Needleman-Wunsch and Smith-Waterman Hardware-Oriented Algorithms using Arithmetic mod-2^N

Journal:	*IEEE Transactions on Circuits and Systems I: Regular Papers*
Manuscript ID	TCAS-I-00463-2019
Manuscript Type:	Regular Paper
Date Submitted by the Author:	28-Apr-2019
Complete List of Authors:	Corrales-Mendoza, Duverney; Universidad del Valle Velasco-Medina, Jaime; Universidad del Valle, School of Electrical and Electronics Engineering
EDICS:	DCS100B0 - Arithmetic circuits and systems < DCS100 - Arithmetic circuits and systems < Digital Circuits and Systems (and VLSI), DCS130 - Digital circuits techniques and building blocks < Digital Circuits and Systems (and VLSI), DCS140 - Embedded digital systems < Digital Circuits and Systems (and VLSI), DCS190 - Reconfigurable and field programmable digital circuits < Digital Circuits and Systems (and VLSI), DCS190A0 - FPGAs < DCS190 - Reconfigurable and field programmable digital circuits < Digital Circuits and Systems (and VLSI), SIPRO110D5 - Algorithms and architectures for digital signal processing < SIPRO110 - Digital signal processing < SIPRO - Signal Processing

Needleman-Wunsch and Smith-Waterman Hardware-Oriented Algorithms using Arithmetic mod-2^N

Duverney Corrales-Mendoza, *Member, IEEE*, Jaime Velasco-Medina, *Senior Member, IEEE*

Abstract— Pairwise Sequence Alignment (PSA) algorithms based on Dynamic Programming (DP) technique use matrices whose score values increase according to the biological sequence length, requiring a lot of bits for processing very large sequences. Then, the above issue is critical for hardware-based implementations of PSA algorithms. In this paper, we propose hardware-oriented algorithms for performing the Needleman-Wunsch Algorithm (NWA) or Smith-Waterman Algorithm (SWA) using arithmetic mod-2^N, with linear gap or affine gap penalties, and equations for calculating the maximum difference D between the processed data for the above algorithms, in order to determine the value of the modulus 2^N according to the selected score matrix and gap penalties. From the proposed algorithms, we design three Processing Elements (PEs), which can be used to design 1D/2D array hardware architectures for performing the alignment of a base-pair sequence of any length.

Index Terms— Needleman-Wunsch, Smith-Waterman, FPGA, Pairwise Sequence Alignment, Modular Arithmetic, Dynamic Programming, Hardware-Oriented Algorithm, Hardware Design.

I. Introduction

DP technique is used to simplify complex computational problems by breaking them down into simpler sub-problems, which are processed in a recursive manner in order to find an optimal solution. DP-based algorithms are widely used in Bioinformatics for performing tasks such as sequence alignment [1], protein folding [2], RNA structure prediction [3] and protein-DNA binding [4]. Also, they are used in other contexts as matching learning [5], control of non-linear systems [6], artificial vision [7], wireless communication [8], among others.

In Bioinformatics context, PSA is used to find functional, structural and evolutionary relationships between a biological base-pair sequence (DNA, RNA or proteins), and DP-based PSA algorithms are used to perform optimal PSA. The two main DP-based PSA algorithms are NWA for global PSA [9] and SWA for local PSA [10]. However, those algorithms are computationally demanding because they have a computational complexity $O(n^2)$, and the scoring values of their matrices increase according to the base-pair sequence length. Then, the PSA of a very large base-pair (bp) sequence using hardware accelerators requires integer arithmetic units of many bits. For example, the integer arithmetic units for performing proteins PSA of 128bp (2^7) require 13-bit for representing $\pm 128 * 17$, where value 17 (5-bit) is the maximum value of the score matrix *PAM250*, and 1-Mbp (2^{20}) requires 26-bit.

In order to address the above issue, some hardware designs based on specific codifications or modular arithmetic have been proposed. In [11], the authors present a systolic array architecture for DNA local-PSA using PEs of 1-bit, reaching a high PE density (4032-PEs); that design is restricted to a specific score scheme $\{match, mismatch, gap\} = \{0, 2, 1\}$. However, it is important to mention that the scoring scheme is selected according to the statistical properties of the biological base-pair sequence, and different PSAs can be obtained for each score scheme. Therefore, the above approach is not a general solution. In [12], the authors present PE designs of 32-bit for the score scheme $\{match, mismatch, gap\} = \{2, -1, -1\}$, where some PE signals are processed using 2-bit. In [13], the authors use the Residue Number System (RNS) for processing score values less than the value M, where $M = 1680$ is the product of all m_i moduli set $\{2^N, 2^{N-1}, 2^{N-1} - 1\}$ with $N = 4$. Then, the above works do not solve the hardware design issues for performing NWA and SWA of a very large base-pair sequence with a configurable score scheme.

Therefore, taking into account the hardware design issues, it is necessary to design hardware-oriented algorithms for performing NWA and SWA, for example, algorithms based on modular arithmetic. In that case, it is necessary to perform the $A<B$ modular comparison, which can be used unambiguously when the maximum difference D between the two processed data is fixed, that is, $D = |A - B|$ [14]. Then, the $A<B$ modular comparison can be used to develop hardware-oriented algorithms for performing NWA and SWA due to the recursive property of the DP technique. In this context, the arithmetic

This paragraph of the first footnote will contain the date on which you submitted your paper for review. It will also contain support information, including sponsor and financial support acknowledgment. For example, "This work was supported in part by the U.S. Department of Commerce under Grant BS123456."

This work was supported by Universidad del Valle.

The authors are with the School of Electrical and Electronic Engineering, Universidad del Valle, Cali 760032, Colombia. (e-mail: duverney.corrales @correounivalle.edu.co; jaime.velasco@correounivalle.edu.co)

mod-2^N is adequate for hardware implementation of PSA algorithms, since $A<B$ modular comparison can be performed using two's complement adders of N-bit, where $A<B$ if *MSB[A-B]='1'* for the case where $|A-B| < 2^{N-1}$ [15].

To the best of our knowledge, this is the first time that practical algorithms for efficient hardware implementations of NWA and SWA using arithmetic mod-2^N are proposed. In this case, we also propose the equations to calculate the maximum difference *D* according to the selected score scheme with linear gap or affine gap penalties. The proposed algorithms solve the above-mentioned hardware design issues, allowing the alignment of a base-pair sequence of any length, disregard the selected score scheme.

The rest of this paper is organized as follows: In Section II are presented the issues of NWA and SWA for their efficient hardware implementations using arithmetic mod-2^N. In Section III are described the proposed hardware-oriented algorithms for performing NWA and SWA, and equations for calculating the maximum difference *D* between the processed data for the above algorithms. In Section IV are presented the designs of three PEs for performing NWA from proposed hardware-oriented algorithms. In section V are presented the design of PEs for performing SWA using the PEs for NWA. In Section VI are discussed the synthesis and performance results of the designed PEs for NWA, and conclusions are presented in Section VII.

II. Issues for Hardware Implementation of NWA and SWA Using Modular Arithmetic

In this section, we present the issues for the straightforward hardware implementations, using modular arithmetic, of NWA and SWA with linear gap or affine gap penalties.

PSA is achieved by performing two processes: forward process (FWP) and traceback process (TBP). NWA and SWA are used to perform the forward process (FWP) of global and local PSA, respectively, considering a scoring scheme with a score matrix $s(S1_i, S2_j)$ (e.g. *match/mismatch*, *blosum62*, *PAM250*) and gap penalties (e.g. linear gap, affine gap). The score matrix *match/mismatch* is generally used to align DNA sequences, where the value "*match*" is assigned when $S1_i = S2_j$ and the value "*mismatch*" when $S1_i \neq S2_j$. Linear gap penalty considers the same biological cost (g) for all gaps, and affine gap penalty considers different biological costs for open (d) and extended (e) gap penalties in order to improve the gap distribution [16].

The algorithms of NWA with linear gap penalty [9], NWA with affine gap penalty of three matrices [16] and NWA with affine gap penalty of two matrices [17], are presented in Algorithms 1, 2 and 3, respectively, and SWA with linear gap penalty [10] is presented in Algorithm 4.

