PLOS ONE

RESEARCH ARTICLE

Proposal of Smith-Waterman algorithm on FPGA to accelerate the forward and backtracking steps

Fabio F. de Oliveira[1,4☯], **Leonardo A. Dias**[2☯], **Marcelo A. C. Fernandes**[1,3,4☯*]

1 Laboratory of Machine Learning and Intelligent Instrumentation, nPITI/IMD, Federal University of Rio Grande do Norte, Natal, Brazil, **2** Centre for Cyber Security and Privacy, School of Computer Science, University of Birmingham, Birmingham, United Kingdom, **3** Department of Computer and Automation Engineering, Federal University of Rio Grande do Norte, Natal, Brazil, **4** Bioinformatics Multidisciplinary Environment (BioME), Federal University of Rio Grande do Norte, Natal 59078-970, RN, Brazil

☯ These authors contributed equally to this work.
* mfernandes@dca.ufrn.br

Citation: Oliveira FFd, Dias LA, Fernandes MAC (2022) Proposal of Smith-Waterman algorithm on FPGA to accelerate the forward and backtracking steps. PLoS ONE 17(6): e0254736. https://doi.org/10.1371/journal.pone.0254736

Editor: Slavisa Jovanovic, Institut Jean Lamour, FRANCE

Received: June 27, 2021

Accepted: June 11, 2022

Published: June 30, 2022

Data Availability Statement: All relevant data are within the manuscript.

Funding: This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Finance Code 001. The funders had no role in the study design, data collection, and analysis, decision to publish, or preparation of the manuscript.

Competing interests: The authors have declared that no competing interests exist.

Abstract

In bioinformatics, alignment is an essential technique for finding similarities between biological sequences. Usually, the alignment is performed with the Smith-Waterman (SW) algorithm, a well-known sequence alignment technique of high-level precision based on dynamic programming. However, given the massive data volume in biological databases and their continuous exponential increase, high-speed data processing is necessary. Therefore, this work proposes a parallel hardware design for the SW algorithm with a systolic array structure to accelerate the forward and backtracking steps. For this purpose, the architecture calculates and stores the paths in the forward stage for pre-organizing the alignment, which reduces the complexity of the backtracking stage. The backtracking starts from the maximum score position in the matrix and generates the optimal SW sequence alignment path. The architecture was validated on Field-Programmable Gate Array (FPGA), and synthesis analyses have shown that the proposed design reaches up to 79.5 Giga Cell Updates per Second (GCPUS).

1 Introduction

In Bioinformatic, the analysis can be divided into three parts called primary, secondary, and tertiary analysis [1, 2]. The primary analysis is responsible for generating genomic data information from biological material. In the primary analysis, the sequencing machines create raw genomic data (or raw data). The raw data is composed of several genome reads. The secondary analysis involves reads alignment and trimming based on quality, and at the end of this step, a whole genomic is created. Finally, tertiary analysis can be characterized as interpreting results and extracting meaningful information from the data. In this last step, many algorithms and techniques can be applied. Also, many applications are created from these analyses. The tertiary analysis covers various applications, from genome characterization to a vaccine or drug treatment creation [2].

A large amount of raw data has been generated in recent years due to the replacement of Sanger sequencing by Next-Generation Sequencing (NGS), also called High-Throughput Sequencing (HTS) [3, 4], where NGS data analysis includes the three analysis categories mentioned above. Each sequencing machine can be created about 7 Tera base pairs (bp) per hour (Tbp/h) [5]. Intense sequencing of raw data from organisms and sharing data around the world are critical to monitoring major SARS-CoV-2 viral mutations and viral control [6]. Disease caused by the SARS-CoV-2 virus has been spreading worldwide and has been declared a pandemic by the World Health Organization [7, 8].

After sequencing reads, alignment methods can be performed to map and determine the evolutionary line of the targeted organism, such as its phylogeny. As a result, it is possible to understand the sample's action mechanics by comparing them with cataloged samples in existing databases [2, 9]. The most used meta-heuristic alignment method for the sequences is the Basic Local Alignment Search Tool (BLAST) due to its fast processing speed and less memory usage than deterministic alignment algorithms [10].

However, different from the meta-heuristics, deterministic alignment methods offer the optimal alignment for a given input sequence instead of an approximate solution. The main deterministic methods are the Needleman-Wunsch (NW) and Smith-Waterman (SW) algorithms for global and local alignment, respectively [11, 12]. Nonetheless, a significant disadvantage of these algorithms is their slow processing speed and high memory usage due to the computational complexity. For example, SARS-CoV-2, commonly vary from 28k to 31k base pairs (bp) in size. Thus, performing thousands of large-size sequence alignments became a real challenge for extracting information on the raw data.

Thus, it is essential that the processing of algorithms associated with the bioinformatics area cover three critical requirements: high processing speed (high-throughput), ultra-low-latency, and low-power [13–15]. Bioinformatics analysis algorithms are critically dependent on the computational infrastructure to cover high-throughput, ultra-low-latency, and low-power requirements. It can be said that there are three categories of infrastructure used, which are: Central Process Units (CPUs) [16], Graphics Processing Units (GPUs) [17, 18], and Custom Hardware Architectures (CHA) [19–21].

Genomic analysis solutions associated with the High-Performance Computing (HPC), which is first (CPUs) and second (GPUs) categories of computational infrastructure, use systems based only on software that can be implemented using only CPUs and GPUs. According to results presented in [22–24], these software-only approaches struggle to keep up with the growing computational demands of genomic analysis, given the barriers to reducing large-volume latency and power consumption using only CPUs and GPUs. In addition, as the number of nodes grows to handle increasing amounts of data, performance is not scaled linearly [16, 25–27]. The third (CHA) category of infrastructure has been presenting itself as an exciting alternative to satisfy high-throughput, ultra-low-latency, and low-power requirements [28–33].

Therefore, this work presents a parallel FPGA design with a systolic array structure to accelerate both the forward and backtracking stages of the SW algorithm. The main contributions are high-speed data processing implementation and low memory usage that allows for high scalability. According to [34], the systolic array is a class of parallel computing architecture that describes an array for dense linear algebraic calculations, proposed by [28]. Its hardware implementation usually uses a pipeline structure, where the data is propagated between Processing Elements (PEs). Besides, its main advantage is to reduce the number of memory accesses throughout the data flow. Hence, systolic arrays simplify the architecture and improve the system's operating frequency [35].

To overcome the low-speed processing bottleneck and maintain the optimal alignment of deterministic algorithms, parallel hardware implementations for the SW algorithm have been proposed in the literature. The main platforms used are Field Programmable Gate Arrays (FPGAs), Central Processing Units (CPUs), and GPUs. FPGAs are widely known for their flexibility for parallelization and low-power consumption. An FPGA is a matrix of logic blocks that allows designing different circuits, such as processors, logic circuits, and even algorithm development [31]. FPGA platforms can be categorized as third generation computational infrastructure in bioinformatics, as it is a CHA. Also, the logical blocks within the FPGA are independent, allowing operations to be carried out in parallel and only one clock cycle, unlike CPUs that operate sequentially based on instructions and GPUs that require constant access to memories.

1.1 Related works

This subsection briefly discusses hardware-based approaches for the SW algorithm available in the literature. Usually, the SW is used for protein and DNA sequence alignment, but their parallel strategies are quite different. Many hardware implementations targeting protein alignment were developed on supercomputers, CPUs, GPUs, and FPGAs, as can be seen in [36–40]. Due to an extensive number of previous works in the field, we consider the DNA alignment implementations, which in turn are commonly based on Resistive Content Addressable Memories (ReCAMs) [22], Application-Specific Integrated Circuits (ASICs) [41], and FPGAs [23, 24, 39, 42–53].

GPUs are well-known for their high degree of parallelism and computing intensity. However, high-performance GPUs have a high cost, and, broadly speaking, GPUs have significant computing latency, and low energy efficiency compared to custom hardware, but with more energy efficiency than CPUs, as seen in the results [22, 23, 40]. The high computing latency is due to the high number of cores and low cache memory to control these cores. In contrast to GPUs, FPGAs are customizable according to the user's needs, achieving better computing performance and lower latency [15, 54, 55]. However, FPGA hardware development is usually complex and takes a long time.

Unlike the conventional platforms previously mentioned, the SW algorithm has also been implemented on ResCAMs, as can be seen in [22]. ResCAMs is a storage accelerator system that allows millions of processing units (PUs) to be deployed over multiple silicon arrays. In [22], the implementation was used to compare the homologous chromosomes between humans (GRCh37) and chimpanzees (panTro4), and the only SW step performed was the building of the score matrix. As a result, their proposal achieved 5, 300GCUPS, a 4.8× speedup over the GPU performance. Besides, it also had a 1.7× better energy efficiency compared to an FPGA implementation.

In [45], a hardware/software co-design was implemented on FPGA to reduce the execution time of short-sequence alignment during genome sequence analysis. The analysis was performed using the Shouji method, a highly parallel and accurate pre-alignment filter that reduces the need for computationally-costly dynamic programming algorithms. The FPGA was used to boost the algorithm's performance. As a result, integrating the Shouji and aligner methods reduced the alignment total execution time by up to 18.8×.

In [48], a systolic-array-based architecture is presented as a DNA sequencer, using the SW algorithm affine gap penalty score. According to [48], the SW algorithm was implemented on a Xilinx Virtex-6 FPGA, reaching 465 Giga Cell Update per Second (GCUPS), and reducing the area occupation by 90% compared to other architectures.

In [50], an FPGA runtime accelerator is presented to align data information sequences. The authors' implementation is based on the seed-and-extend model of Bowtie2, achieving a similar alignment rate while mapping reads by ≈2× faster. Meanwhile, it is proposed by [46] an FPGA implementation of the SW algorithm to replace the GPU used in [40], called SWIFOLD. The FPGA implementation is based on OpenGL and utilized for long DNA sequences alignment. The SWIFOLD approach for accelerating the SW alignment reached an average of 125 GCUPS.

In [41], an ASIC design for traceback recording with penalty scoring is proposed. The design is implemented on a TSMC 40nm technology, and the aligner strategy can speed up pairwise alignment by 71× compared to the CPU.

Meantime, in [47] is presented an FPGA approach to meet the alignment operation processing requirements. The paper introduces a register-file concept to reduce run-time storage, and it does not require any sorting or comparison operations to prepare for the final sequence alignment. As a result, a 128 GCUPS performance was achieved using 256 PEs.

In [56], an FPGA hardware implementation of SW and NW algorithms is presented. The performance and area occupation is evaluated for different hardware designs. Besides, a convolution neural network model is introduced to implement the NW algorithm, achieving 98.3% accuracy.

In [23], a heterogeneous FPGA architecture for sequence alignment is proposed. Unlike most of the works in the literature, their implementation aims to accelerate the entire SW algorithm with the backtracking process. For this purpose, the architecture can process long strings of data based on parallelism and partitioning strategies; and the backtracking process was performed by dividing the equal parts of the similarity matrix, while the search started from the lower right sub-matrix. The tests were performed for 512 Processing Elements (PEs), reaching 76.8GCUPS at 150MHz and 105.9GCUPS (with external memory) for 200MHz. As a result, a speedup of 3.6× to 25.2× was achieved regarding other SW designs implemented on FPGA and GPUs. Besides, it reached a 26% reduction in power consumption compared to the GPU implementation.

Similarly, more FPGA approaches using systolic arrays for the NW and SW with backtracking sequencing techniques have been proposed, such as [39, 57]. In [39], a VHDL SW implementation, using Dynamic Programming (DP) with approximation correspondences for two different strategies, was proposed. It achieved 23.5GCUPS with speedups between 150× to 400× compared to a 2004-era PC. Meanwhile, in [57], the implementation was based on PEs to perform elementary calculations and a diagonally backtracking search, also developed in VHDL. Comparisons were made with the linear and affine strategies, achieving 10.5GCUPS.

In [44], the SW forward and backtracking processes were implemented in an FPGA. The Qnet structure was adopted for communicating with the FPGA, reaching 25.6GCUPS, a speedup of 300× compared to a desktop computer.

Therefore, it can be noted from the literature that the key points for a high-performance SW implementation on FPGA are the operating frequency and number of PEs, which in turn are associated with the hardware capacity and design critical path. Thus, we present an FPGA implementation for the SW algorithm using systolic arrays, as in [23, 39, 44, 57].

Our approach performs both the forward and backtracking stages of the algorithm. Unlike the approaches in the literature, we obtain the alignment path distances during the forward stage processing and the maximum score, reducing the complexity of the backtracking stage processing. Memories are used to propagate the distances and maximum score, allowing the backtracking step to follow the path directly. Thus, our architecture achieves good performance (short critical path), reduced memory usage and, theoretically, high scalability, and prevents memory access overlap latency, even implementing the two stages of the SW algorithm.

It is essential to mention that our proposed hardware architecture can perform the alignment of any sequence length. However, the resources available in the target FPGA are a limiting factor for the size of the score matrix, as shown in [23, 47, 58]. Nonetheless, we provide a proof-of-concept and an actual implementation on the Virtex-6 FPGA using a synthetic dataset.

2 Smith-Waterman algorithm

Smith and Waterman originally proposed the SW algorithm in 1981 to performs local sequence alignment of nucleotides and proteins in the biological field [12]. The sequence alignment of the SW algorithm includes the forward and backtracking stages, which are performed by the calculation results of the alignment similarity score. Besides, the alignment is performed based on two input sequences called query sequence, **q**, and dataset sequence **s**. The query sequence can be expressed by

$$\mathbf{q} = [q_1, \cdots, q_j, \cdots, q_N] \tag{1}$$

where q_j is the j-th nucleotide or amino-acid protein and N is the length of the query sequence. The dataset sequence can be expressed by

$$\mathbf{s} = [s_1, \cdots, s_i, \cdots, s_M] \tag{2}$$

where s_i is the i-th nucleotide or amino-acid protein, and M is the dataset sequence length. Therefore, the SW algorithm is calculated iteratively for two dimensions, and it has a computational complexity of $O(M \times N)$.

