between Wolfram Mathematica built-in function and SoC-FPGA design.

Seqs A-B	**Built-in function[s] SequenceAlignment**	**SoC [s] Forward + Traceback**	**Ratio**
1	6.25 e-2	1.181 e-1	0.529
2	3.91 e-1	3.125 e-1	1.251
3	3.40 e0	8.676 e-1	3.919
4	3.738 e1	2.561 e0	14.596
5	4.419 e2	8.424 e0	52.457
6	3.113 e3	2.517 e1	123.70
7	3.838 e4	8.479 e1	452.63
8	-	2.493 e2	-
9	-	8.597 e2	-
10	-	2.574 e3	-

6.4.2.Quality Results

The alignments generated by the streaming processor was compared with its software simulation, obtaining the same alignments. Additionally, the results were compared with the in-built function SequenceAlignment of Wolfram Mathematica for evaluating the K-Band method. The results of similarity and distance are reported in Table 64. Alignments within the K-Band are identical from a practical point of view, even for large sequences as the cases 6, 9 and 10, reported in Table 61. For other cases, the alignments are different; however, the similarity and distance are similar for both K-band and full PSA methods. Therefore, The K-Band alignment for large alignments can be advantageous since it only considers the alignment of very close segments.

Table 64. Similarities / distances comparison for both K-Band method and full global PSA.

Seqs A-B	**Length**	**Built-in function[s] SequenceAlignmen t[ ]**	**K-Band Linear**	**K-Band Affine**
1	1-Kbp	872/182	874 / 181	842 / 197
2	3-Kbp	3221/106	3228 / 102	3209 / 112
3	10-Kbp	9883/187	9883 / 187	9875 / 191
4	30-Kbp	4731/14268	3885 / 14299	821 / 15923
5	100-Kbp	17830/45786	15532 / 45751	4947 / 51196
6	300-Kbp	299358/627	299364 / 624	299314 / 649
7	1-Mbp	562350/257206	-48897 / 509079	-43647 / 561344
8	3-Mbp	-	93361 / 1554377	-171281 / 1707310
9	10-Mbp	-	10322112 / 0	10322112 / 0
10	30-Mbp	-	30904614 / 0	30904614 / 0

Chapter 7

Conclusions and Future Works

7.1. Conclusions

According to Section 2.4, the algorithms for Multiple Sequence Alignment (MSA) have two critical tasks: Pairwise Sequence Alignment (PSA) and profile-profile alignment. Both tasks require hardware acceleration, while the other MSA tasks can be solved efficiently by software. This work was addressed for optimizing PSA in order to speed up the MSA process. MSA tasks were not integrated because the profile-profile alignment was not accelerated, restricting the MSA to small sequences.

The main contribution of this work is the hardware-oriented algorithms for performing global PSA based on arithmetic mod 2^N. This approach could be extrapolated to other DP-based algorithms used in other applications.

Using the above algorithms, we design two processors for performing the KBand-based global PSA, for both linear gap (Needleman-Wunsch Algorithm) and affine gap (Gotoh Algorithm). Each processor based on modular arithmetic 2^N was designed using a new systolic array core, which was implemented with new PE designs, maximizing the PE density for a selected scoring scheme. The designed processors allow performing PSA for sequences of any length (Verified up 30Mbp), reaching 2048-PEs of 3-bits for linear gap and 1024-PEs of 4-bits for affine gap using a low-cost SoC-FPGA, and 4096-PEs of 3-bits for linear gap and 2048-PEs of 4-bits for affine gap using a high-end FPGA. The processors were verified on a SoC-FPGA embedded system platform, and they were used to accelerate the forward process. Traceback was processed in software.

The PSA results from DE10-nano SoC-FPGA platform ($130, 10W/h) were compared with the results obtained from the workstation DELL Precision 7910 ($4000, 1300W/h), reaching an average speed-up of 58x for the linear gap and 427x for the affine gap.

7.2. Future Works

Future works are oriented to improve the PSA and MSA methods using arithmetic mod 2^N, considering the next activities:

- Design two streaming processor for performing the local PSA (Smith-Waterman) using the K-Band method, for both linear gap and affine gap.
- Design a method for increasing the band coverture, ideally an adaptative method.
- Design a streaming processor for performing progressive profile-profile alignments.
- Evaluate other applications of the arithmetic mod 2^N to bioinformatic algorithms.
- Simplify the traceback process to perform only the similarity and the distance, increasing the performance for this purpose.

References

[1] "MS Bioinformatics." RESEARCH CENTER FOR MODELING & SIMULATION (RCMS), 2018.

[2] J. Xiong, *Essential Bioinformatics*. New York: Cambridge University Press, 2006.

[3] D. Huson, *Algorithms in Bioinformatics*, vol. 2452. 2002.

[4] M. S. Waterman, "Identification of Common Molecular Subsequences," *J. Mol. Biol.*, pp. 195–197, 1981.

[5] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similiarities in the amino acid sequence of two proteins," *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970.

[6] J. Silvestre-Ryan, "Global and Local Sequence Alignment Algorithms." Wolfram Demonstrations Project, 2011.

[7] O. Gotoh, "An improved algorith for matching biological sequences," *J. Mol. Biol.*, vol. 162, pp. 705–708, 1982.

[8] P. Compeau, P. Pevzner, G. Teslar, "How to apply de Bruijn graphs to genome assembly," *Nat. Biotechnol.*, pp. 987–991, 2011.