Algorithm 1. NWA with linear gap penalty

Input: Base-pair-sequences $S1=\{S1_1 \ldots S1_n\}$ and $S2=\{S2_1 \ldots S2_m\}$
Output: Matrices H and A_H

//Initialization
1. $H_{(0,0)} = 0$
2. **for** $i = 1{:}n$ **do** $H_{(i,0)} = H_{(i-1,0)} + g$
3. **for** $j = 1{:}m$ **do** $H_{(0,j)} = H_{(0,j-1)} + g$

//Calculate similarity matrix H and arrow matrix A_H
for $i = 1{:}n$ **do**
  for $j = 1{:}m$ **do**
4. $\{H, A_H\}_{(i,j)} = \boldsymbol{max}\left\{\begin{bmatrix} H_{(i-1,j-1)} + s(S1_i, S2_j) \\ H_{(i-1,j)} + g \\ H_{(i,j-1)} + g \end{bmatrix}, \begin{bmatrix} diag(\searrow) \\ up(\downarrow) \\ left(\rightarrow) \end{bmatrix}\right\}$
  end for
end for
return H and A_H

Algorithm 2. NWA with affine gap penalty of three matrices

Input: Base-pair-sequences $S1=\{S1_1 \ldots S1_n\}$ and $S2=\{S2_1 \ldots S2_m\}$
Output: Matrices H and A_H

//Initialization
1. $H_{(0,0)} = 0$
2. $H_{(0,1)} = H_{(1,0)} = d$
3. **for** $i = 2{:}n$ **do** $H_{(i,0)} = H_{(i-1,0)} + e$
4. **for** $j = 2{:}m$ **do** $H_{(0,j)} = H_{(0,j-1)} + e$
5. $G_{(0,0)} = -\infty$
6. **for** $i = 1{:}n$ **do** $Gy_{(i,0)} = Gy_{(i-1,0)} + e = -\infty$
7. **for** $j = 1{:}m$ **do** $Gx_{(0,j)} = Gx_{(0,j-1)} + e = -\infty$

//Calculate value matrices H, G_X and G_Y, and arrow matrix A_H
for $i = 1{:}n$ **do**
  for $j = 1{:}m$ **do**
8. $\{H, A_H\}_{(i,j)} = \mathbf{max}\left\{\begin{bmatrix} H_{(i-1,j-1)} + s(S1_i, S2_j) \\ Gx_{(i-1,j-1)} + s(S1_i, S2_j) \\ Gy_{(i-1,j-1)} + s(S1_i, S2_j) \end{bmatrix}, \begin{bmatrix} diag(\searrow) \\ up(\downarrow) \\ left(\rightarrow) \end{bmatrix}\right\}$
9. $Gx_{(i,j)} = \boldsymbol{max}\left\{\begin{bmatrix} H_{(i-1,j)} + d \\ Gx_{(i-1,j)} + e \end{bmatrix}\right\}$
10. $Gy_{(i,j)} = \boldsymbol{max}\left\{\begin{bmatrix} H_{(i,j-1)} + d \\ Gx_{(i,j-1)} + e \end{bmatrix}\right\}$
  end for
end for
return H and A_H

Algorithm 3. NWA with affine gap of two matrices

Input: Base-pair-sequences $S1=\{S1_1 \ldots S1_n\}$ and $S2=\{S2_1 \ldots S2_m\}$
Output: Matrices H and A_H

//Initialization
1. $H_{(0,0)} = 0$
2. $H_{(0,1)} = H_{(1,0)} = d$
3. **for** $i = 2{:}n$ **do** $H_{(i,0)} = H_{(i-1,0)} + e$
4. **for** $j = 2{:}m$ **do** $H_{(0,j)} = H_{(0,j-1)} + e$
5. $G_{(0,0)} = -\infty$
6. **for** $i = 1{:}n$ **do** $G_{(i,0)} = G_{(i-1,0)} + e = -\infty$
7. **for** $j = 1{:}m$ **do** $G_{(0,j)} = G_{(0,j-1)} + e = -\infty$

//Calculate value matrices H and G, and arrow matrices A_H and A_G
for $i = 1{:}n$ **do**
  for $j = 1{:}m$ **do**
8. $\{H, A_H\}_{(i,j)} = s(S1_i, S2_j) + \boldsymbol{max}\left\{\begin{bmatrix} H_{(i-1,j-1)} \\ G_{(i-1,j-1)} \end{bmatrix}, \begin{bmatrix} diag(\searrow) \\ up|left(\downarrow\rightarrow) \end{bmatrix}\right\}$
9. $\{G, A_G\}_{(i,j)} = \boldsymbol{max}\left\{\begin{bmatrix} H_{(i-1,j)} + d \\ G_{(i-1,j)} + e \\ H_{(i,j-1)} + d \\ G_{(i,j-1)} + e \end{bmatrix}, \begin{bmatrix} up(\downarrow) \\ up(\downarrow) \\ left(\rightarrow) \\ left(\rightarrow) \end{bmatrix}\right\}$
  end for
end for
return H and A_H

Algorithm 4. SWA with linear gap penalty

Input: Base-pair-sequences $S1=\{S1_1 \dots S1_n\}$ and $S2=\{S2_1 \dots S2_m\}$
Output: Matrices H and A_H, and coordinates $[I,J]$ *of the cell of the matrix* H *with MLSV*

//Initialization
1. $H_{(0,0)} = 0$
2. ***for*** $i = 1{:}n$ ***do*** $H_{(i,0)} = 0$
3. ***for*** $j = 1{:}m$ ***do*** $H_{(0,j)} = 0$

//Calculate similarity matrix H, arrow matrix A_H *and coordinates [I,J]*
for $i = 1{:}m$ ***do***
  for $j = 1{:}m$ ***do***
4. $\{H, A_H\}_{(i,j)} = \boldsymbol{max}\left\{\begin{bmatrix} \mathbf{0} \\ H_{(i-1,j-1)} + s(S1_i, S2_j) \\ H_{(i-1,j)} + g \\ H_{(i,j-1)} + g \end{bmatrix}, \begin{bmatrix} null \\ diag(\searrow) \\ up(\downarrow) \\ left(\rightarrow) \end{bmatrix}\right\}$

//Seek coordinates [i,j] of the cell of the matrix H with MLSV
if $H_{(i,j)} < MLSV$
5. ***then*** $\{\text{MLSV}, [\text{I}, \text{J}]\} = \{\text{MLSV}, [\text{I}, \text{J}]\}$
6. ***else*** $\{\text{MLSV}, [\text{I}, \text{J}]\} = \{H_{(i,j)}, [i,j]\}$
end if
end for
end for
return H, A_H *and* $[I,J]$

In FWP, NWA and SWA calculate progressively the similarity matrix H, the gap matrices G_X, G_Y and G, and the arrow matrices A_H and A_G. NWA and SWA with linear gap penalty use only the matrices H and A_H, while NWA and SWA with affine gap penalty use three matrices $\{H, G_X, G_Y\}$, and the matrix A_H. However, the matrices G_X and G_Y can be reduced to one matrix G, obtaining two matrices $\{H, G\}$ and two matrices $\{A_H, A_G\}$ for the case when $mismatch \leq 2e$ [17]. Matrices A_H and A_G indicate the source path associated to the maximum selected value of each cell of the matrices H and G, where A_G is used to calculate A_H, and the arrows of A_H are used to perform the traceback process (TBP)

In TBP, optimal PSA is determined by the path indicated by the arrows, where each arrow indicates the location of the next arrow. In the global PSA case, the PSA starts in the last matrix cell $A_{H(n,m)}$, and it stops in the first matrix cell $A_{H(0,0)}$, whereas the local PSA starts in the matrix cell $A_{H(I,J)}$, where $[I,J]$ are the coordinates of the matrix H cell with the *Maximum-local score value* (*MLSV*), and it stops when a matrix cell $H_{(i,j)}$ has the value 0.

The global PSA of the sequences *S1* and *S2* using NWA with linear gap penalty (See Algorithm 1) is shown in Figure 1, which was obtained from the software tool developed by [18] and using the score scheme $\{match, mismatch, gap\} = \{3, -1, -2\}$.

In this case, the score values and arrows of the matrices H and A_H are calculated in FWP using four steps:

1) Initialize the matrix cell $H_{(0,0)}$ with the value zero.
2) Initialize the matrix cells $H_{(i,0)}$ with the values $H_{(i-1,0)} + g$ for $1 \leq i \leq n$.
3) Initialize the matrix cells $H_{(0,j)}$ with the values $H_{(0,j-1)} + g$ for $1 \leq j \leq m$.
4) Calculate the matrix cells $H_{(i,j)}$ and $A_{H(i,j)}$ for $1 \leq i \leq n$ and $1 \leq j \leq m$ using the function *max*, assigning to each cell $H_{(i,j)}$ the maximum value between $\{H_{(i-1,j-1)} + s(S1_i, S2_j), H_{(i-1,j)} + g, H_{(i,j-1)} + g\}$, and assigning to each cell $A_{H(i,j)}$ an arrow for indicating the source path of the selected value in the cell $H_{(i,j)}$.