The forward stage calculates the scoring matrix, **H**, where **H** is a two-dimensional array that can only take values greater than or equal to 0 (i.e., $\mathbf{H} \in \mathbb{N}^2$). This matrix is generated by comparing the elements of the sequences **q** and **s**. Usually, **H** is generated using DP, and it is initialized with zeros in the first row and column. Subsequently, the DP process is performed to calculate the sequence scores. Based in works presented in [23, 36, 59], the recurrence relationship can be defined as

$$\mathbf{H}_{M,N} = \begin{cases} \mathbf{H}(i,j) = \max\{0, \mathbf{E}(i,j), \mathbf{F}(i,j), \mathbf{H}(i-1,j-1) + \mathbf{P}(s_i, q_j)\} \\ \mathbf{E}(i,j) = \max\{\mathbf{H}(i,j-1) + \rho, \mathbf{E}(i,j-1) + \sigma\} \\ \mathbf{F}(i,j) = \max\{\mathbf{H}(i-1,j) + \rho, \mathbf{F}(i-1,j) + \sigma\} \end{cases} \tag{3}$$

where $\mathbf{H}(i,j)\{(i,j) \in \mathbb{N} \,|\, 1 \leq i \leq M,\ 1 \leq j \leq N\}$, **P** is the score matrix used for obtaining the similarity score between s_i and q_j, **E** and **F** are two assisted matrices when calculating matrix **H**, ρ is the gap opening penalty and σ is gap extension penalty. In the particular case of $\rho = \sigma$, a linear gap penalty model is obtained, opening and extending a gap with the cost γ. **P** is also called a substitution matrix, where the simplest version is when the diagonal receives the match value and the rest of the matrix has a mismatch value. When performing all element calculations, this expression is the $\mathbf{H}_{M,N}$ matrix. Therefore, $\mathbf{H}(i, j)$ is the maximum alignment score of two sub-sequences **s** and **q**. The initialization condition is

$$\mathbf{H}(i,0) = \mathbf{H}(0,j) = \mathbf{E}(i,0) = \mathbf{F}(0,j) = 0 \forall\ \{(i,j) \in \mathbb{N} \,|\, 1 \leq i \leq M,\ 1 \leq j \leq N\}. \tag{4}$$

The maximum score value of $\mathbf{H}(i, j)$ in the forward stage is the last sequence that will be aligned. To determine the relationship, the previous neighborhood values of the analyzed element are required, i.e., the values on the diagonal, horizontal, and vertical positions, as illustrated in Fig 1. As can be observed, the score of w can be found based on its neighborhood (x, y, v), which is $\mathbf{H}(i-1, j-1)$, $\mathbf{H}(i-1, j)$, $\mathbf{H}(i, j-1)$, respectively. This windowing step occurs throughout the process of determining all scores in **H**.

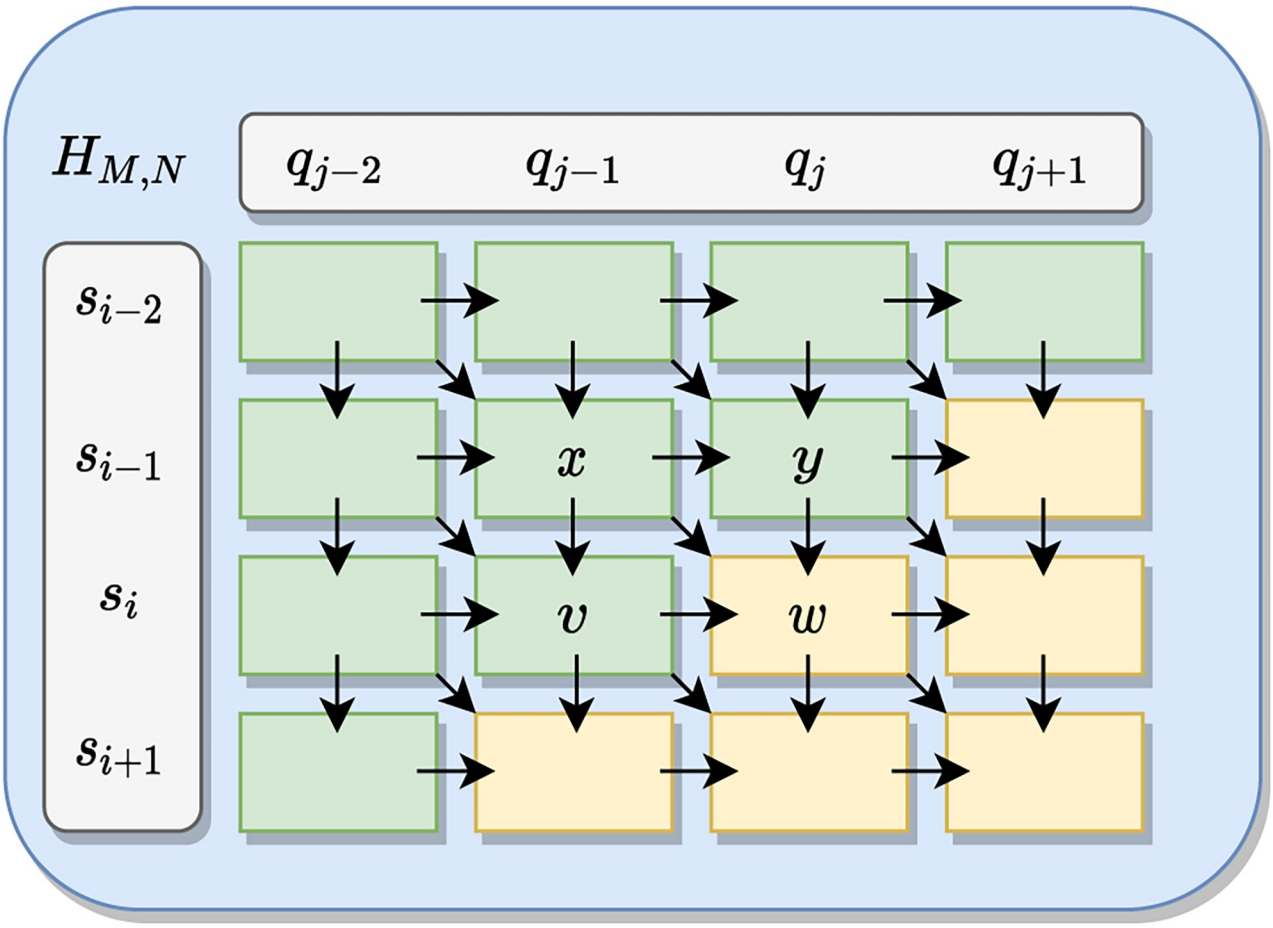

Fig 1. The direction of the score computation in the matrix during the SW forward stage. To determine a score, such as *w*, the neighborhood values (*x*, *y*, and *v*) have to be known. The green-colored cells indicate already computed values, while the yellow cells indicate that the values to be calculated.

https://doi.org/10.1371/journal.pone.0254736.g001

As shown in Fig 1, the neighborhood values *x*, *y* and *v*, must necessarily be known to determine the value of *w* (i.e., **H**(*i*, *j*)). For this purpose, those values are defined based on the sequences **q** and **s**. Thereby, the *w* score is determined as

$$w = \max \begin{cases} x + \alpha & \text{if } q_i = s_j \\ x + \beta & \text{if } q_i \neq s_j \\ y - \gamma & \\ v - \gamma & \\ 0 & \end{cases}, \tag{5}$$

where γ, α, and β represent the linear gap, a match, and a mismatch, respectively. A gap is a penalty that causes an empty element in the sequence (represented by a dash symbol), while the other sequence continues. It can result from the query or database sequence. The Eq 5 is equivalent the Eq 3, where $x+ (\alpha \vee \beta) = \mathbf{H}(i-1, j-1) + \mathbf{P}(s_i, q_j)$, $y+ \gamma = \mathbf{F}(i, j)$ and $v+ \gamma = \mathbf{E}(i, j)$. Finally, when fully populated, the **H** matrix contains the score and path information.

The backtracking stage starts after determining all the scores in the **H** matrix, i.e., calculating the score of all cells **H**(*M*, *N*). Hence, the backtracking begins at the cell with the highest

value in the **H** matrix (maximum score) and trace-back the next position based on the highest neighborhood value, according to Eq 5, which can be on the diagonal, horizontal, or vertical direction. This is an iterative process that repeats until it reaches the limit value, usually set to a score of 0. Also, a directional flag indicates the path. Finally, the backtracking path determines the best local alignment. The diagonal direction points to a match in the alignment, while the horizontal and vertical directions indicate gaps which are represented by dashes in the **s** and **q** sequences, respectively.

3 Implementation description

The hardware architecture for the SW algorithm proposed in this work was developed using systolic arrays to input two DNA sequences and increase the processing speed of the local sequence alignment. An overview of the systolic array structure of the proposal for *N* PEs is shown in Fig 2. Besides, each PE is divided into 3 modules. These modules are the forward stage, the storage process, and the backtracking stage, as seen in Section 2. Each module is illustrated in blue, green, and yellow, respectively. The forward stage has its module named as Matrix Score Module (MSM), the storage process module is called as Memory Module (MM), and the backtracking stage has its module as backtracking stage (BS).

The labeled signals shown in Fig 2 are generated outside the modules. Meanwhile, the non-labeled ones are generated by computations inside the modules and detailed throughout this Section. The sequences **q** and **s**, defined according to Eqs 1 and 2, are external discrete signals used as inputs of the SW algorithm. Furthermore, each signal in the sequences represents one of the four DNA nucleotides, i.e., *A, G, T, or C* (also accepting twenty distinct levels referring to amino acids or another set of sequences). The design proposed supports any sequence set as

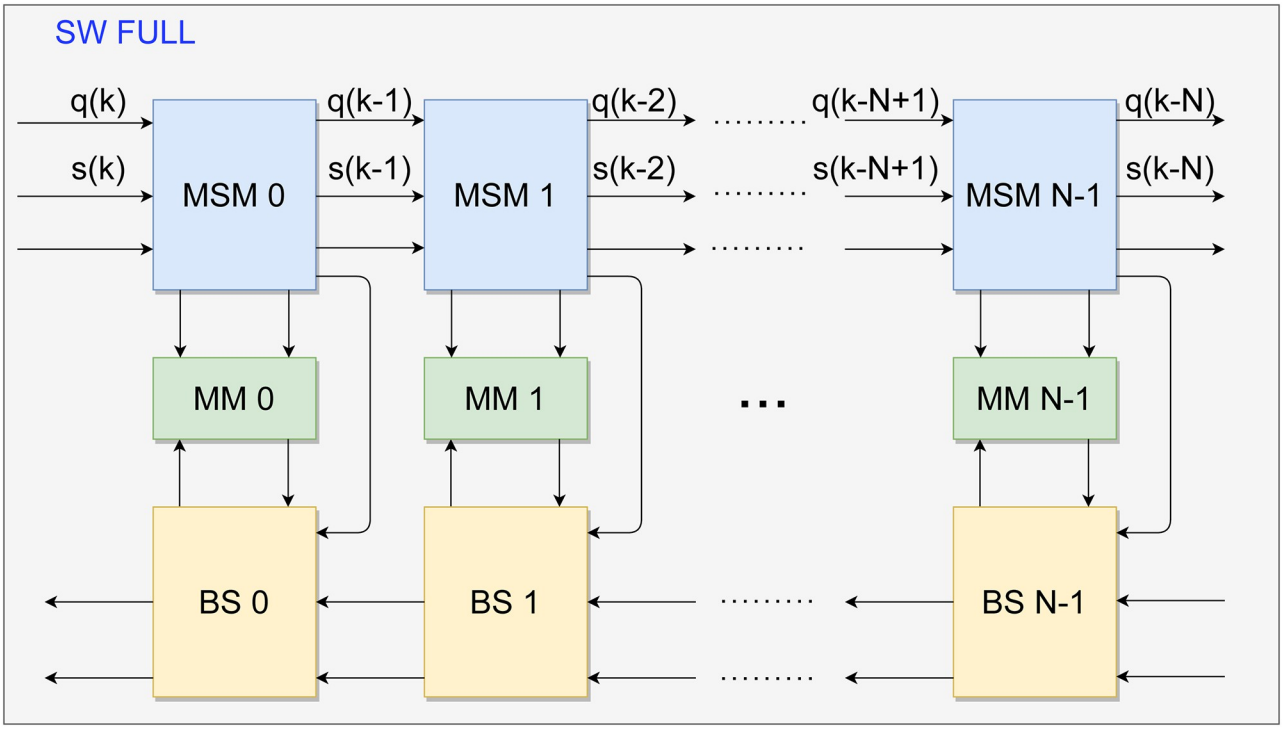

Fig 2. General architecture for the SW algorithm. The forward stage (MSM) is represented by the blue block, the backtracking stage (BS) by the yellow block, and the Memory (MM) by the green block. Only external signals are displayed, i.e., the **q** and **s** signals.

https://doi.org/10.1371/journal.pone.0254736.g002

in any SW algorithm, but for an efficient alignment, it is necessary to adopt a suitable scoring matrix that models each possible symbol's frequencies that can occur in the sequences.

Initially, the circuit starts when the MSM modules propagate the **q** and **s** signals. As seen in Fig 2, each *k*-th element of the **q** and **s** sequences are shifted to each MSM output to shorten and stabilize the critical path, as well as allowing the computation of scores synchronously, preserving the systolic array structure. Afterward, the MSM computes the score according to Eq 3, and propagates the sequence elements to the next MSM; also, the computed results are sent to the respective MM in their order of entry. During this process, the MM operates exclusively in writing mode while the process has not reached the last computation between the two sequences.

The forward stage is completed after fully computing the scores of the **H** matrix. Also, the last MSM enables the backtracking process. Consequently, the MM switches to the read mode, and the BS reads the data computed by its respective MSM. The alignment starts from the calculations performed in the MSM. Then, from the respective defined PE in the forward stage, the process starts and ends according to the definitions of the SW algorithm.