[9] "BLAT (bioinformatics)." Wikipedia, 2018.

[10] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Commun. ACM*, vol. 18, no. 6, pp. 341–343, 1975.

[11] E. W. Myers and W. Miller, "Optimal alignments in linear space," *Bioinformatics*, vol. 4, no. 1, pp. 11–17, 1988.

[12] R. J. Lipton and D. Lopresti, "Comparing long strings on a short systolic array," *Princent. Univ.*, 1986.

[13] K. M. Chao, W. R. Pearson, and W. Miller, "Aligning two sequences within a specified diagonal band," *Bioinformatics*, vol. 8, no. 5, pp. 481–487, 1992.

[14] A. B. B. Harris, “Acceleration of Gapped Alignment in BLASTP Using the Mercury System,” vol. 47, 2006.

[15] Y. Liao, Y. Li, N. Chen, and Y. Lu, “Adaptively Banded Smith-Waterman Algorithm for Long Reads and Its Hardware Accelerator,” *2018 IEEE 29th Int. Conf. Appl. Syst. Archit. Process.*, pp. 1–9, 2018.

[16] H. Suzuki and M. Kasahara, “Acceleration of Nucleotide Semi-Global Alignment with Adaptive Banded Dynamic Programming,” *bioRxiv*, vol. 25, no. 18, pp. 1–1, 2017.

[17] Greg Caporaso, *An Introduction To Applied Bioinformatics*. Caporaso Laboratory - Northern Arizona University, 2015.

[18] T. Ohlson, B. Wallner, and A. Elofsson, “Profile-profile methods provide improved fold-recognition: A study of different profile-profile alignment methods,” *Proteins Struct. Funct. Genet.*, vol. 57, no. 1, pp. 188–197, 2004.

[19] B.-J. Yoon, “Hidden Markov Models and their Applications in Biological Sequence Analysis,” *Curr. Genomics*, vol. 10, no. 6, pp. 402–415, 2009.

[20] J. Söding, “Protein homology detection by HMM-HMM comparison,” *Bioinformatics*, vol. 21, no. 7, pp. 951–960, 2005.

[21] J. Daugelaite, A. O ’driscoll, and R. D. Sleator, “An Overview of Multiple Sequence Alignments and Cloud Computing in Bioinformatics,” *ISRN Biomath.*, vol. 14, 2013.

[22] J. D. Thompson, F. Plewniak, and O. Poch, “A comprehensive comparison of multiple sequence alignment programs,” *Nucleic Acids Res.*, vol. 27, no. 13, pp. 2682–2690, 1999.

[23] X. D. Wang, J. X. Li, Y. Xu, and J. Zhang, “A survey of multiple sequence alignment techniques,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015.

[24] J. D. Thompson, D. G. Higgins, and T. J. Gibson, “CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice,” *Nucleic Acids Res.*, 1994.

[25] F. Sievers *et al.*, “Fast, scalable generation of high-quality protein multiple

sequence alignments using Clustal Omega.," *Mol. Syst. Biol.*, vol. 7, no. 1, p. 539, 2011.

[26] C. Notredame, D. G. Higgins, and J. Heringa, "T-coffee: A novel method for fast and accurate multiple sequence alignment," *J. Mol. Biol.*, vol. 302, no. 1, pp. 205–217, 2000.

[27] K. Katoh, K. Misawa, K. Kuma, and T. Miyata, "MAFFT a novel method for rapid multiple sequence aligment based on fast Fourier transform," *Nucleic Acids Res.*, vol. 30, no. 14, pp. 3059–3066, 2002.

[28] E. Becker, A. Cotillard, V. Meyer, H. Madaoui, and R. Guérois, "HMM-Kalign: A tool for generating sub-optimal HMM alignments," *Bioinformatics*, vol. 23, no. 22, pp. 3095–3097, 2007.

[29] R. C. Edgar, "MUSCLE: Multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Res.*, vol. 32, no. 5, pp. 1792–1797, 2004.

[30] A. Jacob, J. Lancaster, J. Buhler, B. Harris, and R. D. Chamberlain, "Mercury BLASTP: Accelerating Protein Sequence Alignment," *Science (80-.).*, vol. 1, no. 2, pp. 1–62, 2009.

[31] G. Davida and B. Litow, "Fast Parallel Arithmetic Via Modular Representation," *Soc. Ind. Appl. Math.*, vol. 20, no. 4, pp. 756–765, 1991.

[32] D. Wei, N. Ansari, and J. Chen, "A Compressed and Dynamic-range-based Expression of Timestamp and Period for Timestamp-based Schedulers *," *IEEE Glob. Telecommun. Conf. (Cat. No.01CH37270), San Antonio, TX*, vol. 4, pp. 2353–2357, 2001.

[33] P. Faes *et al.*, "Scalable Hardware Accelerator for Comparing {DNA} and Protein Sequences," *InfoScale '06 Proc. 1st Int. Conf. Scalable Inf. Syst.*, no. January, pp. 33–38, 2006.

[34] X. Jiang, X. Liu, L. Xu, P. Zhang, and N. Sun, "A reconfigurable accelerator for Smith-Waterman algorithm," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 54, no. 12, pp. 1077–1081, 2007.

[35] M. N. Isa, K. Benkrid, T. Clayton, C. Ling, and A. T. Erdogan, "An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis," *Proc. 2011 NASA/ESA Conf. Adapt. Hardw. Syst. AHS 2011*, pp. 344–351, 2011.