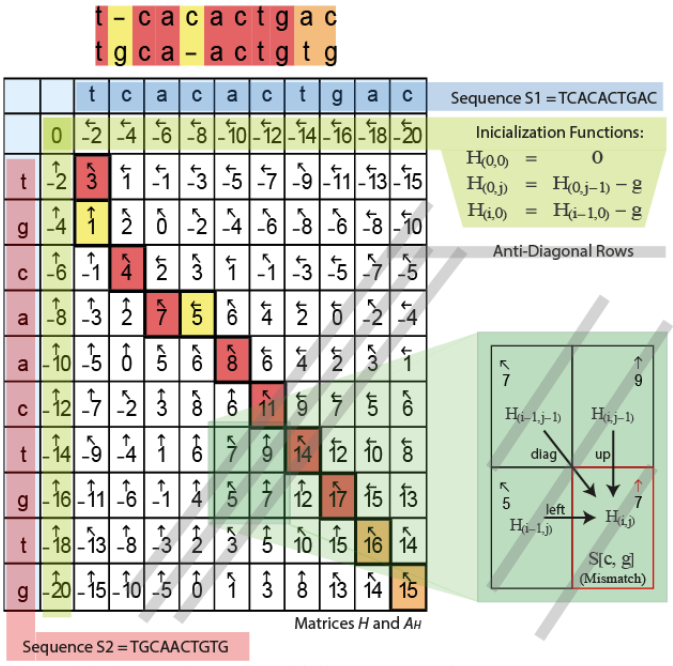

Fig 1. Global PSA using NWA with linear gap penalty.

Optimal global PSA is performed in TBP from matrix A_H using three steps:

1) Start PSA in the arrow of last matrix cell $A_{H(n,m)}$.
2) Find the optimal path by following the path indicated by each arrow, and simultaneously build the optimal PSA adding to it the base-pair symbols $\{S1_i, S2_j\}$ when $A_{H(i,j)} = \searrow$, $\{S1_i, \_\}$ when $A_{H(i,j)} = \rightarrow$ and $\{\_, S2_j\}$ when $A_{H(i,j)} = \downarrow$.
3) Stop PSA when the matrix cell $A_{H(0,0)}$ is reached.

In Figure 1 also can be seen that the cells of the matrices H and A_H of the same antidiagonal row can be processed in parallel way taking into account only the score values of the last two antidiagonal rows. Therefore, the matrices H and A_H can be processed in parallel by using optimized algorithms and high-throughput hardware architectures based on wave-front symmetries.

The calculated score values of the matrices H, G, G_X and G_Y increase according to the base-pair sequence length, that is, the calculation of the score values require of arithmetic units of a lot of bits for performing NWA or SWA using integer arithmetic. However, the maximum difference $D=|A-B|$ between the score values of the neighbor cells is small, except the value $-\infty$. Then, this small difference allows calculating the $A<B$ comparisons of the function *max* in modular space which enable to perform NWA and SWA in modular space.

Figure 2 shows the comparison $A<B$ *mod-*2^N of the integer values $a = 3228$ and $b = 3234$, which are represented in modular space *mod-*2^5 as $A = 28 = mod\text{-}2^5[a]$ and $B = 2 = mod\text{-}2^5[b]$, where $N = 5$ because it is considered the assumption $D = |a - b| < 2^{N-1}$. In this case, the differences

in integer space $(a - b = -6)$ and mod-2^5 space $(A - B = -6)$ are the same when $N \geq \lceil \log_2 D \rceil + 1$, that is, the Equation 1 is validated when $2^N > 2D$.

$$|a - b| = |mod2^N[a] - mod2^N[b]| = |A - B| \quad (1)$$

Therefore, the comparison $A<B$ *mod-*2^N can be performed using a N-bit adder, where the modular comparison $A<B$ *mod-*2^N=$MSB[A - B]$. In this case, $MSB[28 - 2] = MSB[\text{“}11100\text{”} - \text{“}00010\text{”}] = MSB[\text{“}11010\text{”}] = 1$, then $A<B$ *mod-*$2^N = \text{‘}1\text{’}$.

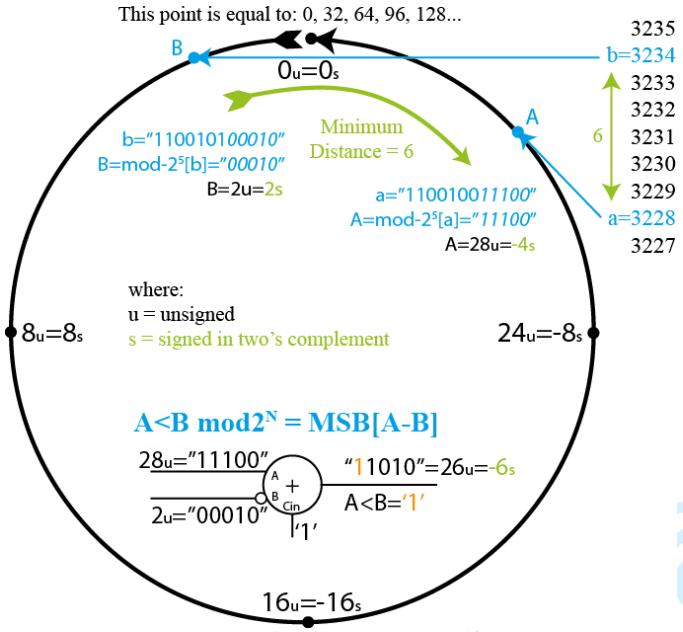

Fig 2. Comparison $A<B$ *mod-*2^5, considering $|a\text{-}b|<2^{N-1}=16$.

It is important to mention that SWA and NWA with affine gap penalty cannot be directly implemented in hardware using modular arithmetic because some values of the matrices H, G, G_X, and G_Y make the implementation of the $A<B$ modular comparison in hardware difficult. Then, the following values must be considered to develop hardware-oriented algorithms for performing NWA and SWA using arithmetic mod-2^N:

1) Value *Zero* of the comparison function *max* of SWA (See Algorithm 4) prevents the dissimilarities at the begin of the alignment. However, the *Zero* is difficult to process into modular space because it is not possible to distinguish this value from other $mod2^N(k * 2^N)$ values (e.g. 16,32,48… for $N = 4$). Then, it must be processed by a post-threshold function.
2) Value $-\infty$ of NWA (or SWA) with affine gap penalty (See Algorithms 2 and 3) is used to exclude from the comparison function *max* the values of the cells $G_{(i,0)}, G_{(0,j)}, Gx_{(0,j)}$ and $Gy_{(i,0)}$ (initialization functions of the gap matrices) for the cases $G_{(1,j)}, G_{(i,1)}, Gx_{(1,j)}$ and $Gy_{(i,1)}$, selecting always the values of the cells $H_{(i,0)}$ and $H_{(0,j)}$ (initialization functions of the matrix H). However, the value $-\infty$ cannot be represented in modular arithmetic. Then, the value $-\infty$ must be replaced by anyone score value less or equal than the corresponding value of the initialization function of the matrix H.
3) *Maximum-local score value (MLSV)* of the matrix H of SWA (See Algorithm 4) is used to seek the coordinates $[I, J]$ of the position of the cell with *MLSV* which is the start-cell of TBP for performing local PSA, but *MLSV* is difficult to seek into the modular space. However, the calculation of *MLSV* can be omitted because the coordinates $[I, J]$ of the start-cell of TBP can be obtained by using simple methods such as *dot-matrix*, *k-mers*, among others.

III. Hardware-Oriented Algorithms for NWA and SWA

In this section, we propose hardware-oriented algorithms for performing NWA and SWA using arithmetic mod-2^N, and the equations for calculating the maximum difference D between all the processed data by the function *max_mod2*^{N} from the selected score scheme for NWA and SWA with linear gap or affine gap penalties.

Equations 2 and 3 are proposed to calculate the maximum differences D_{Linear} and D_{Affine}, which are deduced from the score matrix and linear gap or affine gap penalties (Score scheme), respectively. The value D determinates the minimum value of the modulus 2^N, where N is the minimum number of bits that processes the arithmetic units of the PEs using arithmetic mod-2^N and its value is calculated by Equation 4.

$$D_{Linear} = Match + max\begin{Bmatrix} 2|g| \\ |misMatch| \end{Bmatrix} \quad (2)$$

$$D_{Affine} = Match + max\begin{Bmatrix} |max(d,e) + e| \\ |misMatch| \end{Bmatrix} \quad (3)$$

$$N = \lceil \log_2 D \rceil + 1 \quad (4)$$

In proteins PSA case, *match* and *mismatch* values correspond to the maximum and minimum values of the score matrix $s(S1_i, S2_j)$, respectively. For example, the maximum value $\{match = 17\}$ and minimum value $\{mismatch = -8\}$ correspond to the score matrix *PAM250*.