Fig 3 shows the block design that represents each PE of the systolic array, with a detailed description of the signals between the modules within one PE. As can be observed, besides the two input sequences to be compared, **q** and **s**, the MSM also receives an enable signal, **e n**. After computing the score between each *k*-th element of the two sequences (i.e., an element of the **H** matrix), the MSM outputs to the next PE the following signals: the calculated score, Sc_j; the maximum score, *MaxVal*, and its position, $AddrRAM_{i\wedge j}$; the PE *index*; along with the input signals *q*, *s*, and *en*, shifted in time. In addition, the MSM also outputs signals to the MM, which are the calculated path direction, *Direction*, and the storage address of that path *wAddrDir*.

Subsequently, after fully populating the **H** matrix and, consequently, the **D** matrix, the forward stage is finished enabling the *Traceback* signal, which in turn begins the BS. Firstly, the BS sets signal BT_{Start} to 1, indicating the start of the backtracking process. Therefore, the $mRAM_{i\wedge j}$ are propagated back until it reaches the BS with maximum score, which is identified by the signal *index*. From this location match, the *btcontrol* signal is changed to allow the reading of the memory by MM. Thus, the BS receives the path value from the MM at signal d_j when sending the memory address $rAddrDir_j$ signal. The d_j value allows the BS to calculate the next requested address and propagate it to the next module through the *path*(*j*) signal, representing the memory address of the request path in MM. Lastly, the alignment value enters *valDir*, and the process continues until it reaches the complete alignment. All modules are detailed in the following subsections. All signals present in this Section are shown in Table 1.

3.1 Forward approach

Firstly, based on the principles "divide and conquer" for solving computational problems, we propose a matrix used to store only the values of the recursive path, called the **D** matrix. The **D** matrix is not widely used in the SW literature. However, it is important to achieve a solution at lower-level programming. Besides, a matrix with two different types of information, such as the **H** matrix, increases the hardware design complexity. Matrix **D** needs to store only 4 levels of values which are: 0, 1, 2 and 3. Each element of the matrix **D** needs 2 bits to be expressed, delivering a more economical storage process compared to **H**, which can certainly need more than 2 bits to represent each element.

As previously mentioned, the alignment process is performed based on the query and dataset input sequences, **q** and **s**, respectively. Also, there can have different sizes, represented by

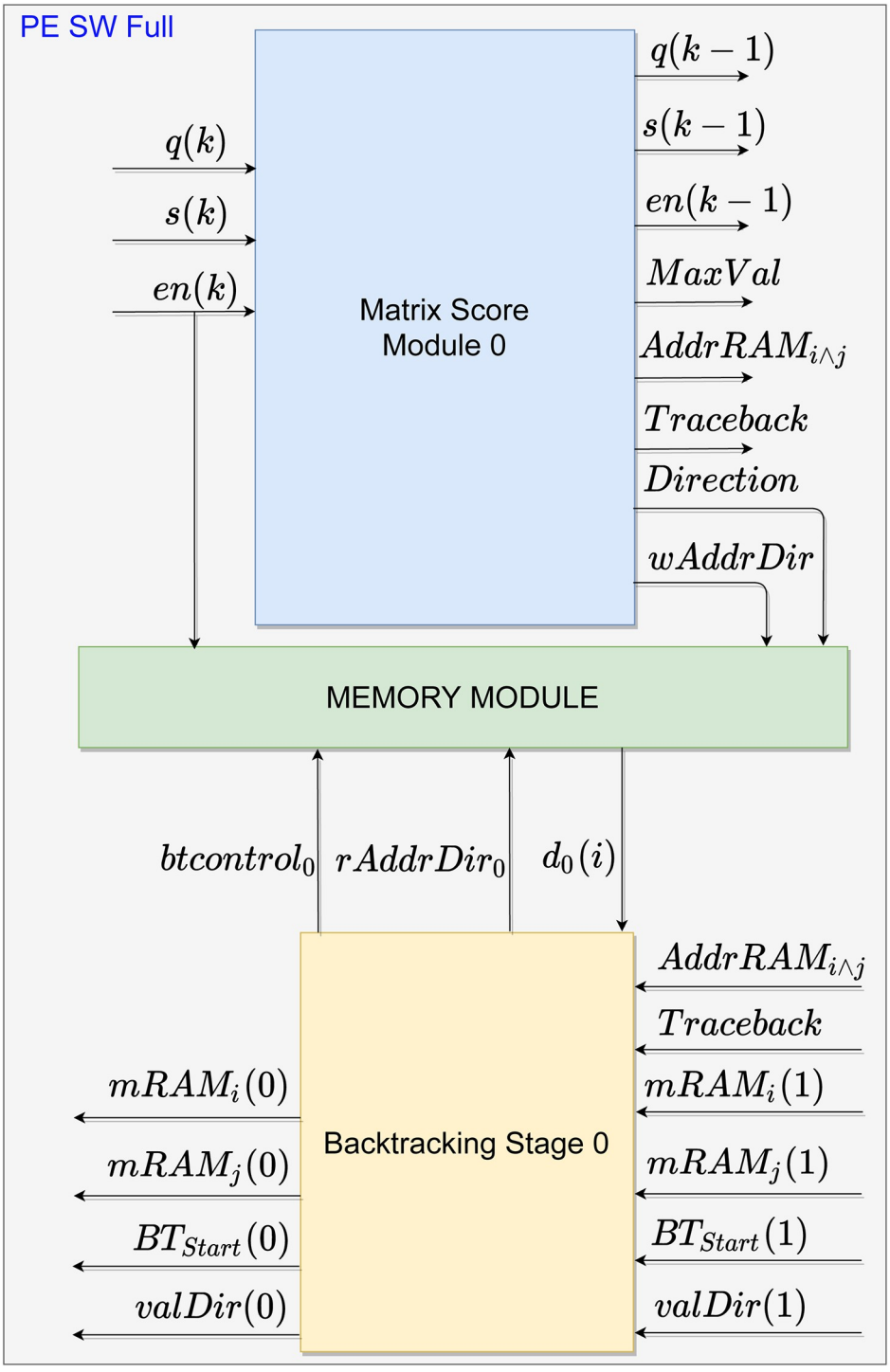

Fig 3. Architecture of each PE in the systolic array. The forward stage is represented by the blue block, the backtracking stage by the yellow block, and the Memory by the green block.

https://doi.org/10.1371/journal.pone.0254736.g003

Table 1. Description of signals and the algorithm stage they are used. The forward stage is represented by F, storage stage by F, and backtracking stage by B. They are shown in the Figs 4, 7 and 8.

Signal	Stage	Description
$\mathbf{q}(k)$	F	query sequence, to be compared with the database sequence.
$\mathbf{s}(k)$	F	database sequence.
$\mathbf{en}(k)$	F,S	sequence that enables the PE cells.
$\mathbf{Sc}(i)$	F	vector of scores.
$MaxVal$	F	maximum score.
$Addr\ RAM_{i\wedge k}$	F,B	memory address of the highest maximum score.
$index$	F,B	corresponding addressing of the modules.
$Direction$	F,S	calculated value of the direction to be stored in RAM.
$wAddrDir$	F,S	storage address corresponding to the direction in RAM.
$Traceback$	F,B	flag to indicate the start of the backtracking in PE $N-1$.
$btcontrol$	S,B	flag to change the state of the write-to-read memory
$rAddrDir$	S,B	choosing the corresponding value for reading in RAM.
$d(i)$	S,B	return of the value of the path passed from the RAM.
$mRAM_{i\wedge k}$	S,B	addresses of the path to be followed in the alignment.
BT_{Start}	B	enable flag of the backtracking after *Traceback*.
BT_{Next}	B	flag for enabling the internal circuits to choose and process.
$valDir$	B	alignment path value for that PE.
$path(j)$	B	memory position of the current alignment in the module.

https://doi.org/10.1371/journal.pone.0254736.t001

N and M, which define the size of the matrices **H** and **D**, respectively. The Matrix Score Module (MSM) calculates the scores and distances in columns of matrices **H** and **D** in parallel.

The systolic array structure developed for the matrices is composed of N PEs. Therefore, for each j-th element in **q**, there is a j-th PE. It is based on dividing the construction of the **H** score matrix expressed by

$$\mathbf{H} = [\mathbf{g}_0, \ldots, \mathbf{g}_j, \ldots, \mathbf{g}_{N-1}], \tag{6}$$

and finding the best path in which the **D** matrix returns the correct sequence alignment, which in turn is equivalent to the directional flags that determined the alignment path. Moreover, for each PE_j (which represents a column of the matrix **H**) there is i-th **s**(**i**) that varies from 0 to $M-1$, according to the following

$$\mathbf{g} = \begin{bmatrix} g(0) \\ \vdots \\ g(i) \\ \vdots \\ g(M-1) \end{bmatrix}. \tag{7}$$

The number of MSMs submodules corresponds to the number of elements in **q**, i.e., $\{j \in \mathbb{N} \,|\, 0 \leq j < N\}$, as can be observed in Fig 4. Therefore, **H** is formed by N columns, according to Eq 6. Besides, the MSM also calculates the path, the maximum score value and its position, which are subsequently stored in the Memory Module (MM).

The SW algorithm in this work is initialized by the $en(k)$ signal, which enables the memory components in the MSM and MM modules to allocate the two sequences $q(k)$ and $s(k)$. The $en(k)$ is a sequence of pulses of value 1 with size equal to the **s** sequence. Thus, the sequences

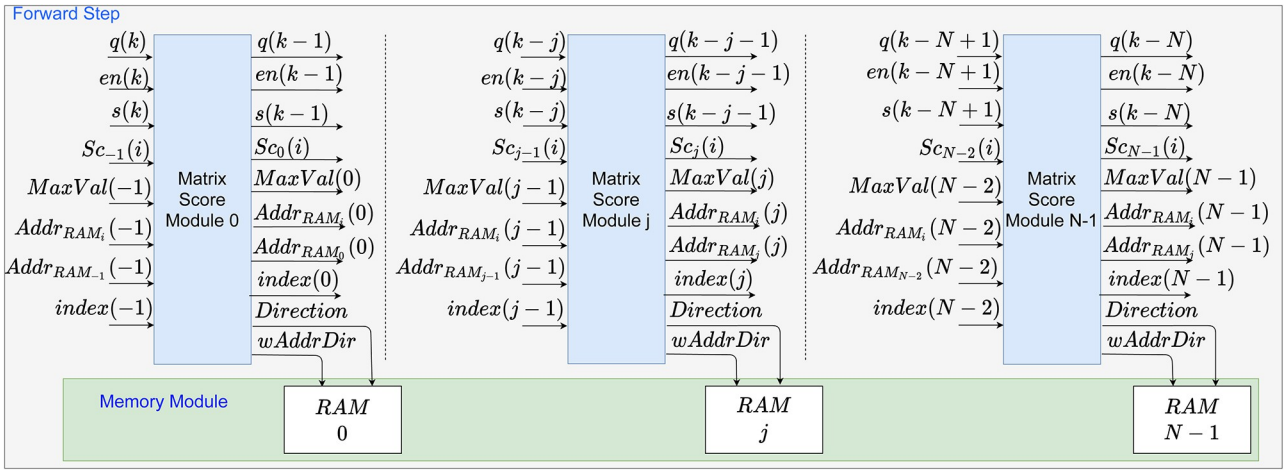

Fig 4. Hardware representation of the H score matrix on the forward stage. The modules are generated from 0 to $N-1$.

https://doi.org/10.1371/journal.pone.0254736.g004

are transmitted at each sampling time to the forward module. The signals are received in MSM, and the respective $q(k)$ is allocated according to its position, while $s(k)$ is propagated to the MSM based on the internal counter within each module. The counters within each MSM module are activated with each pulse of the $en(k)$ signal.

Each k-th q element is compared to all elements in **s**, iteratively, i.e., the **s** is traversing, going from the first PE to the last one. If the values are equal, a value from the *Match* constant is propagated; otherwise, the value of *Mismatch* is propagated. *Match* corresponds to a reward for similarity, while *Mismatch* is a penalty for inequality between values. Afterward, the addition block sum the values according to

$$\mathbf{g}_j(i) = \begin{cases} \mathbf{g}_{j-1}(i-1) + \alpha & q_j = s(i) \\ \mathbf{g}_{j-1}(i-1) + \beta & q_j \neq s(i) \end{cases}, \tag{8}$$

where α and β are arbitrary values that correspond to the match value and mismatch values, respectively.

Subsequently, the score value, $\mathbf{Sc}_{j-1}(i-1)$, and correspondence value, $\alpha \wedge \beta$, are added to define a portion of $\mathbf{g}_j(i)$. The $\mathbf{Sc}_{j-1}(i-1)$ value is equivalent to the $\mathbf{H}(i-1)(j-1)$ value (i.e., $\mathbf{g}_{j-1}(i-1)$). The values of $\mathbf{Sc}_{-j}(i)$, $MaxValue(-1)$, $AddrRAM_{i \wedge j}(-1)$ and $index(-1)$ are initialized with 0. At the same time, the $\mathbf{Sc}_{j-1}(i)$, which is the score value of the previous block, it is received and operated with the value of *Gap*. In addition, the value of the scoring operation of this block in the previous time, $Sc_j(i-1)$, is also operated with the *Gap*. Thus completing the computation of $g_j(i)$ that can be seen in the Eqs 5 and 9.