Taking into account the above equations, we design hardware-oriented algorithms for performing NWA and SWA using arithmetic mod-2^N. Algorithms 5 and 6 describe the hardware-oriented NWA and SWA with linear gap penalty, respectively. Algorithms 7 and 8 describe the hardware-oriented NWA with affine gap penalty considering two matrices or three matrices, respectively. Algorithms 9 and 10 describe the hardware-oriented SWA with affine gap penalty considering two matrices or three matrices, respectively.

Algorithm 5. NWA with linear gap penalty using arithmetic mod2^N

Input: Base-pair-sequences *S1={S1*$_1$*… S1*$_n$*}* and *S2={S2*$_1$*… S2*$_m$*}*
Output: Matrices H and A_H
Pre-computation: $s'(S1_i, S2_j) = \boldsymbol{mod2^N}(\, s(S1_i, S2_j) - \boldsymbol{g}\,)$

//Initialization

1. $H_{(0,0)} = \boldsymbol{mod2^N}(0)$
2. $\boldsymbol{for}\ i = 1{:}n\ \boldsymbol{do}\ H_{(i,0)} = \boldsymbol{mod2^N}(H_{(i-1,0)} + g)$
3. $\boldsymbol{for}\ j = 1{:}m\ \boldsymbol{do}\ H_{(0,j)} = \boldsymbol{mod2^N}(H_{(0,j-1)} + g)$

//Calculate similarity matrix H and arrow matrix A_H
$\boldsymbol{for}\ i = 1{:}n\ \boldsymbol{do}$
  $\boldsymbol{for}\ j = 1{:}m\ \boldsymbol{do}$
4. $\{H, A_H\}_{(i,j)} =$

$$\mathbf{max\_}\boldsymbol{mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j-1)} + s'(S1_i, S2_j) + \boldsymbol{g}) \\ \boldsymbol{mod2^N}(H_{(i-1,j)} + g) \\ \boldsymbol{mod2^N}(H_{(i,j-1)} + g)\end{bmatrix}, \begin{bmatrix}diag(\searrow) \\ up(\downarrow) \\ left(\rightarrow)\end{bmatrix}\right\}$$

  $\boldsymbol{end\ for}$
$\boldsymbol{end\ for}$
$\boldsymbol{return}\ H$ and A_H

Algorithm 6. SWA with linear gap penalty using arithmetic mod2^N

Input: Base-pair-sequences *S1={$S1_1$... $S1_n$}* and *S2={$S2_1$... $S2_m$}*
Output: Matrices *H* and A_H, and coordinates *[I,J] of the cell of the matrix H with MLSV*
Pre-computation: $s'(S1_i, S2_j) = \boldsymbol{mod2^N}(s(S1_i, S2_j) - \boldsymbol{g})$

//Initialization
1. $H_{(0,0)} = \boldsymbol{mod2^N}(0)$
2. $\boldsymbol{for}\ i = 1{:}n\ \boldsymbol{do}\ H_{(i,0)} = \boldsymbol{mod2^N}(0)$
3. $\boldsymbol{for}\ j = 1{:}m\ \boldsymbol{do}\ H_{(0,j)} = \boldsymbol{mod2^N}(0)$

//Calculate similarity matrix H, arrow matrix A_H and coordinates [I,J]
$\boldsymbol{for}\ i = 1{:}n\ \boldsymbol{do}$
  $\boldsymbol{for}\ j = 1{:}m\ \boldsymbol{do}$
4. $\{H', A'_H\}_{(i,j)} =$

$$\mathbf{max\_}\boldsymbol{mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j-1)} + s'(S1_i, S2_j) + \boldsymbol{g}) \\ \boldsymbol{mod2^N}(H_{(i-1,j)} + g) \\ \boldsymbol{mod2^N}(H_{(i,j-1)} + g)\end{bmatrix}, \begin{bmatrix}diag(\searrow) \\ up(\downarrow) \\ left(\rightarrow)\end{bmatrix}\right\}$$

//Process the value Zero by the threshold function maxTH
5. $\{H, A_H\}_{(i,j)} = \boldsymbol{maxTH}\left\{\begin{bmatrix}\mathbf{0} \\ H'\end{bmatrix}, \begin{bmatrix}null \\ A'_H\end{bmatrix}\right\}$

//Seek coordinates [I,J] of the cell with MLSV of the matrix H
  $\boldsymbol{if}\ H_{(i,j)} < MLSV$
6. $\boldsymbol{then}\ \{\text{MLSV}, \text{posIJ}\} = \{\text{MLSV}, posIJ\}$
7. $\boldsymbol{else}\ \{\text{MLSV}, \text{posIJ}\} = \{H_{(i,j)}, [i,j]\}$
  $\boldsymbol{end\ if}$
  $\boldsymbol{end\ for}$
$\boldsymbol{end\ for}$
$\boldsymbol{return}\ H, A_H$ *and* $[I,J]$

Algorithm 7. NWA with affine gap penalty of three matrices using arithmetic mod2^N

Input: Base-pair-sequences *S1={$S1_1$... $S1_n$}* and *S2={$S2_1$... $S2_m$}*
Output: Matrices *H* and A_H

//Initialization
1. $H_{(0,0)} = \boldsymbol{mod2^N}(0)$
2. $H_{(0,1)} = H_{(1,0)} = Gx_{(0,1)} = Gx_{(1,0)} = Gy_{(0,1)} = Gy_{(1,0)} = \boldsymbol{mod2^N}(d)$
3. $\boldsymbol{for}\ i = 2{:}n\ \boldsymbol{do}\ H_{(i,0)} = \boldsymbol{mod2^N}(H_{(i-1,0)} + \mathrm{e})$
4. $\boldsymbol{for}\ j = 2{:}m\ \boldsymbol{do}\ H_{(0,j)} = \boldsymbol{mod2^N}(H_{(0,j-1)} + e)$
5. $\boldsymbol{for}\ i = 2{:}n\ \boldsymbol{do}\ Gy_{(i,0)} = \boldsymbol{mod2^N}(Gy_{(i-1,0)} + \mathrm{e})$
6. $\boldsymbol{for}\ j = 2{:}m\ \boldsymbol{do}\ Gx_{(0,j)} = \boldsymbol{mod2^N}(Gx_{(0,j-1)} + e)$

//Calculate value matrices H, G_X and G_Y, and arrow matrix A_H
$\boldsymbol{for}\ i = 1{:}n\ \boldsymbol{do}$
  $\boldsymbol{for}\ j = 1{:}m\ \boldsymbol{do}$
7. $\{H, A_H\}_{(i,j)} =$

$$\mathbf{max\_}\boldsymbol{mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j-1)} + s(S1_i, S2_j)) \\ \boldsymbol{mod2^N}(Gx_{(i-1,j-1)} + s(S1_i, S2_j)) \\ \boldsymbol{mod2^N}(Gy_{(i-1,j-1)} + s(S1_i, S2_j))\end{bmatrix}, \begin{bmatrix}diag(\searrow) \\ up(\downarrow) \\ left(\rightarrow)\end{bmatrix}\right\}$$

8. $Gx_{(i,j)} = \boldsymbol{max\_mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j)} + d) \\ \boldsymbol{mod2^N}(Gx_{(i-1,j)} + e)\end{bmatrix}\right\}$
9. $Gy_{(i,j)} = \boldsymbol{max\_mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i,j-1)} + d) \\ \boldsymbol{mod2^N}(Gx_{(i,j-1)} + e)\end{bmatrix}\right\}$

  $\boldsymbol{end\ for}$
$\boldsymbol{end\ for}$
$\boldsymbol{return}\ H$ *and* A_H

Algorithm 8. NWA with affine gap penalty of two matrices using arithmetic mod2^N

Input: Base-pair-sequences *S1={$S1_1$... $S1_n$}* and *S2={$S2_1$... $S2_m$}*
Output: Matrices *H* and A_H

//Initialization
1. $H_{(0,0)} = \boldsymbol{mod2^N}(0)$
2. $G_{(0,0)} = H_{(0,1)} = H_{(1,0)} = G_{(0,1)} = G_{(1,0)} = \boldsymbol{mod2^N}(d)$
3. $\boldsymbol{for}\ i = 2{:}n\ \boldsymbol{do}\ H_{(i,0)} = \boldsymbol{mod2^N}(H_{(i-1,0)} + \mathrm{e})$
4. $\boldsymbol{for}\ j = 2{:}m\ \boldsymbol{do}\ H_{(0,j)} = \boldsymbol{mod2^N}(H_{(0,j-1)} + e)$
5. $\boldsymbol{for}\ i = 2{:}n\ \boldsymbol{do}\ G_{(i,0)} = \boldsymbol{mod2^N}(G_{(i-1,0)} + \mathrm{e})$
6. $\boldsymbol{for}\ j = 2{:}m\ \boldsymbol{do}\ G_{(0,j)} = \boldsymbol{mod2^N}(G_{(0,j-1)} + e)$

//Calculate value matrices H and G, and arrow matrices A_H and A_G
$\boldsymbol{for}\ i = 1{:}n\ \boldsymbol{do}$
  $\boldsymbol{for}\ j = 1{:}m\ \boldsymbol{do}$
7. $\{H, A_H\}_{(i,j)} =$

$$\boldsymbol{max\_mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j-1)} + s(S1_i, S2_j)) \\ \boldsymbol{mod2^N}(G_{(i-1,j-1)} + s(S1_i, S2_j))\end{bmatrix}, \begin{bmatrix}diag(\searrow) \\ up|left(\downarrow\rightarrow)\end{bmatrix}\right\}$$

8. $\{G, A_G\}_{(i,j)} =$

$$\boldsymbol{max\_mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j)} + d) \\ \boldsymbol{mod2^N}(G_{(i-1,j)} + e) \\ \boldsymbol{mod2^N}(H_{(i,j-1)} + d) \\ \boldsymbol{mod2^N}(G_{(i,j-1)} + e)\end{bmatrix}, \begin{bmatrix}up(\downarrow) \\ up(\downarrow) \\ left(\rightarrow) \\ left(\rightarrow)\end{bmatrix}\right\}$$

  $\boldsymbol{end\ for}$
$\boldsymbol{end\ for}$
$\boldsymbol{return}\ H$ *and* A_H

Algorithm 9. SWA with affine gap penalty of three matrices using arithmetic mod2^N

Input: Base-pair-sequences *S1={$S1_1$... $S1_n$}* and *S2={$S2_1$... $S2_m$}*
Output: Matrices *H* and A_H, and coordinates *[I,J] of the cell of the matrix H with MLSV*

//Initialization
1. $H_{(0,0)} = \boldsymbol{mod2^N}(0)$
2. $H_{(i,0)} = Gx_{(i,0)} = Gy_{(i,0)} = \boldsymbol{mod2^N}(0)$
3. $H_{(0,j)} = Gx_{(0,j)} = Gy_{(0,j)} = \boldsymbol{mod2^N}(0)$

//Calculate value matrices H, G_X and G_Y, and arrow matrix A_H and coordinates [I,J]
$\boldsymbol{for}\ i = 1{:}n\ \boldsymbol{do}$
  $\boldsymbol{for}\ j = 1{:}m\ \boldsymbol{do}$
4. $\{H', A'_H\}_{(i,j)} =$

$$\mathbf{max\_}\boldsymbol{mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j-1)} + s(S1_i, S2_j)) \\ \boldsymbol{mod2^N}(Gx_{(i-1,j-1)} + s(S1_i, S2_j)) \\ \boldsymbol{mod2^N}(Gy_{(i-1,j-1)} + s(S1_i, S2_j))\end{bmatrix}, \begin{bmatrix}diag(\searrow) \\ up(\downarrow) \\ left(\rightarrow)\end{bmatrix}\right\}$$

5. $Gx_{(i,j)} = \boldsymbol{max\_mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j)} + d) \\ \boldsymbol{mod2^N}(Gx_{(i-1,j)} + e)\end{bmatrix}\right\}$

6. $Gy_{(i,j)} = \boldsymbol{max\_mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i,j-1)}+d)\\ \boldsymbol{mod2^N}(Gx_{(i,j-1)}+e)\end{bmatrix}\right\}$

//Process the value Zero by the threshold function maxTH

7. $\{H, A_H\}_{(i,j)} = \boldsymbol{maxTH}\left\{\begin{bmatrix}\mathbf{0}\\ H'\end{bmatrix}, \begin{bmatrix}null\\ A'_H\end{bmatrix}\right\}$

//Seek coordinates [I,J] of the cell with MLSV of the matrix H

if $H_{(i,j)} < MLSV$

8. ***then*** $\{\text{MLSV}, \text{posIJ}\} = \{\text{MLSV}, posIJ\}$
9. ***else*** $\{\text{MLSV}, \text{posIJ}\} = \{H_{(i,j)}, [i,j]\}$

end if

end for

end for

return H, A_H *and* $[I,J]$

Algorithm 10. SWA with affine gap penalty of two matrices using arithmetic mod2^N

Input: Base-pair-sequences *S1={S1₁… S1ₙ}* and *S2={S2₁… S2ₘ}*
Output: Matrices *H* and A_H, and coordinates *[I,J] of the cell of the matrix H with MLSV*

//Initialization

1. $H_{(0,0)} = \boldsymbol{mod2^N}(0)$
2. ***for*** $i = 1{:}n$ ***do*** $H_{(i,0)} = G_{(i,0)} = \boldsymbol{mod2^N}(0)$
3. ***for*** $j = 1{:}m$ ***do*** $H_{(0,j)} = G_{(0,j)} = \boldsymbol{mod2^N}(0)$

//Calculate value matrices H and G, and arrow matrices A_H and A_G and coordinates [I,J]

for $i = 1{:}n$ ***do***

for $j = 1{:}m$ ***do***

4. $\{H', A'_H\}_{(i,j)} = \boldsymbol{max\_mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j-1)}+s(S1_i,S2_j))\\ \boldsymbol{mod2^N}(G_{(i-1,j-1)}+s(S1_i,S2_j))\end{bmatrix}, \begin{bmatrix}diag(\searrow)\\ up|left(\downarrow\rightarrow)\end{bmatrix}\right\}$

5. $\{G, A_G\}_{(i,j)} = \boldsymbol{max\_mod2^N}\left\{\begin{bmatrix}\boldsymbol{mod2^N}(H_{(i-1,j)}+d)\\ \boldsymbol{mod2^N}(G_{(i-1,j)}+e)\\ \boldsymbol{mod2^N}(H_{(i,j-1)}+d)\\ \boldsymbol{mod2^N}(G_{(i,j-1)}+e)\end{bmatrix}, \begin{bmatrix}up(\downarrow)\\ up(\downarrow)\\ left(\rightarrow)\\ left(\rightarrow)\end{bmatrix}\right\}$

//Process the value Zero by the threshold function maxTH

6. $\{H, A_H\}_{(i,j)} = \boldsymbol{maxTH}\left\{\begin{bmatrix}\mathbf{0}\\ H'\end{bmatrix}, \begin{bmatrix}null\\ A'_H\end{bmatrix}\right\}$

//Seek coordinates [I,J] of the cell with MLSV of the matrix H

if $H_{(i,j)} < MLSV$

7. ***then*** $\{\text{MLSV}, \text{posIJ}\} = \{\text{MLSV}, posIJ\}$
8. ***else*** $\{\text{MLSV}, \text{posIJ}\} = \{H_{(i,j)}, [i,j]\}$

end if

end for

end for

return H, A_H *and* $[I,J]$

From the proposed algorithms, it is important to mention the following:

1) Matrices *H*, *G*, G_X and G_Y store the score values calculated in the modular space of module 2^N, and the matrices A_H and A_G store the arrows associated to the maximum selected score value of each cell, where the arrows of A_H are used to perform the TBP.
2) Function *mod2^N* is implemented in hardware by using arithmetic units of N-bit, disregarding the generated carry.
3) Function *max_mod2^N* determinates the maximum score value into a modular space and can be calculated by successive *A<B* modular comparisons. For example, $max\_mod2^N[X,Y,Z] = max\_mod2^N[max\_mod2^N[X,Y],Z]$. Then, each comparison $max\_mod2^N[A,B]$ is implemented using a two's complement adder of N-bit for performing *A<B mod2^N*, that is, *A<B* $mod2^N = MSB[A-B]$. For example, if *N=4*, *A=15*, and *B=1*, then $max\_mod2^N[A,B] = B$ due to $MSB[\text{“1111”} - \text{“0001”}] = MSB[\text{“1110”}] = 1$; therefore *A<B* = '1', then *B* is the greatest value.
4) Value *Zero* of SWA is not included in the function *max_mod2^N*, in order to perform the modular comparisons. However, the value *Zero* is taking into account by the function *maxTH*.
5) Function *maxTH* prevents the processing of negative score values generated by the function *max_mod2^N*, discriminating the value *Zero* of other $mod2^N(k*2^N)$ values (e.g. 16,32,48… for *N=4*) until to achieve an adequate aligned segment with high similarity, that is, a score value greater than 2^{T-1}, where *T>N*. After that, the score value of the function *maxTH* is not taking into account, fixing the number of bits of the function *maxTH* to the value *T*.
6) Value $-\infty$ in $G_{(0,0)}, G_{(0,1)}$ and $G_{(1,0)}$ is replaced by the value *mod2^N(d)* for affine gap penalty, getting the same functionality.
7) Coordinates $[I,J]$ of the start-cell of TBP for SWA are calculated using simple methods such as *dot-matrix*, *k-mers*, among others, or using an functional block implemented with integer arithmetic circuits for calculating the value *MLSV* and the coordinates $[I,J]$. These calculations are not possible using arithmetic mod-2^N because the module 2^N only allows the calculation of the maximum score value between neighbor cells of the matrices *H*, *G*, G_X, and G_Y.
8) $\boldsymbol{mod2^N}(H_{(i-1,j-1)} + s'(x_i, y_j) + \boldsymbol{g})$ allows performing the addition of the value *g* of the gap penalty using only one adder for NWA or SWA with linear gap. The above addition of the value g is compensated by subtracting the value g to the score matrix $s(S1_i, S2_j)$, obtaining the new constant matrix $s'(S1_i, S2_j)$.