Fig 5 shows the submodule that constitutes each MSM module. The three blocks in pink are used to perform the addition and subtraction operations, representing the SW's relations to generate the M elements. Thereby, the process of choosing the maximum value among the

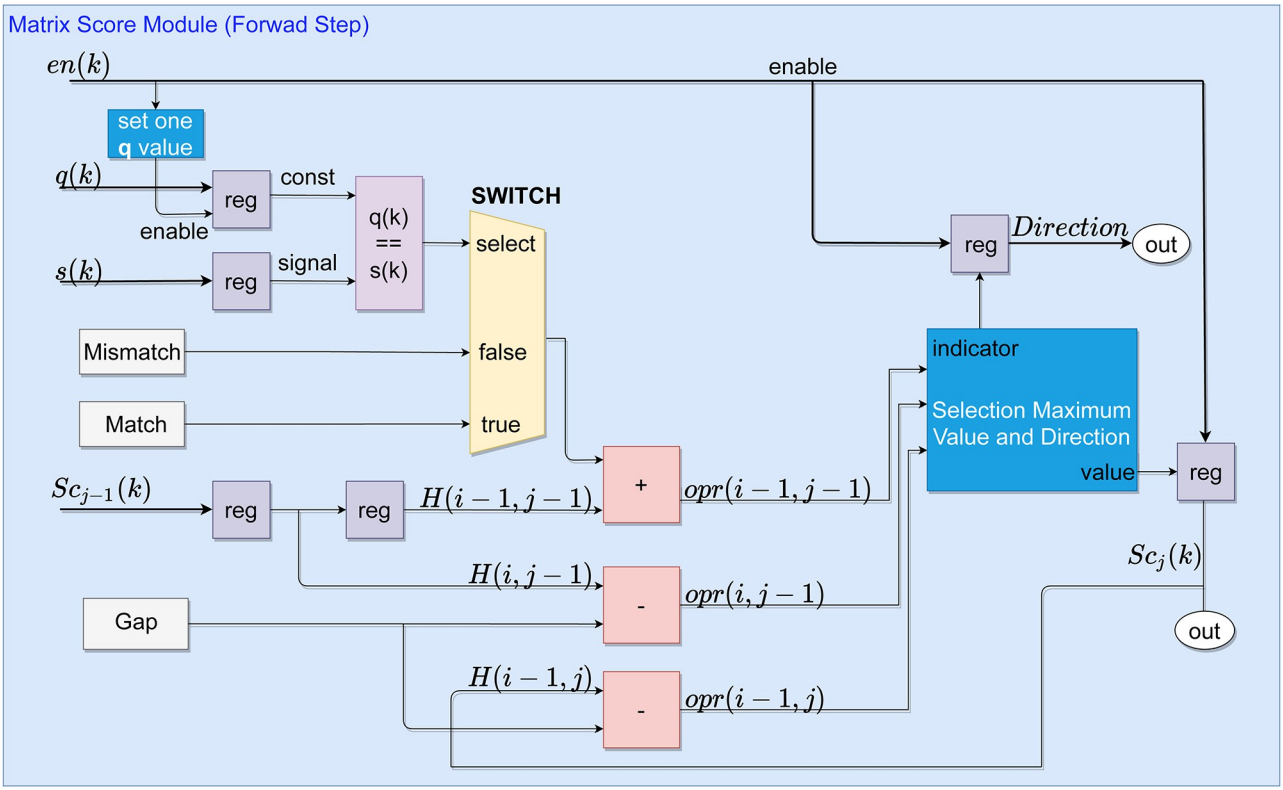

Fig 5. Submodules that constitute a Matrix Score Module. The representation of the circuit and signals is only related to the forward stage.

https://doi.org/10.1371/journal.pone.0254736.g005

calculated scores is carried out based on Eq 5 as follows

$$\mathbf{g}_j(i) = \max \begin{cases} 0 \\ \mathbf{g}_{j-1}(i-1) + \alpha & q_j = s(i) \\ \mathbf{g}_{j-1}(i-1) + \beta & q_j \neq s(i) \\ \mathbf{g}_{j-1}(i) - \gamma \\ \mathbf{g}_j(i-1) - \gamma \end{cases}, \tag{9}$$

where γ is an arbitrary value that represents the chosen linear gap value. This expression is equivalent to Eq 3.

The output of the pink blocks, called *opr*, are propagated to the next submodule for choosing the maximum score and distance path, as shown in Fig 5. This submodule is built with a set of multiplexers and relational circuits that can find the maximum score value with the coded distance of the path by comparing the *opr* signals, as seen in Fig 6.

Selecting path distances is based on a simple encoding of three levels representing the alignment action to be adopted: 2, 1, and 3. Therefore, the levels 2, 1, and 3 represent a match, a gap in the target sequence **q** and **s**, respectively, as described in Section 2. The encoding process of directions is performed in the forward step, as illustrated in Fig 6. During this process, the same signals used to calculate the **H** score matrix are needed, i.e., the $opr_{j-1}(i–1)$, $opr_{j-1}(i)$ and $opr_j(i-1)$, as seen in Fig 5. These values are compared in relational circuits and subsequently chosen according to the criteria of the SW, as seen in the Fig 6.

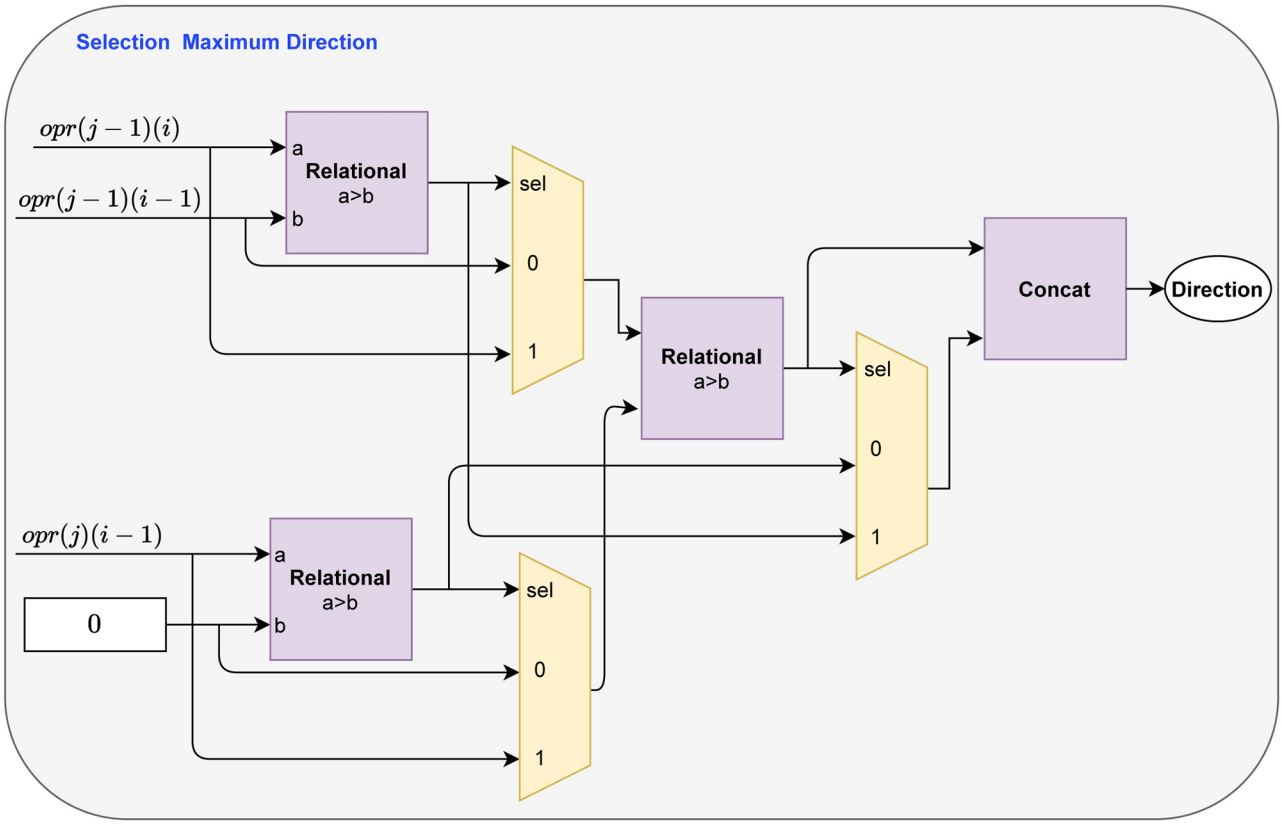

Fig 6. Circuits that constitute the submodule for finding the maximum score and distance path within an MSM. The relational circuits are represented in purple and the multiplexers in yellow.

https://doi.org/10.1371/journal.pone.0254736.g006

Then, for demonstrating the realization of the path coding process is done, the information in Fig 1 is used. When looking at the Fig 1, four variables are distributed in an **H** score matrix. The variables $x = H(i-1, j-1)$, $y = H(i-1, j)$ and $v = H(i, j-1)$ are known values, while w is a score to be computed. Starting from $w = H(i, j)$ as the observed cell for determining a generic path and x, y and v as the neighborhood. An integer value is associated with the d_j corresponding to the address of w, according to the maximum value determined in the neighborhood, these values are assigned according to the expression

$$\mathbf{d}_j(i) = \begin{cases} 1 & | \quad y - \gamma > x + (\alpha \vee \beta), y > v \\ 2 & | \quad x + (\alpha \vee \beta) \geq y - \gamma, x + (\alpha \vee \beta) \geq v - \gamma \\ 3 & | \quad v - \gamma > x + (\alpha \vee \beta), v > y \end{cases}, \tag{10}$$

where 1, 2 and 3 is the vertical, diagonal, and horizontal paths, respectively. The Eq 10 is equivalent to the circuit implementation illustrated in the Fig 6, where $(x + (\alpha \vee \beta)) = opr_{j-1}(i-1)$, $(y - \gamma) = opr_j(i-1)$ and $(v - \gamma) = opr_{j-1}(i)$. Besides, $(\alpha \vee \beta) = \alpha$ for a match and $(\alpha \vee \beta) = \beta$ for a mismatch.

Algorithm 1: SW foward stage pseudo-code based in structure this proposal

```
Input: query sequence q
Input: dataset sequence s
Output: distance path matrix D
Output: row position of maximum score posMi
Output: column position of maximum score posMj
```

```
//length query sequence N, length dataset sequence M, match value α,
mismatch value β and linear gap value γ;
for for k = 0 to M × N step 1 do do
  Initialize the DP matrix H and D with zeros;
end
//Forward Stage;
for for j = 0 to N − 1 do
  for for i = 0 to M − 1 do
    if q(j) = s(i) then
      sel ← α;
    else
      sel ← β;
    end
    //H(i + 1, j + 1) computation;
    x = H(i, j) + sel; y = H(i, j + 1) + γ; v = H(i + 1, j) + γ;
    score ← 0, direction ← 0;
    if x > y ⋀ x > v then
      score ← x; direction ← 2;
    else
      if y > v then
        score ← y; direction ← 1;
      else
        score ← v; direction ← 3;
      end
    end
    // Stores the score in matrix H and the direction in matrix D
calculated;
    H(i + 1, j + 1) = score; D(i + 1, j + 1) = direction;
    // checking which is the highest calculated score;
    if maxVal < score then
      maxVal ← score; posMi ← i + 1; posMj ← j + 1;
    end
  end
end
return D, posMi, posMj;
```

After the process of selection the score and direction, it has the choice of the maximum score based on a logic of multiplexers and relational blocks. There is a counter, called *cntR*, to determines the number of times that the selection of the score and direction is carried out, i.e., the **H** matrix row that the process is on. This is necessary to determine the $AddrRAM_i$ address. At the beginning of MSM processing, $index(j-1)$ is added to 1, just once for each MSM, becoming $index(j)$ and determining the address of this MSM. For the determination of *Maxval*, it is seen whether the previous value is less than the current computed score value, then the calculated current score value becomes the *Maxval*, $AddrRAM_j = index(j)$ and respective row process value is $AddrRAM_i$. It is noted the $AddrRAM_{i \wedge j}$ signal are corresponding to the location of the maximum score value.

In parallel with the process of determining the maximum score value, there is the process of storing the directions. Thus, the output *Direction* of the submodule is prepared in set with the value *wAddrDir*, which comes from the **H** matrix row calculated at that moment, allowing to write in order in RAM memory according to the respective positions of **H** matrix (i.e., same position of **D** matrix).

Finally, according to the systolic structure, after the MSM processing is over, the signals are parallelly sent to the next MSM. Thereupon, $q(k)$, $s(k)$, and $en(k)$ are shifted in time, that is, $q(k-1)$, $s(k-1)$, and $en(k-1)$, to match the calculation structure of the **H** matrix, as seen in the Fig 4. Besides, the calculated signals $Sc(i)$, *MaxVal*, $AddrRAM_{i \wedge j}$ and *index* are also

propagated to the next MSM to preserve the scores calculating structure. This process repeats until the last element of **s** is calculated with the last element of **q**; a counter in is used to determine that moment since the values of the sequences are previously informed to all PEs. The forward stage finishes with the calculation of the last element of the matrix, i.e., $\mathbf{H}(M-1)(N-1)$. Consequently, the signal *Traceback* is enabled, indicating the end of the process in all MSM, and the addresses $AddrRAM_{i \wedge j}$ corresponding to the maximum score value is sent to the next step (i.e., backtracking process).

Algorithm 1 presents the SW pseudo-code for forward stage and storage process structures. The Algorithm 1, is prepared to perform the calculation of scores and storage of matrices **H** and **D**. The input is the signals **q** and **s**, which is Eqs 1 and 2, respectively. The first loop, in the Algorithm 1, represents each *N* element used, as seen in Fig 2. The second Loop is the interactions made by the signal *En* to allow the calculation of each element of s in each PE. The first conditional structure is the multiplexer for making choices in the MSM, as seen in Fig 5. Submodule Selection Maximum Value and Direction, Fig 6, is represented by the second conditional structure, which compares variables *x*, *y* and *v*. The outputs are **D** matrix stored in MM and the position of the maximum values defined in MSM.

3.2 Memory Module (MM)

The MM communicates with both the MSM and the BS, as shown in Fig 7. During the forward stage, the data regarding the distance values are written to the MM. Meanwhile, during the BS, the memory addresses to align the sequences are fetched from the MM. The size of each memory is defined by the size of the **s** sequence; also, there is a flag to indicate that the memory is in write mode while computing the **H** matrix and, subsequently, in fetch mode, in the backtracking process.

The MM consists of Random Access Memories (RAMs) used to store the path directions, *Direction*, obtained in the MSM that is thereafter needed in the BS module. Hence, the RAMs are in write mode throughout the forward stage and reading mode during backtracking. The RAM input ports are the address and data busses and write enable mode. Besides, the memory size of each memory is defined based on the size of the sequence **s**, which in turn, the amount of RAM memories is equal to the number of PEs in the systolic array.