In order to verify the proposed hardware-oriented algorithms for NWA and SWA with linear gap penalty using arithmetic mod-2^N, they were implemented in Wolfram Mathematica from the software tool developed by [18]. The similarity matrix *H* and the arrows matrix A_H for NWA and SWA with linear gap penalty using integer arithmetic are shown in Figures 3.a and 4.a, and using arithmetic mod-2^N in Figures 3.b and 4.b, where $D_{Linear} = 7 = (3 + 2|-2|)$ is calculated using Equation 2 for the score scheme $\{match, mismatch, gap\} = \{3, -1, -2\}$, and $N = 4 = \lceil \log_2 7 \rceil + 1$ is calculated using Equation 4. In these cases, identical alignments are obtained using integer arithmetic or mod-2^N arithmetic because the obtained arrow matrices A_H are the same.

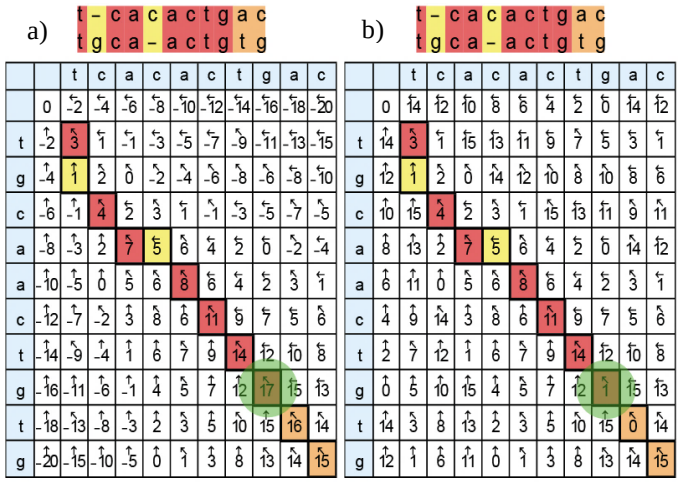

Fig 3. Matrices H and A_H for NWA with linear gap penalty using a) integer arithmetic or b) mod-2^N arithmetic.

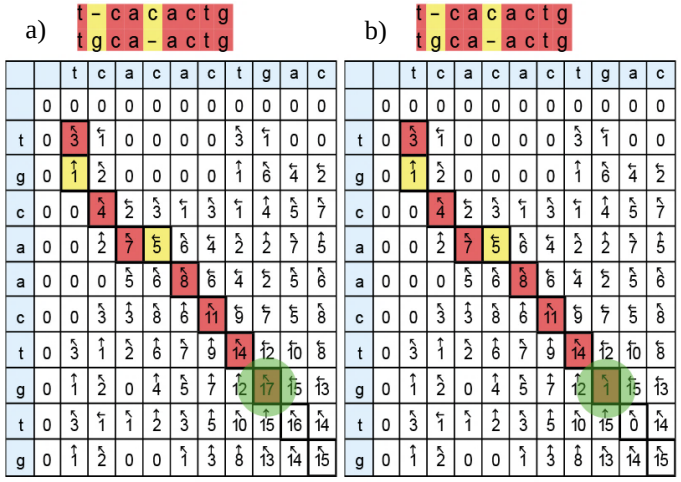

Fig 4. Matrices H and A_H for SWA with linear gap penalty using a) integer arithmetic or b) mod-2^N arithmetic.

It is important to mention that the cells circular-highlighted of the matrix H of Figures 3.a and 4.a have the *score-value=17* when they are calculated using integer arithmetic, and the same cells of the matrix H of Figures 3.b and 4.b have the *score-value=1* when they are calculated using arithmetic mod-2^N. In the first case, the $score\text{-}value = 17$ was obtained from $max\{14+3, 12-2, 12-2\}$ and in the second case the $score\text{-}value = 1$ was obtained from $max\_mod2^4\{mod2^4[14+3], mod2^4[12-2], mod2^4[12-2]\} = \max\_mod2^4\{1,10,10\} = 1$ because the $MSB[mod2^4[1-10]] = MSB[mod2^4["0001"-"1010"]] = MSB["0111"] =$ '0', then *1<10 mod2⁴ = '0'*.

IV. Design of PEs for NWA using Arithmetic mod-2^N

In this section, we describe the design of three PEs from proposed hardware-oriented algorithms for performing NWA with linear gap or affine gap penalties using arithmetic mod-2^N.

Figures 5, 6 and 7 show the three PEs designed from algorithms 5, 7 and 8 for performing NWA with linear gap penalty (PE_{NWA}-1), affine gap penalty of three matrices (PE_{NWA}-2) and affine gap penalty of two matrices (PE_{NWA}-3), respectively, and they can be used to design PEs for performing SWA.

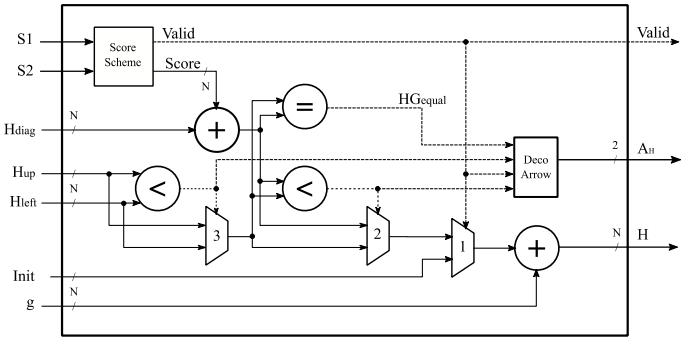

Fig 5. PE_{NWA}-1. PE for NWA with linear gap penalty.

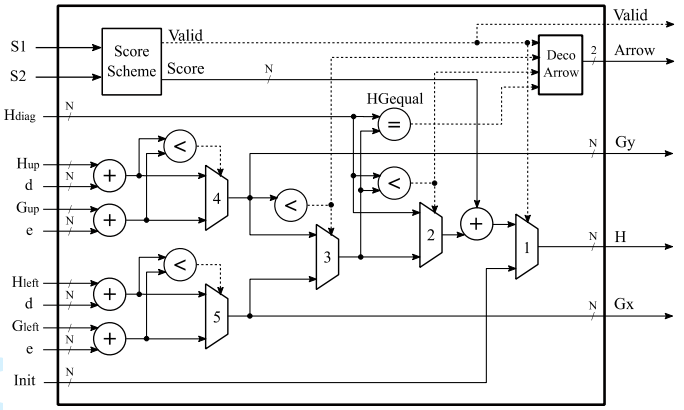

Fig 6. PE_{NWA}-2. PE for NWA with affine gap penalty of three matrices.

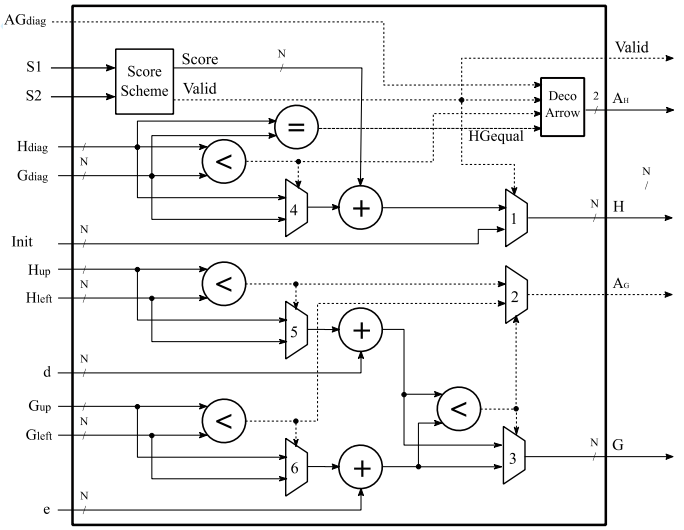

Fig 7. PE_{NWA}-3. PE for NWA with affine gap penalty of two matrices.

From the designed PEs, it is important to mention the following:

1) Signal *HGequal* is used to codify the two-arrow cases $\{\searrow,\downarrow\}$ and $\{\searrow,\rightarrow\}$, increasing the number of optimal PSAs that can be obtained by the TBP. For example, if $\{H, A_H\}_{(i,j)} = \max[\{12,12,11\}, \{\searrow,\downarrow,\rightarrow\}]$, then the arrows $\{\searrow\}$ and $\{\downarrow\}$ should be stored. Generally, the hardware accelerators store only one arrow considering priority

criteria.

2) Signal A_H is codified as follows: $\{\searrow\}$ = "11", $\{\downarrow\}$ = "01", $\{\rightarrow\}$ = "10", and the cases $\{null\}$, $\{\searrow, \downarrow\}$ and $\{\searrow, \rightarrow\}$ use the code "00".
3) Signal *Init* is used to perform the initialization functions $H_{(0,0)}, H_{(i,0)}$ *and* $H_{(0,j)}$, and it is selected by *mux-1*.
4) Signal *g (Gap penalty)* is processed using only one adder (See Figure 5), due to the addition $H_{(i-1,j-1)} + s(S1_i, S2_j)$ of Algorithms 1 and 4 was replaced by the addition $\boldsymbol{mod2^N}(H_{(i-1,j-1)} + s'(S1_i, S2_j) + \boldsymbol{g})$ for Algorithms 5 and 6**,** reducing the number of adders used.
5) Signal A_G indicates the source path of the selected maximum score value of each cell of the matrix *G* (See Figure 7).
6) Signal *AGdiag* is used to codify the signal A_H (See Figure 7), which is required to perform the TBP.

V. Design of PEs for SWA based on PEs for NWA

In this section, we describe the design of three PEs for performing SWA (PE_{SWA}) with linear gap or affine gap penalties from their proposed hardware-oriented algorithms. These PE_{SWA} are designed using PE_{NWA}, which use arithmetic mod-2^N.

PEs for SWA perform the function maxTH and calculate the coordinates $[I,J]$ from MLSV using arithmetic mod-2^{T+N-1}. Therefore, the number of bits *T* must be selected by considering the follow assumptions:

1) Integer values of the matrix *H* (*Hint*) do not decrease to zero after they reach the threshold value 2^{T+N-1},
2) The maximum distance $D_{TH} = |MLSV - Hint|$ for calculating the coordinates $[I,J]$ from MLSV is less than 2^{T+N-1}.

PEs for performing SWA with linear gap penalty (PE_{SWA}-1), affine gap penalty of three matrices (PE_{SWA}-2) and affine gap penalty of two matrices (PE_{SWA}-3) were designed from the respective designed PEs for NWA, and adding the functional blocks *mTH*, *coordIJ* and *Counter*, as it is shown in Figure 8.

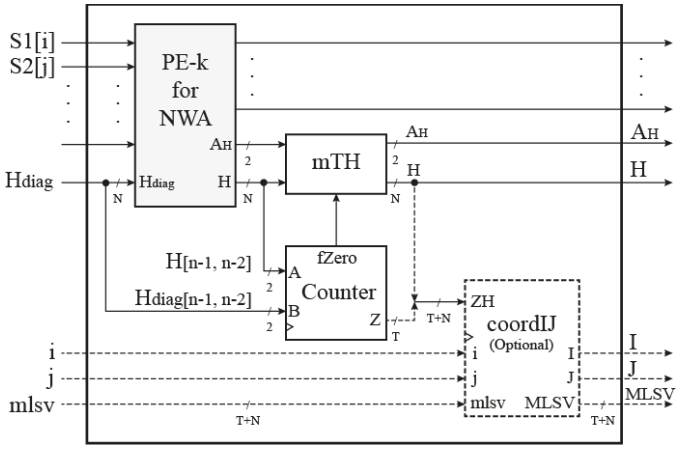

Fig 8. PE_{SWA}-k. PEs for SWA using PEs for NWA (PE_{NWA}-k).

Block *mTH* implements the function *maxTH* and it selects $\{H, A_H\} = \{H^{NWA}, A_H^{NWA}\}$ when $fZero = '1'$, or $\{H, A_H\} = \{0, null\}$ when $fZero = '0'$.

Block *coordIJ* calculates the coordinates $[I,J]$ by comparing the values $ZH = Z\&H$ and *mlsv*. In this case, the outputs *I*, *J* and *MLSV* are updated with the respective values *i*, *j*, and *ZH* when $MLSV < ZH$. Otherwise, these outputs retain their previous value. This block is optional because the coordinates $[I,J]$ can be calculated by other methods.

Block *Counter* increases the modular space from 2^N to 2^{N+T}. The above allows to perform the function *maxTH* and calculate the coordinates $[I,J]$ using arithmetic mod-2^{N+T}. In this case, $Z_{[T-1..0]}$ corresponds to most significant bits and $H_{[N-1..0]}$ corresponds to less significant bits. The design of this block is shown in Figure 9, and it is composed of the sub-blocks Zero Crossing Detector (*ZCD*), Accumulator (*ACC)* and Threshold (*TH*).

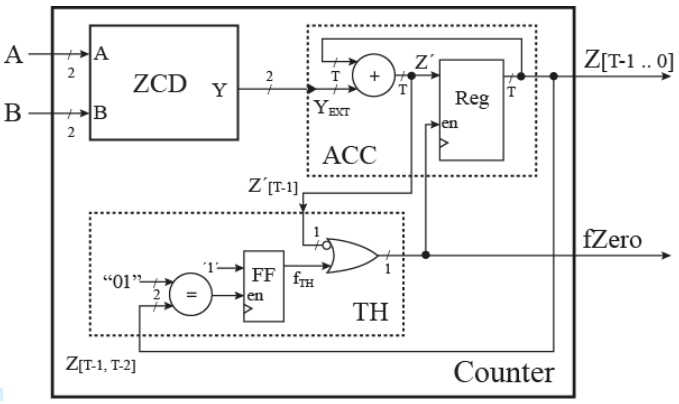

Fig 9. Block *Counter*.

1) Sub-block *ZCD* detects positive and negative zero crossing of the value *H* into the mod-2^N space, from the 2-bit inputs $A = H_{[n-1,n-2]}$ and $B = H_{diag\,[n-1,n-2]}$. The signed output $Y = "b_1 b_0"$ is codified as $Y = "01" = +1$ when is presented a positive zero crossing ($A\&B = "0011"$), $Y = "11" = -1$ when is presented a negative zero crossing ($A\&B = "1100"$) or $Y = "00" = 0$ when are presented other cases. However, the output *Y* is extended in the block *ACC* by extending the sign-bit, that is, $Y_{EXT} = "b_1 \ldots b_1 b_0"$.
2) Sub-block *ACC* accumulates the number of zero crossing of the signal *H* by adding the signals Y_{EXT} and *Z*. Then, *Z* increases when $Y_{EXT} = +1$, decreases when $Y_{EXT} = -1$ and hold its value when $Y_{EXT} = 0$. Signal *Z* also is held when $Z = 0$ and $Y = -1$, since the register *Reg* is disabled when $fZero = '0'$, that is, when $Z'_{[T-1]} = '1'$ and $f_{TH} = '0'$.
3) Sub-block *TH* generates the flag $fZero = \overline{Z'_{[T-1]}}\ OR\ f_{TH}$ to prevent values in Z less than *zero* $(Z < 0)$ until *Z* reaches the threshold value $Z \geq 2^{T-1}$. In this case, $Z < 0$ when $Z_{[T-1]} = '1'$, whereas $Z \geq 2^{T-1}$ when $Z_{[T-1,T-2]} =$ "01". Then, values $Z < 0$ are allowed after $Z_{[T-1,T-2]} =$ "01", because the flag $fZero$ is fixed to $'1'$ when the flip-flop *FF* is enabled, then f_{TH} is fixed to $'1'$. It is important to mention that f_{TH} return to $'0'$ if and only if this sub-block is reset.

Figure 10 shows the range of the number of bits required by the arithmetic units of each PE for calculating the score values of the matrix *H* using integer or arithmetic mod-2^N, for NWA

or SWA of DNA (Continuous lines) or proteins (Dashed lines).

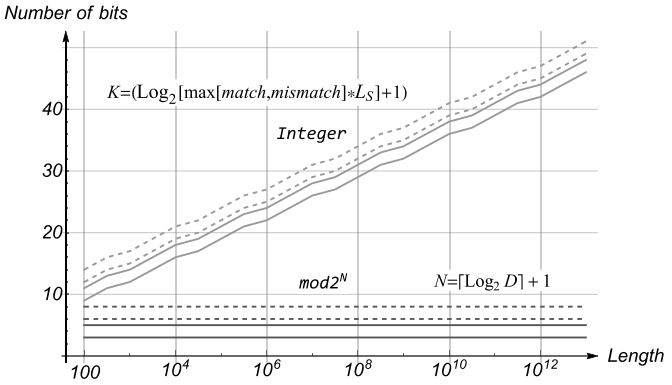

Fig. 10. Range of number of bits for NWA or SWA of DNA (Continuous) and proteins (Dashed), using integer or mod-2^N arithmetic.