The enable signal, *en*, is used as write enable for each RAM in the MM. Therefore, $en = 1$ defines the write mode, while $en = 0$ the read mode. In addition, the *btcontrol* signal selects which module controls the RAM address bus. Hence, for $btcontrol = 0$ the memory addresses are defined by the MSM module through *wAddrDir* signal, while $btcontrol = 1$ selects the BS module to define the addresses via *rAddrDir* signal.

Thus, in write mode ($en = 1$ and $btcontrol = 0$) the *wAddrDir* signal defines the address of the RAMs where the *Direction* value is stored by the MSM. Subsequently, after the **H** matrix is fully calculated, the *Traceback* is enabled to indicate the end of the forward stage, and the MM goes into reading mode ($en = 0$ and $btcontrol = 1$). Accordingly, the *rAddrDir* signal defines the address space the BS fetches the data corresponding to the value reported by the traceback.

3.3 Backtracking approach

The backtracking process starts when the *Traceback* signal is enabled in the MSM by counters that determine the last PE and the last processed element of **s**, as described in forward stage. As previously mentioned in subsection 3.1, the MSM propagates to the MM the maximum score address that is used as the starting point for alignment, as shown in Fig 8. Meantime, the Fig 9 details the submodules used to create each BS module. The submodules in green are

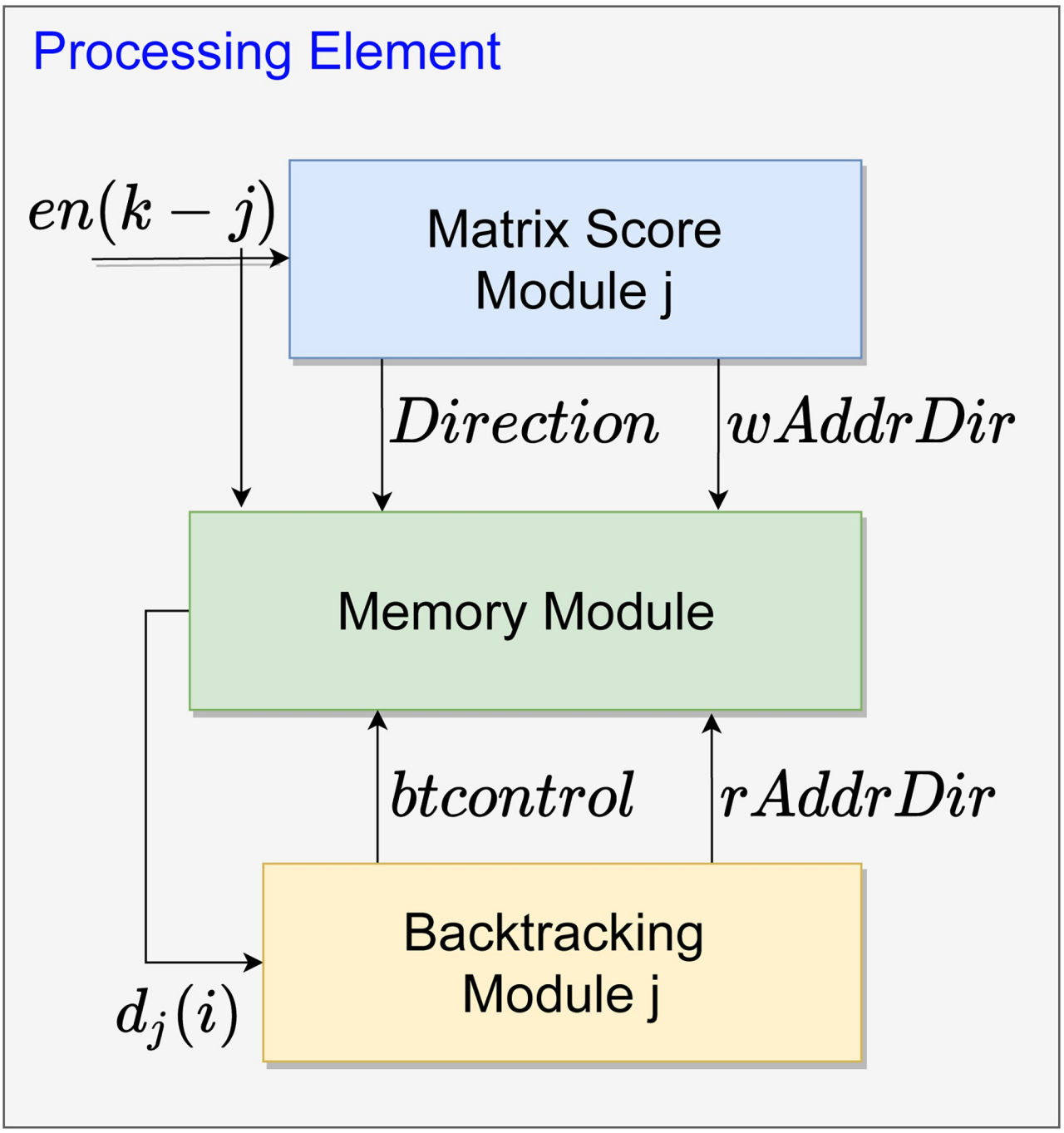

Fig 7. Representation of the simplified Memory Module structure. This model is practically as is the complete processing PE of each column of the **H** matrix.

https://doi.org/10.1371/journal.pone.0254736.g007

circuits for controlling and synchronizing all signals during the module operation, while the blue submodule performs the alignment path described in this section.

Firstly, after *Traceback* is enabled, the *BTStart* signal is enabled, and the addresses of the maximum score element, $Addr_{RAMi}(N-1)$ and $Addr_{RAMj}(N-1)$, are sent to the respective BS. Also, the values of $Addr_{RAMi}(N-1)$ and $Addr_{RAMji}(N-1)$ are assigned to $mRAMi(N-1)$ and

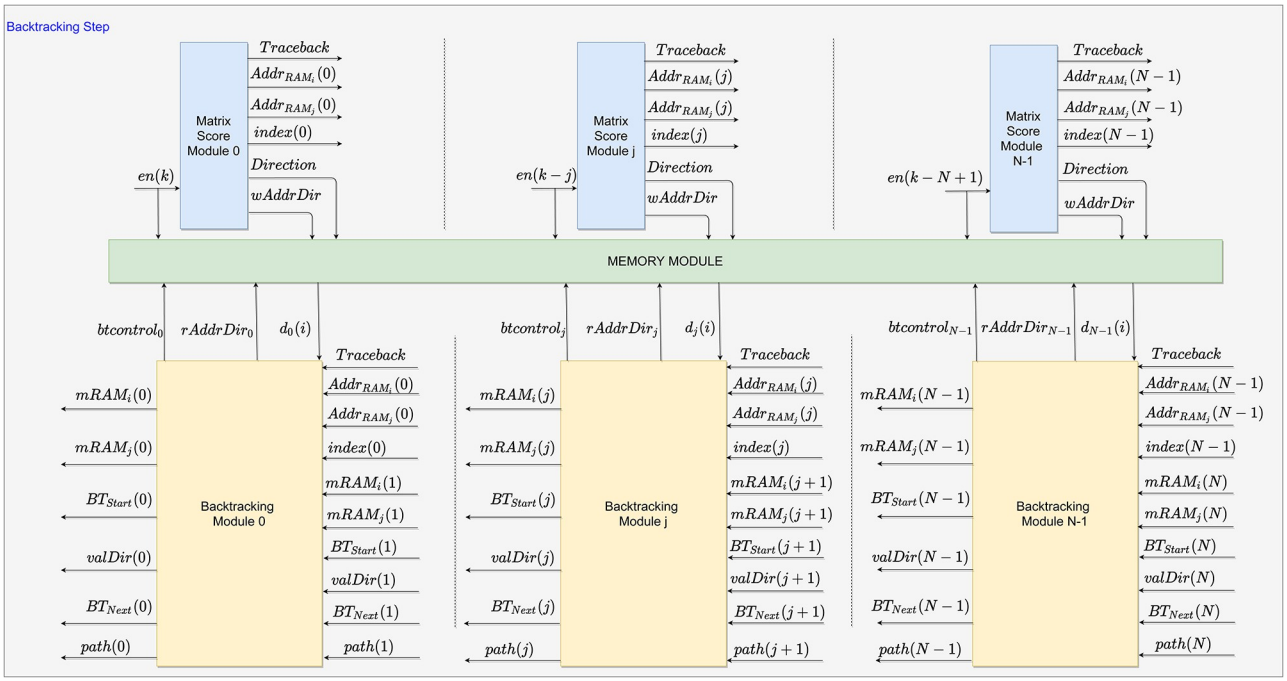

Fig 8. Backtracking Module structure in the FPGA. The operation of this block starts after the forward step.

https://doi.org/10.1371/journal.pone.0254736.g008

$mRAMj(N-1)$, respectively, by the BT Enable submodule. It is important to emphasize that if the $mRAMj(N-1)$ value (i.e., $Addr_{RAMj}(N-1)$) is not already in the BS PE, it will trace-back by checking the Memory Index submodule. This process happens until it reaches the PE corresponding to the maximum score location. Afterward, the Memory Index submodule assigns

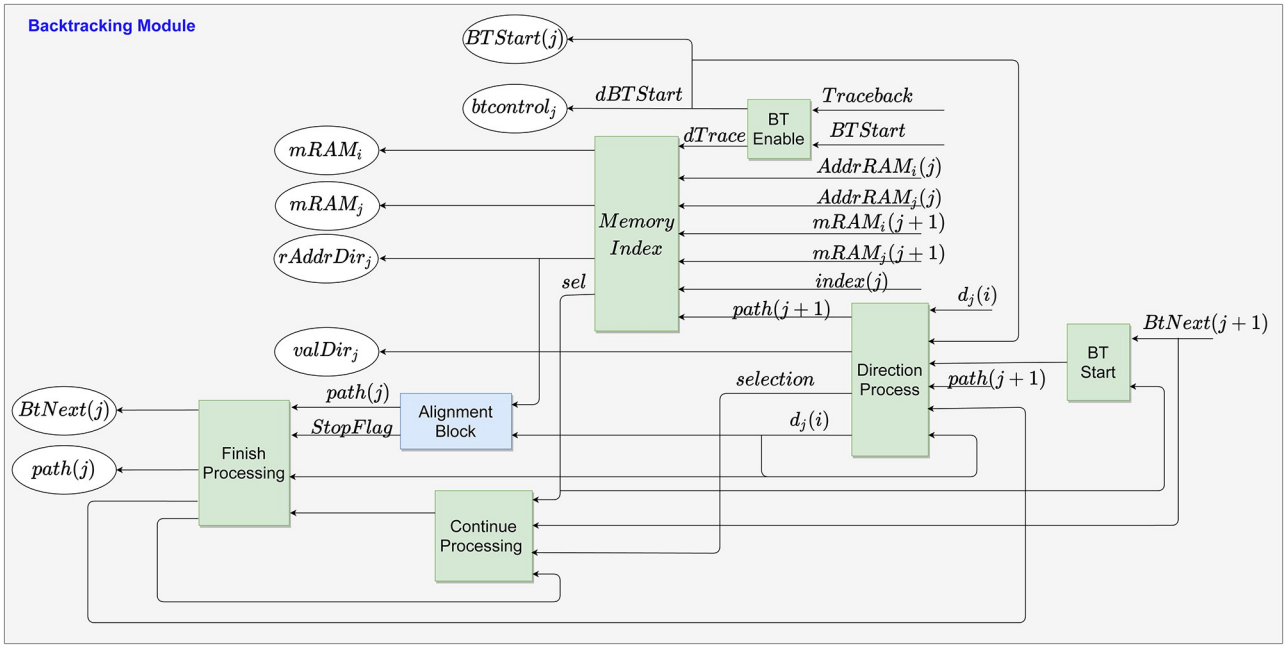

Fig 9. Submodules that constitute the backtracking stage module. The green submodules represent the control submodules, while the blue submodule represents the circuit that performs the alignment.

https://doi.org/10.1371/journal.pone.0254736.g009

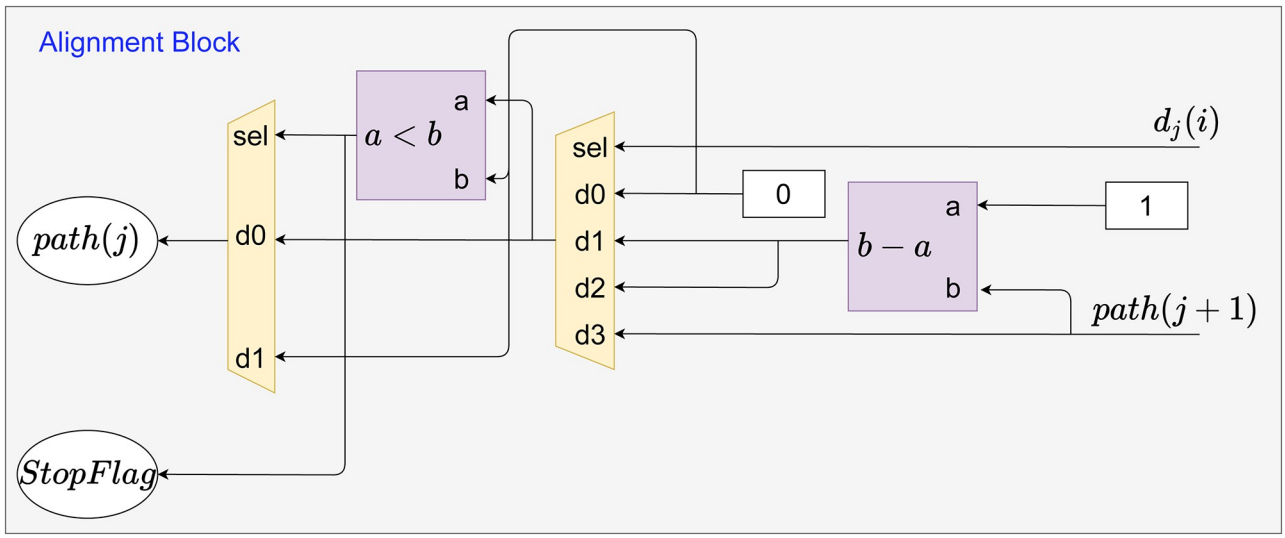

Fig 10. Logical circuits used to build the Alignment Block submodule.

https://doi.org/10.1371/journal.pone.0254736.g010

$mRAM_i$ value to *rAddrDir* to read the memories in the MM, which in turn, returns the $d(i)$ value to the Direction Process submodule, as can be seen in Fig 9.

Secondly, the alignment process starts. The circuits used to build the alignment submodule are shown in Fig 10. As can be observed, the input $d_j(i)$ is used as the multiplexer selector to perform the Eq 10. Therefore, for $d_j(i) = 3$, BS remains in the same memory position and moves back one BS module, i.e., horizontal displacement. While for $d_j(i) = 1$, only the memory position decreases by 1, and BS is verified by the Direction Process and Continue Processing submodules (i.e., vertical displacement). Meanwhile, for $d_j(i) = 2$, the memory position also decreases by 1, and it moves to the previous module with the displacement in the memory position. The circuit after the first multiplexer prevents negative addresses in the memory.

Given that the path to align the first element is found, the Alignment Block submodule receives the $rAddrDir_j$ and $d_j(i)$ signals to define the path to be followed by the next BS, as seen in Fig 9. Initially, a logical circuit enables the BT Start and Direction Process submodules to propagate those signals to the Alignment Block. The Direction Process and Continue Processing submodules carry out checks to define which BS module is active, that is, for $d_j(i) = 1$, the data processing is held in the current BS module, and for $d_j(i) \neq 1$, the signal BT_{Next} is enabled, indicating the end of data processing in the current PE to start in the next one.

After finding the module for the maximum score, the $mRAM_i$ and $mRAM_j$ signs finish their function. Thus, from the determination of the BS with the maximum score, the $path(j)$ sign is used as a guide for locating the alignment of each module. Then, the data in MM is requested and the d_j value is returned for verification and establishment of alignment. The verification and establishment of the alignment path is done by the Memory Index, Direction Process, and Continue Processing submodules. Decisions related to d_j value are made in Alignment Block submodule, as illustrated in Fig 10.

Finally, the Finish Processing and Continue Processing submodules finish the data processing in the module. Thereby, the *valDir* output of each submodule is used to construct the alignment path, along with the maximum score position values. The trace-back continues until it reaches PE_0 or finds a path direction with a value of 0.

Algorithm 2 presents the SW pseudo-code for backtracking stage this proposal. The backtracking stage, algorithm 2, is ready to perform the alignment in a list using the path informed

in **D**, starting from the positions of the maximum score, as seen in this Section. Inputs for this step are provided by Algorithm 1. The loop for this step represents all backtracking stage modules from $N-1$ to 0. The conditional structure of Algorithm 2 is the representation of submodule Alignment Block, Fig 10, which allows it to trace-back. And the return of the alignment path is storing the data, *valDir*, in RAM memory.

Algorithm 2: SW backracking stage pseudo-code based in structure this proposal

```
Input: query sequence q
Input: dataset sequence s
Input: distance path matrix D
Input: row position of maximum score posMi
Input: column position of maximum score posMj
Output: alignment sequences list A
Output: alignment path sequences list path
//Backtracking Stage;
auxi = posMi; auxj = posMj,; aux ← D(auxi, auxj);A ← [];
path ← concat(path, aux);
while aux > 0 do
  A ← concat(A, [q(auxj - 1); s(auxi - 1)]);
  if aux = 2 then
    auxi = auxi - 1, auxj = auxj - 1;
  else
    if aux = 3 then
      auxj = auxj - 1;
    else
      if aux = 1 then
        auxi = auxi - 1;
      else
        break;
      end
    end
  end
  aux ← D(auxi, auxj);
  path ← concat(path, aux);
end
return A, path;
```

4 Results and discussion

This section presents the synthesis results for the architecture described in the previous section and analyses it regarding the following key points: critical path, operation frequency, number of PEs, and performance. The performance measures the time to calculate an element of the scoring matrix.

The development of the algorithm was carried out using the development platform provided by the FPGA manufacturer, in this case, Xilinx [60]. This platform allows the user to develop circuits using the block diagram strategy instead of VHDL or Verilog. The architecture was deployed on the FPGA Virtex-6 XC6VLX240T and compared to state-of-the-art works. Usually, hardware implementations of the SW algorithm in the literature were implemented only the forward stage or both the forward and backtracking stages. In our proposal, both stages were implemented.

The performance for hardware implementations of the SW algorithm is usually measured in Giga Cell Update Per Second (GCPUS), which in turn is defined as

$$\text{GCUPS} = \frac{\text{number of cells}}{\text{total processingtime} \times 10^9}, \tag{11}$$

in which a cell can be a matrix element to be computed. This metric can also be described based on the clock frequency, that is,

$$\text{GCUPS} = \text{number of cells} \times \text{clock frequency} \times 10^{-9}. \quad (12)$$

The latter equation is often used to compare the systolic array efficiency. Since the number of cells is equivalent to the number of PEs, and the clock frequency defines the operating frequency, it is unnecessary to measure the total runtime of the algorithm.

4.1 Hardware architecture validation

To validate the architecture proposed in this work, the sequences **q** and **s** were randomly generated and varying the match, mismatch, and linear gap values. Initially, the analysis was carried out for 8 PEs and by varying the size of the sequences **q** and **s** from 8 to 32. The number of PEs also varied according to the size of **q**. Our architecture works with sequences of varying lengths, only requiring that the length of **s** is greater than or equal to the **q** length.

Firstly, the correctness of the matrices **H** and **D** was verified by monitoring the MSM outputs, such as *Sc* and *Direction*, as described in Section 2. Secondly, it was verified if the **D** matrix elements were stored in the correct memory positions in the MM. Lastly, the operation of the BS modules was also verified by monitoring the $path(j + 1)$ bus and the Memory Index submodule.

Following, the Alignment Block and Direction Process are observed to check if the memory accesses are in accordance with the $path(j + 1)$ value, that is, according to Eq 10. Also, the Finish Processing and Continue Processing submodules are monitored to verify the values propagated for a match (2), horizontal gap (3), and vertical gap (1).

The data bit-width was defined by the maximum size of the input sequences, limited by FPGA memory capacity. Hence, the input sequence bit-width was set to 3 while constants were defined according to its value. Besides, the bit-width for the MSM buses that perform mathematical operations was defined as $\log_{total\ -\ PEs} \times \alpha$. Meanwhile, the sequence counters for **s** is $\log_{s\ -\ size}$.

Fig 11 shows the architecture deployed and running on the Virtex-6 FPGA. The host computer (i7-3632QM CPU and 8GB of RAM) was used to plot the results and compare them to a software implementation presented in [61], as shown in Fig 12. In the Fig 12, it can be seen that the y axis refers to the **s** sequence, while the x axis refers to the **q** sequence. To increase the resolution of the image, only the parts of the sequences that are aligned are used, where the position at which the alignment starts and the maximum score value are shown in the title of the illustration. The value of Row refers to the position in the **s**, whereas Column is related to the element of the **q**. The amount of sequence alignment performed is represented by Number of Alignments.

The architecture parameters for the demo were set to $match = 5$, $mismatch = -5$, $gap = 1$, and 128 PEs. Hence, the size of the sequence **q** is also 128. Meanwhile, the size of the sequence **s** was set to 8, 192, resulting in a total of 1, 048, 576 calculated cells. Sequence **q** is loaded into memory at each iteration, where it can vary between 4 different 128 nucleotide sequences in the demonstration. The demo is available at [62], and the implementation source code is available at [63].

The SW architecture was developed using the Xilinx System Generator on Matlab, and the traffic of data between the host PC and the FPGA was accomplished via Ethernet protocol. Moreover, we added on the FPGA a buffer to store data and developed a *manager circuit* to control the data flow. Therefore, the **q** and **s** sequences were transferred to the FPGA (via

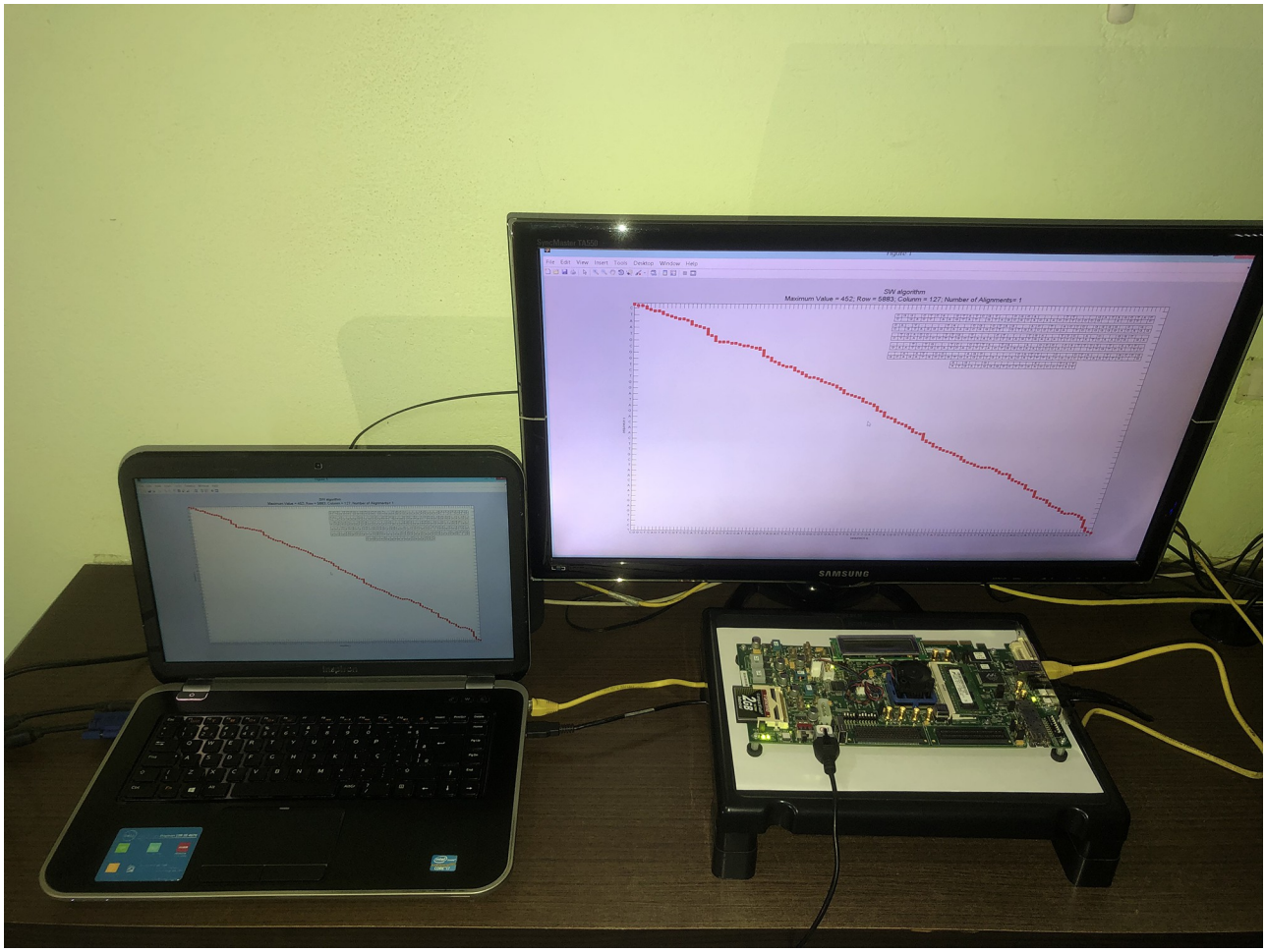

Fig 11. Photo of the hardware architecture deployed on the Virtex-6 FPGA and the host computer used to plot the results.

https://doi.org/10.1371/journal.pone.0254736.g011

Ethernet protocol) and stored in the buffer, and, subsequently, fed into the SW architecture to perform the alignment by the manager circuit.

4.2 Synthesis analysis

Analysis of the synthesis results for the SW hardware implementation were carried out for two FPGAs: Virtex-6 XC6VLX240T and Virtex-7 XC7VX485T. Table 2 presents the hardware area occupation and frequency for a different number of PEs. The size of the input sequences were defined according to the number of PEs.