From the above figure, it is possible to observe that the number of bits K increases according to the base-pair sequence length using integer arithmetic. In this case, the maximum number of bits K is given by Equation 5, where L_S is the length of the shortest sequence of the base-pair sequence, and the value '1' corresponds to the sign bit.

$$K = (\log_2[Max[\text{match}, \text{mismatch}] * L_S] + 1) \quad (5)$$

However, the number of bits $N = \lceil \log_2 D \rceil + 1$ is small and fixed using arithmetic mod-2^N for alignment of a base-pair sequence of any length, requiring from 3-bit to 5-bit for performing NWA or SWA of DNA, and from 6-bit to 8-bit for performing NWA or SWA of proteins, considering the score matrix *PAM250* and gap penalty values *{g,d,e}* less than the value *match=17*.

For example, the calculation of the score values of the matrix H of NWA for a DNA base-pair sequence of 1-Mbp (2^{20}) using integer arithmetic and the score scheme *{match, mismatch, gap} = {5,-2,-3}*, require PEs of 24-bit to represent any value in the range $\{-3x2^{20}, 5x2^{20}\}$ using two's complement representation, where the range is determined by the minimum and maximum values of the scoring scheme and the base-pair sequence length. Unlike, PEs of 5-bit are required using arithmetic mod-2^N for the above case, because the maximum distance D_{Linear} is $D = 11 = \{5 + 2|-3|\}$ and $N = 5 = \lceil \log_2 11 \rceil + 1$.

VI. Synthesis and Performance Results

In this section, we present the synthesis and performance results of the designed PEs for NWA using arithmetic mod-2^N. The PEs were synthesized in the low-cost SoC-FPGA Cyclone V 5CSEBA6U23I7NDK of Intel, included in the DE10nano development board of Terasic. It is important to mention that if the area of the PE is minimal, then the number of PEs that can be embedded into an FPGA device is increased, allowing the implementation of the proposed hardware-oriented algorithms on 1D/2D array architectures.

Tables 1 and 2 present the synthesis results of the designed PEs using arithmetic mod-2^N for NWA of DNA or proteins, respectively, where N is the number of bits that they process. From the obtained results, it is possible to observe that:

1) PE area is increased and PE frequency is decreased when the number of bits N is increased.
2) PEs of proteins require more area than PEs of DNA, because the values and size of the score matrix *PAM250* for proteins are bigger than the values and size of the score matrix *match/mismatch* for the DNA.
3) PEs for affine gap of two matrices are more hardware efficient than PEs for affine gap with three matrices, because they require less area and operate with higher frequency.

TABLE I
Synthesis Results of PEs using Arithmetic mod-2^N for NWA of DNA

N	Linear Gap		Affine Gap 3M		Affine Gap 2M	
	ALMs	Freq (MHz)	ALMs	Freq (MHz)	ALMs	Freq (MHz)
3	13.7	215	19.3	203	19.7	253
4	20.2	155	30.8	114	23.7	187
5	25.8	152	39.0	114	28.6	169
6	29.6	140	47.2	107	33.5	176
7	35.6	134	50.7	110	39.9	154
8	39.8	121	57.4	104	46.5	163

TABLE II
Synthesis Results of PEs using Arithmetic mod-2^N for NWA of Proteins

N	Linear Gap		Affine Gap 3M		Affine Gap 2M	
	ALMs	Freq (MHz)	ALMs	Freq (MHz)	ALMs	Freq (MHz)
5	77.2	121	91	118	83.0	164
6	83.4	119	101.0	102	87.5	157
7	85.3	119	104.5	108	92.3	151
8	90.2	114	108.5	103	97.0	152

VII. Conclusions

In this paper, we propose hardware-oriented algorithms for performing NWA or SWA using arithmetic mod-2^N. Also, we propose the equations for calculating the maximum difference D between all the processed data by the comparison function *max_mod2^N* from the selected score scheme for NWA or SWA with linear gap or affine gap penalties. The deduce equations allow performing the comparison $A<B$ into modular space using arithmetic units of small number of bits. The hardware-oriented algorithms perform NWA or SWA for a base-pair sequence of any length due to the arithmetic mod-2^N, which allow the reduction of the number of bits N that processes the arithmetic units of each PE, minimizing the area of the PE and increasing the PE density into an FPGA device.

The designed PEs can be used to design 1D/2D array hardware architectures for performing NWA or SWA, of DNA or proteins, using any value of score matrix and linear gap or affine gap penalties (score scheme), and they can be used to design PEs for performing SWA. In this case, it is necessary to add the blocks *mTH*, *Counter* and optionally *coordIJ*, which increase the modular space from 2^N to 2^{N+T} for performing the function *maxTH* and calculating the coordinates $[I,J]$ of the start-cell from *MLSV* for performing TBP.

Also, it is important to conclude that the proposed equations for calculating the distance D and the hardware improvements using arithmetic mod-2^N, can be extended to other DP-based algorithms used in other applications, allowing their efficient

hardware implementations since the proposed hardware-oriented algorithms minimize and fix the number of bits *N* that processes the arithmetic units of the PEs. Nevertheless, the proposed hardware-oriented algorithms are inefficient for software implementations since they require more computing resources, reducing their performance.

REFERENCES

[1] W. Liu *et al*, "Streaming Algorithms for Biological Sequence Alignment on GPUs," IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 9, pp. 1270-1281, Sept. 2007.

[2] Y. Shibberu and A. Holder, "A Spectral Approach to Protein Structure Alignment," IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 8, no. 4, pp. 867-875, July-Aug. 2011.

[3] A. Jiwan and S. Singh, "A review on RNA pseudoknot structure prediction techniques," 2012 International Conference on Computing, Electronics and Electrical Technologies (ICCEET), Kumaracoil, pp. 975-978, March 2012.

[4] C. Rastogi *et al*, "Accurate and sensitive quantification of protein-DNA binding affinity," Proceedings of the National Academy of Sciences, Apr 2018, 115 (16) E3692-E3701

[5] F. Wang *et al*, "PDP: parallel dynamic programming," IEEE/CAA Journal of Automatica Sinica, vol. 4, no. 1, pp. 1-5, Jan. 2017.

[6] Q. Wei, D. Liu and H. Lin, "Value Iteration Adaptive Dynamic Programming for Optimal Control of Discrete-Time Nonlinear Systems," IEEE Transactions on Cybernetics, vol. 46, no. 3, pp. 840-853, March 2016.

[7] T.M. Khan *et al*, "Real-time iris segmentation and its implementation on FPGA," Springer Journal of Real-Time Image Processing, pp. 1-14, Feb. 2019.

[8] K. Gai *et al*, "Privacy-Preserving Content-Oriented Wireless Communication in Internet-of-Things," IEEE Internet of Things Journal, vol. 5, no. 4, pp. 3059-3067, Aug. 2018.

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

[10] T. Smith and M. Waterman, "Identification of common molecular subsequences," Journal of Molecular Biology, vol. 147, no. 1, pp. 195–197, 1981.

[11] C.W. Yu *et al*, "A Smith-Waterman Systolic Cell," Springer Field Programmable Logic and Application (FPL), vol. 2778, pp. 375–384, Sep. 2003.

[12] M. Gok and C. Yilmaz, "Efficient Cell Designs for Systolic Smith-Waterman Implementations," IEEE International Conference on Field Programmable Logic and Applications, Madrid, pp. 1-4, Aug. 2006.

[13] P.K. Mensah, E.K. Bankas and M.M. Iddrisu, "RNS Smith-Waterman Accelerator based on the moduli set 2^n, 2^{n-1}, 2^{n-1}-1," IEEE 7th International Conference on Adaptive Science & Technology (ICAST), Accra, pp. 1-8, Aug. 2018.

[14] G. Davida and B. Litow, "Fast Parallel Arithmetic Via Modular Representation," Society for industrial and applied mathematics, vol. 20, no. 4, pp. 756–765, 1991.

[15] Dong Wei, N. Ansari and Jianguo Chen, "A compressed and dynamic-range-based expression of timestamp and period for timestamp-based schedulers," GLOBECOM'01. IEEE Global Telecommunications Conference (Cat. No.01CH37270), San Antonio, TX, vol.4, pp. 2353-2357, Aug. 2001.

[16] O. Gotoh, "An improved algorithm for matching biological sequences," Journal of Molecular Biology, vol. 162, pp. 705–708, 1982.

[17] D. Huson, "Algorithms in Bioinformatics I", ZBIT, Uni Tübingen, Book. 2002

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