The critical path of the design was ≈8.34ns and ≈6.44ns for the Virtex-6 and Virtex-7, respectively. Therefore, the maximum clock frequency was 120MHz for the Virtex-6 and 155MHz for the Virtex-7. Regarding the FPGA area occupation, increasing the number of PEs also increases the hardware resources used. For 512 PEs in the Virtex-6, a total of 68% of the Slice Look-Up Tables (LUTs) were used in contrast to only 7% for 64 PEs. Concerning the frequency, a slight decrease is observed as the number of PE increases due to an increase in the critical path. Concerning the Virtex-7, there are unused FPGA resources as less than 35% of Slices LUTs were used. Therefore, it can be used to increase the number of PEs and, thus, the performance. Note that, increasing the number of PEs and, consequently, the size of the

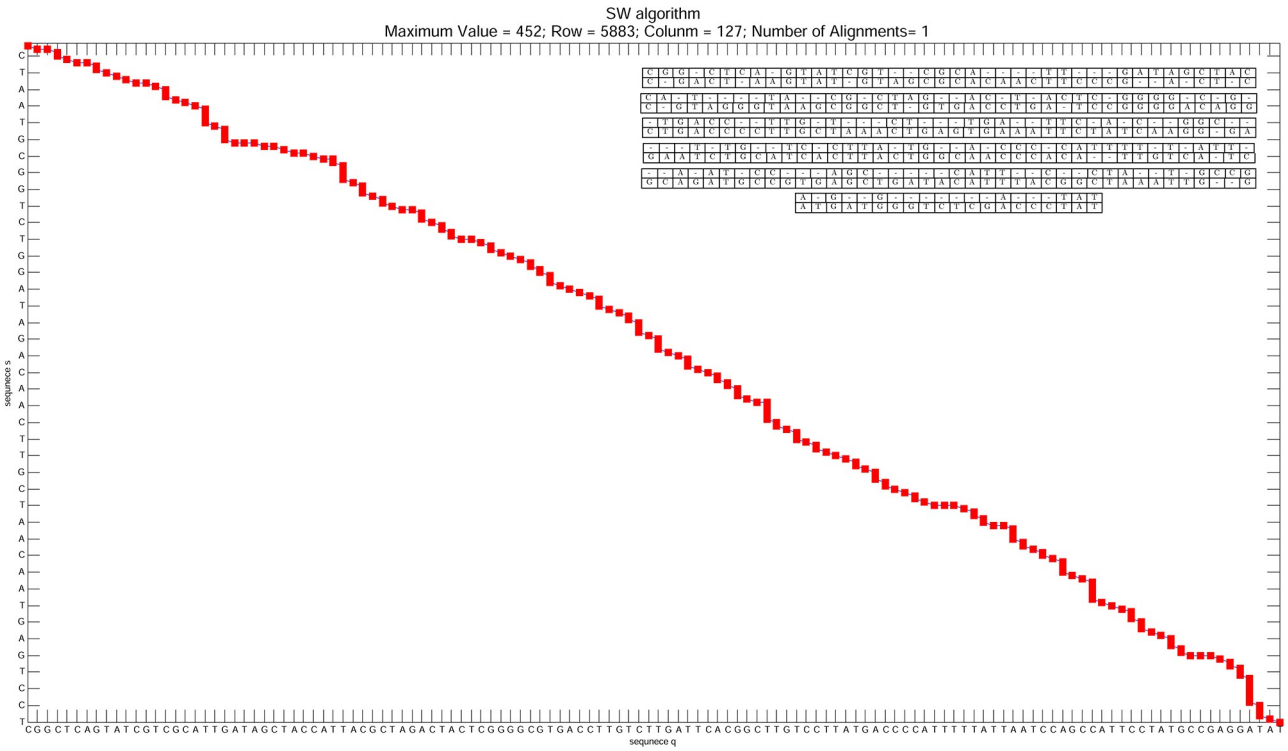

Fig 12. Illustration of the results obtained from our proposal in co-simulation. The image is the most detailed representation of the monitor in Fig 11. It can see that the y-axis refers to the **s**, while the x-axis refers to the **q**. The position at which the alignment starts is indicated by Row and Column. The maximum score value found is presented by Maximum Value. The amount of sequence alignment performed is represented by Number of Alignments.

https://doi.org/10.1371/journal.pone.0254736.g012

sequence **q**, the number of parallel computations will also increase, thus, improving the performance. Therefore, according to the resources available in the target hardware, our architecture can operate with a number significantly bigger than 512 PEs.

4.3 Comparison with other works

Comparisons with state-of-the-art works were also performed. The performance of systolic array-based implementations increases with the number of PEs. Hence, the comparisons were carried out for the maximum number of PE in each proposal. We compared our design to the most relevant and recent works similar to our proposal, i.e., the SW algorithm has to be implemented using a systolic-array structure, deploy the backtracking step, and provide the parameters concerning processing time and area occupation. Given that, we discuss a direct

Table 2. Area occupation results based on the FPGA synthesis of our SW implementation, with forward and backtracking stages.

FPGA Model	Array Size (PE)	Slice LUTS /(%)	Memory /(%)	Frequency (MHz)
Virtex-7	512	103, 778 (34%)	8, 192(6%)	155
Virtex-6	512	103, 807 (68%)	8, 192(14%)	120
Virtex-6	256	47, 725 (31%)	2, 048 (3.5%)	112
Virtex-6	128	24, 386 (16%)	512 (1%)	117
Virtex-6	64	10, 803 (7%)	128 (0.2%)	157

https://doi.org/10.1371/journal.pone.0254736.t002

Table 3. Table adapted from paper [23].

Related Works	Backtracking (Yes or No)	PE number in array	Frequency (MHz)	Performance (GCUPS)
2005 [64]	No	252	50	13.9
2006 [39]	Yes	303	77.5	23.5
2007 [65]	No	384	66.7	25.6
2007 [66]	No	128	125	16.0
2008 [44]	Yes	256	100	25.6
2009 [57]	Yes	168	62.5	10.5
2011 [67]	No	100	111	11.1 × 12
2012 [68]	No	–	250	–
2012 [69]	No	<100	125	16 × 8
2012 [70]	No	100	175	17.5
2012 [59]	No	128	60	7.62
2014 [71]	No	200	200	40
2017 [23]	Yes	512	150	76.8
2017* [23]	Yes	512	200	105.9
2018 [48]	No	776	600	465
2018 [46]	No	-	-	125
2018 [52]	Yes	-	200	-
2020 [50]	Yes	-	200	-
2020 [47]	Yes	256	500	128
2021 [41]	Yes	512	308	79.65
2021 [51]	Yes	-	582	-
2021 [53]	No	-	200	51.20
This work	Yes	512	155	79.5

It compares the proposed SW using reconfigurable hardware based on the operating frequency, number of PEs and performance in GCUPS. It also shows if the work use backtracking or not in the implementation.

* indicates the approach uses external memory to accelerate the alignment process.

https://doi.org/10.1371/journal.pone.0254736.t003

comparison to the architecture presented in [23, 41, 47]. The remaining works shown in Table 3 illustrate general results of other FPGA implementations of the SW.

The works presented in Table 3 are available in [23]. The second column indicates whether the backtracking stage was also developed on FPGA or only the forward. Meantime, the third to fifth columns present the number of PEs, operating frequency, and performance, respectively. The performance was obtained according to Eq 12. As can be seen, our approach and the one proposed by [23] were the only ones to implement a high number of PEs. However, in [23] only the backtracking path was deployed on the FPGA, and a submatrix structure is used to load the path chosen for alignment. Meanwhile, our architecture relies on a memory storage structure and the definition of the maximum score to align the sequences. The architecture proposed by [47] achieved the best performance, as can be seen in Table 3. However, it uses a register-file concept instead of systolic-array. Therefore, due to the similarity of hardware techniques used to deploy the SW algorithm, we discuss a result comparison with [23], which achieved the second-best performance.

Furthermore, a comparison with [23] was also carried out regarding the FPGA area occupation, and it is presented in Table 4. The second and third columns present the FPGA and the number of PEs used, respectively. Meanwhile, the third and fourth columns present the slices and memory blocks occupied, and the fifth column the operating frequency.

Table 4. Table with the summaries of the results of the FPGA synthesis works of SW implementation (hardware SW with backtracking step). The Slice column is related to the logical distribution and refers to the occupied slices in the synthesis.

Related Works	FPGA Model	Array Size (PE)	Slices /(%)	Memory /(%)	Frequency (MHz)
This work	XC7VX485T	512	35, 286/(46%)	0 BRAM/(0%)	155
2017 [23]	XC7VX485T	512	57, 870/(76%)	896 BRAM/(87%)	200

https://doi.org/10.1371/journal.pone.0254736.t004

As shown in Table 4, for the same number of PEs, our architecture occupied 35, 286 slices and 0 BRAMs in contrast to 57, 870 slices and 896 block RAMs (28 Mbits memory) in [23]. Also, the total area occupation was higher than 60%, compared to 46% on ours, due to the substitution matrix. Therefore, our proposal has high scalability due to the low resource usage (can reach up to 1, 024 PEs for the XC7VX485T). Besides, our implementation proposal can be implemented in smaller FPGAs, such as the Virtex-6 XC6VLX240T, with a reasonable nucleotide sequence.

Regarding the operation frequency, our proposal can reach up to 155 MHz. So, it is observed that the proposals with the best performances have a similar structure, even with different approaches to the solution. Our proposal and [23] achieving the same performance for the frequency of 150 MHz. The proposal with the highest frequency achieved was that of [47] reaching 500 MHz with Virtex-5 XC5VLX50T FPGA.

Therefore, our work uses fewer hardware resources to perform the alignment process due to the chosen backtracking approach. As the backtracking stage results in high computational complexity, we simplified the process using the path mapping through the maximum value in **D** and **H**, resulting in linear computational complexity. On the other hand, the architecture proposed by [23] uses considerably more memory resources due to data partitioning and prefetching for the backtracking step. Despite both works achieving similar performance due to the systolic array, there are significant differences in the alignment approach chosen for the FPGA implementation.

The SW proposed by [41] is—to our knowledge—the most recent work on sequence alignment with the SW algorithm that also embeds the backtracking process in custom hardware. Their design achieved similar performance to ours for 512 PEs, as shown in Table 3. However, our approach can reach up to 1024 PEs embedded in the Virtex-7 XC7VX485T, a lower clock frequency and, thus, double the performance in GCUPS. Despite that, the design proposed by [23] achieved the second best overall performance. Meantime, the architecture proposed in [47], using a register-file concept, achieved 128GCUPS, the best overall performance shown in Table 4.

The hardware implementation of the alignment process through our approach, developed based on a chain of directions and the maximum score address, is a key contribution for the low use of memories. Thus, as we did not carry out tests with real biological datasets, theoretically speaking, it is possible to achieve high hardware scalability. Besides, the sequence of any size can be aligned with our approach limited by the hardware resources available. In addition, the proposed method can compress the data, using only 3 bits in a fixed-point implementation.

5 Conclusion

This paper presented a parallel FPGA platform design to accelerate both the forward and backtracking stages of the SW algorithm. The main contributions were the high-speed data processing implementation and low memory usage that theoretically allows high scalability. The hardware resources available on the FPGA are a limiting factor to the size of the score matrix

but not to the size of sequences to be aligned. Therefore, satisfying the high-throughput, ultra-low-latency and low-power requirements and to alleviate the raw data processing problem in bioinformatics. From the strategy of storing alignment path distances and maximum score position during forward stage processing, it was possible to reduce the complexity of backtracking stage processing which allowed to follow the path directly. The proposal architecture achieved a satisfactory critical path, reduced memory usage and, theoretically, a high scalability for two-step SW algorithm. Synthesis results showed that the proposed method could support up to 1, 024 PEs in only one FPGA, using the Xilinx Virtex-7 XC7VX485T. The main advantage is the low hardware resource usage and high performance of 79.5 GCUPS, with an operating frequency of up to 155MHz, without using external resources.

Author Contributions

Conceptualization: Fabio F. de Oliveira, Marcelo A. C. Fernandes.

Data curation: Fabio F. de Oliveira, Leonardo A. Dias, Marcelo A. C. Fernandes.

Formal analysis: Fabio F. de Oliveira, Leonardo A. Dias, Marcelo A. C. Fernandes.

Funding acquisition: Marcelo A. C. Fernandes.

Investigation: Fabio F. de Oliveira, Marcelo A. C. Fernandes.

Methodology: Leonardo A. Dias, Marcelo A. C. Fernandes.

Project administration: Marcelo A. C. Fernandes.

Resources: Marcelo A. C. Fernandes.

Software: Fabio F. de Oliveira, Marcelo A. C. Fernandes.

Supervision: Marcelo A. C. Fernandes.

Validation: Marcelo A. C. Fernandes.

Visualization: Marcelo A. C. Fernandes.

Writing – original draft: Fabio F. de Oliveira, Leonardo A. Dias, Marcelo A. C. Fernandes.

Writing – review & editing: Fabio F. de Oliveira, Leonardo A. Dias, Marcelo A. C. Fernandes.

References

1. Masseroli M, Canakoglu A, Pinoli P, Kaitoua A, Gulino A, Horlova O, et al. Processing of big heterogeneous genomic datasets for tertiary analysis of Next Generation Sequencing data. Bioinformatics. 2018; 35(5):729–736. https://doi.org/10.1093/bioinformatics/bty688
2. Pereira R, Oliveira J, Sousa M. Bioinformatics and Computational Tools for Next-Generation Sequencing Analysis in Clinical Genetics. Journal of Clinical Medicine. 2020; 9(1). https://doi.org/10.3390/jcm9010132 PMID: 31947757
3. Schuster S. Next-generation sequencing transforms today's biology. Nature methods. 2008; 5:16–8. https://doi.org/10.1038/nmeth1156 PMID: 18165802
4. Kumar G, Kocour M. Applications of next-generation sequencing in fisheries research: A review. Fisheries Research. 2017; 186:11–22. https://doi.org/10.1016/j.fishres.2016.07.021
5. Tanjo T, Kawai Y, Tokunaga K, Ogasawara O, Nagasaki M. Practical guide for managing large-scale human genome data in research. Journal of Human Genetics. 2020; 66. https://doi.org/10.1038/s10038-020-00862-1 PMID: 33097812
6. Zhou P, Shi ZL. SARS-CoV-2 spillover events. Science. 2021; 371(6525):120–122. https://doi.org/10.1126/science.abf6097 PMID: 33414206
7. Lyng GD, Sheils NE, Kennedy CJ, Griffin DO, Berke EM. Identifying optimal COVID-19 testing strategies for schools and businesses: Balancing testing frequency, individual test technology, and cost. PLOS ONE. 2021; 16(3):1–13. https://doi.org/10.1371/journal.pone.0248783

8. Mazzarelli A, Giancola ML, Farina A, Marchioni L, Rueca M, Gruber CEM, et al. 16S rRNA gene sequencing of rectal swab in patients affected by COVID-19. PLOS ONE. 2021; 16(2):1–15. https://doi.org/10.1371/journal.pone.0247041 PMID: 33596245
9. Miller D, Martin MA, Harel N, Kustin T, Tirosh O, Meir M, et al. Full genome viral sequences inform patterns of SARS-CoV-2 spread into and within Israel. Nature Communications. 2020; 11:5518. https://doi.org/10.1038/s41467-020-19248-0 PMID: 33139704
10. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. Journal of Molecular Biology. 1990; 215(3):403–410. https://doi.org/10.1016/S0022-2836(05)80360-2 PMID: 2231712
11. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology. 1970; 48(3):443–453. https://doi.org/10.1016/0022-2836(70)90057-4 PMID: 5420325
12. Smith TF, Waterman MS. Identification of common molecular subsequences. Journal of Molecular Biology. 1981; 147(1):195–197. https://doi.org/10.1016/0022-2836(81)90087-5 PMID: 7265238
13. Afifi S, Gholamhosseini H, Sinha R. Hardware Implementations of SVM on FPGA: AState-of-the-Art Review of Current Practice. International Journal of Innovative Science, Engineering & Technology (IJISET). 2015; 2:733–752.
14. Aijaz A, Dohler M, Aghvami AH, Friderikos V, Frodigh M. Realizing the Tactile Internet: Haptic Communications over Next Generation 5G Cellular Networks. IEEE Wireless Communications. 2017; 24 (2):82–89. https://doi.org/10.1109/MWC.2016.1500157RP
15. Houtgast EJ, Sima VM, Bertels K, Al-Ars Z. Hardware acceleration of BWA-MEM genomic short read mapping for longer read lengths. Computational Biology and Chemistry. 2018; 75:54–64. https://doi.org/10.1016/j.compbiolchem.2018.03.024 PMID: 29747076
16. Courneya JP, Mayo A. High-performance computing service for bioinformatics and data science. Journal of the Medical Library Association: JMLA. 2018; 106:494–495. https://doi.org/10.5195/jmla.2018.512 PMID: 30271293
17. Arenas M, Mora A, Romero G, Castillo P. GPU Computation in Bioinformatics. A review. Advances in Intelligent Modelling and Simulation. 2012; p. 433–440.
18. Khan D, Shedole S. Accelerated Deep Learning in Proteomics—A Review. Innovation in Electrical Power Engineering, Communication, and Computing Technology. 2020; p. 291–300. https://doi.org/10.1007/978-981-15-2305-2_23
19. González-Domínguez J, Ramos S, Touriño J, Schmidt B. Parallel pairwise epistasis detection on heterogeneous computing architectures. IEEE Transactions on Parallel and Distributed Systems. 2015; 27 (8):2329–2340.
20. Letras M, Bustio-Martínez L, Cumplido R, Hernández-León R, Feregrino-Uribe C. On the design of hardware architectures for parallel frequent itemsets mining. Expert Systems with Applications. 2020; 157:113440. https://doi.org/10.1016/j.eswa.2020.113440
21. Juvonen MPT, Coutinho JGF, Wang JL, Lo BL, Luk W, Mencer O, et al. Custom hardware architectures for posture analysis. In: Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.; 2005. p. 77–84.
22. Kaplan R, Yavits L, Ginosar R, Weiser U. A Resistive CAM Processing-in-Storage Architecture for DNA Sequence Alignment. IEEE Micro. 2017; 37(4):20–28. https://doi.org/10.1109/MM.2017.3211121
23. Fei X, Dan Z, Lina L, Xin M, Chunlei Z. FPGASW: Accelerating Large-Scale Smith–Waterman Sequence Alignment Application with Backtracking on FPGA Linear Systolic Array. Interdisciplinary Sciences: Computational Life Sciences. 2017; 10. PMID: 28432608
24. Cadenelli N, Jaksic Z, Polo J, Carrera D. Considerations in using OpenCL on GPUs and FPGAs for throughput-oriented genomics workloads. Future Generation Computer Systems. 2019; 94:148–159. https://doi.org/10.1016/j.future.2018.11.028
25. Franke K, Crowgey E. Accelerating next generation sequencing data analysis: an evaluation of optimized best practices for Genome Analysis Toolkit algorithms. Genomics & Informatics. 2020; 18:e10. https://doi.org/10.5808/GI.2020.18.1.e10 PMID: 32224843
26. Nobile MS, Cazzaniga P, Tangherloni A, Besozzi D. Graphics processing units in bioinformatics, computational biology and systems biology. Briefings in Bioinformatics. 2016; 18(5):870–885. https://doi.org/10.1093/bib/bbw058
27. Manconi A, Moscatelli M, Gnocchi M, Armano G, Milanesi L. A GPU-based high performance computing infrastructure for specialized NGS analyses. In: PeerJ Preprints; 2016. p. 3.
28. Kung. Why systolic architectures? Computer. 1982; 15(1):37–46. https://doi.org/10.1109/MC.1982.1653825

29. Kung HT, McDanel B, Zhang SQ. Packing Sparse Convolutional Neural Networks for Efficient Systolic Array Implementations: Column Combining Under Joint Optimization. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. ASPLOS'19. New York, NY, USA: Association for Computing Machinery; 2019. p. 821–834. Available from: https://doi.org/10.1145/3297858.3304028.

30. Sze V. Designing Hardware for Machine Learning: The Important Role Played by Circuit Designers. IEEE Solid-State Circuits Magazine. 2017; 9(4):46–54. https://doi.org/10.1109/MSSC.2017.2745798

31. Dias LA, Ferreira JC, Fernandes MAC. Parallel Implementation of K-Means Algorithm on FPGA. IEEE Access. 2020; 8:41071–41084. https://doi.org/10.1109/ACCESS.2020.2976900

32. Dias LA, Damasceno AM, Gaura E, Fernandes MA. A full-parallel implementation of Self-Organizing Maps on hardware. Neural Networks. 2021;. https://doi.org/10.1016/j.neunet.2021.05.021 PMID: 34112575

33. Barros WK, Dias LA, Fernandes MA. Fully Parallel Implementation of Otsu Automatic Image Thresholding Algorithm on FPGA. Sensors. 2021; 21(12):4151. https://doi.org/10.3390/s21124151 PMID: 34204291

34. Hughey R, Lopresti DP. Architecture of a programmable systolic array. In: [1988] Proceedings. International Conference on Systolic Arrays; 1988. p. 41–49.

35. He D, He J, Liu J, Yang J, Yan Q, Yang Y. An FPGA-Based LSTM Acceleration Engine for Deep Learning Frameworks. Electronics. 2021; 10(6). https://doi.org/10.3390/electronics10060681

36. Zhang H, Fu Y, Feng L, Zhang Y, Hua R. Implementation of Hybrid Alignment Algorithm for Protein Database Search on the SW26010 Many-Core Processor. IEEE Access. 2019; 7:128054–128063. https://doi.org/10.1109/ACCESS.2019.2940044

37. Rognes T. Faster Smith-Waterman database searches by inter-sequence SIMD parallelisation. BMC bioinformatics. 2011; 12:221. https://doi.org/10.1186/1471-2105-12-221 PMID: 21631914

38. Liu Y, Maskell D, Schmidt B. CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. BMC research notes. 2009; 2:73. https://doi.org/10.1186/1756-0500-2-73 PMID: 19416548

39. Court T, Herbordt M. Families of FPGA-Based Accelerators for Approximate String Matching. Microprocessors and microsystems. 2007; 31:135–145. https://doi.org/10.1016/j.micpro.2006.04.001 PMID: 21603598

40. Rucci E, Garcia C, Botella G, Giusti AED, Naiouf M, Prieto-Matias M. OSWALD: OpenCL Smith–Waterman on Altera's FPGA for Large Protein Databases. The International Journal of High Performance Computing Applications. 2018; 32(3):337–350. https://doi.org/10.1177/1094342016654215

41. Wu JP, Lin YC, Wu YW, Hsieh SW, Tai CH, Lu YC. A Memory-Efficient Accelerator for DNA Sequence Alignment with Two-Piece Affine Gap Tracebacks. In: 2021 IEEE International Symposium on Circuits and Systems (ISCAS); 2021. p. 1–4.

42. Banerjee SS, El-Hadedy M, Lim JB, Kalbarczyk ZT, Chen D, Lumetta SS, et al. ASAP: Accelerated Short-Read Alignment on Programmable Hardware. IEEE Transactions on Computers. 2019; 68 (3):331–346. https://doi.org/10.1109/TC.2018.2875733

43. Saavedra A, Lehnert H, Hernández C, Carvajal G, Figueroa M. Mining Discriminative K-Mers in DNA Sequences Using Sketches and Hardware Acceleration. IEEE Access. 2020; 8:114715–114732. https://doi.org/10.1109/ACCESS.2020.3003918

44. Lloyd S, Snell QO. Sequence Alignment with Traceback on Reconfigurable Hardware. In: 2008 International Conference on Reconfigurable Computing and FPGAs; 2008. p. 259–264.

45. Alser M, Hassan H, Kumar A, Mutlu O, Alkan C. Shouji: A fast and efficient pre-alignment filter for sequence alignment. Bioinformatics (Oxford, England). 2019; 35:4255–4263. https://doi.org/10.1093/bioinformatics/btz234 PMID: 30923804

46. Rucci E, Garcia C, Botella G, De Giusti A, Naiouf M, Prieto Matias M. SWIFOLD: Smith-Waterman implementation on FPGA with OpenCL for long DNA sequences. BMC Systems Biology. 2018; 12. https://doi.org/10.1186/s12918-018-0614-6 PMID: 30458766

47. Sarkar A, Banerjee S, Ghosh S. An Energy-Efficient Pipelined-Multiprocessor Architecture for Biological Sequence Alignment. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2020; 28 (12):2598–2611. https://doi.org/10.1109/TVLSI.2020.3015138

48. Nurdin D, md isa mn, Ismail RC, Ahmad MI. High Performance Systolic Array Core Architecture Design for DNA Sequencer. MATEC Web of Conferences. 2018; 150:06009. https://doi.org/10.1051/matecconf/201815006009

49. Arram J, Luk W, Jiang P. Reconfigurable filtered acceleration of short read alignment. In: 2013 International Conference on Field-Programmable Technology (FPT); 2013. p. 438–441.

50. Ng HC, Liu S, Coleman I, Chu RSW, Yue MC, Luk W. Acceleration of Short Read Alignment with Runtime Reconfiguration. In: 2020 International Conference on Field-Programmable Technology (ICFPT); 2020. p. 256–262.

51. Seliem AG, Hamed HFA, Abouelwafa W. MapReduce Model Using FPGA Acceleration for Chromosome Y Sequence Mapping. IEEE Access. 2021; 9:83402–83409. https://doi.org/10.1109/ACCESS.2021.3085997

52. Koliogeorgi K, Voss N, Fytraki S, Xydis S, Gaydadjiev G, Soudris D. Dataflow Acceleration of Smith-Waterman with Traceback for High Throughput Next Generation Sequencing. In: 2019 29th International Conference on Field Programmable Logic and Applications (FPL); 2019. p. 74–80.

53. Chen YL, Chang BY, Yang CH, Chiueh TD. A High-Throughput FPGA Accelerator for Short-Read Mapping of the Whole Human Genome. IEEE Transactions on Parallel and Distributed Systems. 2021; 32 (6):1465–1478. https://doi.org/10.1109/TPDS.2021.3051011

54. Siddiqui F, Amiri S, Minhas UI, Deng T, Woods R, Rafferty K, et al. FPGA-Based Processor Acceleration for Image Processing Applications. Journal of Imaging. 2019; 5(1). https://doi.org/10.3390/jimaging5010016 PMID: 34465705

55. Pilz S, Porrmann F, Kaiser M, Hagemeyer J, Hogan JM, Rückert U. Accelerating Binary String Comparisons with a Scalable, Streaming-Based System Architecture Based on FPGAs. Algorithms. 2020; 13 (2). https://doi.org/10.3390/a13020047

56. Rashed AEED, Obaya M, Moustafa HED. Accelerating DNA pairwise sequence alignment using FPGA and a customized convolutional neural network. Computers & Electrical Engineering. 2021; 92:107112. https://doi.org/10.1016/j.compeleceng.2021.107112

57. Benkrid K, Liu Y, Benkrid A. A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2009; 17(4):561–570. https://doi.org/10.1109/TVLSI.2008.2005314

58. Isa MN, Benkrid K, Clayton T, Ling C, Erdogan AT. An FPGA-based parameterised and scalable optimal solutions for pairwise biological sequence analysis. In: 2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS); 2011. p. 344–351.

59. Sebastiao N, Roma N, Flores P. Integrated Hardware Architecture for Efficient Computation of the *n*-Best Bio-Sequence Local Alignments in Embedded Platforms. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2012; 20(7):1262–1275. https://doi.org/10.1109/TVLSI.2011.2157541

60. Xilinx. System Generator for DSP; 2008, Accessed on Jan 30, 2020. Available from: https://www.xilinx.com/.

61. Vasco P. Smith-Waterman-Algorithm; 2019, Accessed on June 04, 2021. Available from: https://github.com/pedrovasco96/Smith-Waterman-Algorithm/.

62. Oliveira F, Fernandes M. Smith-Waterman-Algorithm Demo; 2021, Accessed on June 22, 2021. Available from: https://drive.google.com/drive/folders/1Mr78U1MNA6HvKV1fWA248Zp05LCGdJN0?usp=sharing.

63. Oliveira F, Fernandes M. Smith-Waterman-Algorithm-on-FPGA; 2021, Accessed on December 02, 2021. Available from: https://github.com/Veritate/Smith-Waterman-Algorithm-on-FPGA.

64. Oliver T, Schmidt B, Maskell D. Hyper customized processors for bio-sequence database scanning on FPGAs; 2005. p. 229–237.

65. Zhang P, Tan G, Gao G. Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform; 2007. p. 39–48.

66. Storaasli O, Yu W, Strenski D, Maltby J. Performance Evaluation of FPGA-Based Biological Applications. Seattle; 2007.

67. Alachiotis N, Berger SA, Stamatakis A. Accelerating Phylogeny-Aware Short DNA Read Alignment with FPGAs. In: 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines; 2011. p. 226–233.

68. Olson CB, Kim M, Clauson C, Kogon B, Ebeling C, Hauck S, et al. Hardware Acceleration of Short Read Mapping. In: 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines; 2012. p. 161–168.

69. Preuber TB, Knodel O, Spallek RG. Short-Read Mapping by a Systolic Custom FPGA Computation. In: 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines; 2012. p. 169–176.

70. Tang W, Wang W, Duan B, Zhang C, Tan G, Zhang P, et al. Accelerating Millions of Short Reads Mapping on a Heterogeneous Architecture with FPGA Accelerator. In: 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines; 2012. p. 184–187.

71. Chen P, Wang C, Li X, Zhou X. Accelerating the Next Generation Long Read Mapping with the FPGA-Based System. IEEE/ACM Transactions on Computational Biology and Bioinformatics. 2014; 11 (5):840–852. https://doi.org/10.1109/TCBB.2014.2326876 PMID: 26356857