Bit-Parallelism Score Computation with Multi Integer Weight

Setyorini[1,3], Kuspriyanto[1], Dwi Hendratmo Widyantoro[1], and Adi Pancoro[2]

[1]School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia
[2]School of Life Sciences & Technology, Institut Teknologi Bandung, Indonesia
[3]School of Computing, Telkom University, Indonesia

Abstract: Dynamic Programming (DP) is still the core algorithm in many biological analysis tools, especially similarity analysis. DP always promises optimal solutions, but as the size and number of sequences increase, time performance decreases. This makes various researches related to the improvement of time performance offered. Performance improvement is offered by adding process units that work together in parallel environments or reducing the accuracy by only calculating the input part. Bit-parallelism offers an increase in speed by changing the score matrix process unit to a larger process unit, namely word. Then this word unit will be processed in parallel like word processing on a computer system. Bit-parallelism has been successfully applied to cases with simple weights such as LCS and Edit Distance. Applications in more complex cases, namely with integer weights also exist, but still assume an integer weight for match representation. In the case of protein alignment where there is more than one integer value to represent the match of the residual pair, this solution cannot be applied directly. This paper presents a preliminary research on how to formulate a computational algorithm score bit-parallelism with multi-integer weights. The solution offered is the development of General Integer scoring, by applying functional multi tables. The results show that the algorithm is able to obtain the same score matrix as the DP score matrix, with computing $O(m)$, m is size of the sequence, and $O(b)$ space where b is the range of integer set weights.

Keywords: Sequence Alignment; Dynamic Programming; Bit-Parallelism, General Integer Scoring.

1. Introduction

Dynamic Programming (DP) is a classic optimization algorithm for a problem that can be constructed into sub-problems and can be solved iteratively. All possible sub solutions are visited in this algorithm, so the guarantee of optimal solutions will always be obtained. The way this algorithm works becomes a problem when the size and number of sequences increases and becomes unrealistic at a size large enough.

Adding a processing unit that is run in parallel is one effort to increase the speed of the process. The ability to coordinate to get optimal speed is the focus of the parallel process. The next approach is to tolerate the accuracy. Not all possible solutions are visited, so there is no guarantee of getting an optimal solution. The best solution is strived to provide the best heuristic information, so the effort proposed in this approach is an effort to improve the heuristic information.

In computer science, sequence alignment is an approximate string-matching problem. Similarity of a string is measured by the number of edit operations that must be performed to convert a string into another string. there are 4 techniques for matching strings approximate [1]: DP, Automaton, Filtering and Bit-parallelism. Automaton represents problems into states and transitions. Bit-parallelism is a proposed improvement on DP and automaton by parallelizing the computational process.

Sequence alignment is an important step in biology to analyze the similarity of biological sequences such as: DNA or protein. Similarity analysis is useful to determine the relationship between ancestors or functional similarities. The accuracy of the analysis is determined by the quality and quantity of data that builds a model of similarity.

Received: November 15th, 2018. Accepted: February 14th, 2019
DOI: 10.15676/ijeei.2019.11.1.3

Many sequence alignment devices are available online and offline. A popular sequence alignment application for searching similarity sequences in a database is BLAST [2] and FAST [3]. This application offers an approximate solution, the DP algorithm is used to find similarity in sequence pieces. the best k-pieces become heuristics to build similarities in whole sequences. There is no guarantee of finding an optimal solution. The initial version of BLAST is said to lose 30% significant sequence. Improvements to BLAST and FASTA were carried out by adopting a profile search and motif of sequences, namely the conserved area of the sequence that characterizes a sequence. PSI-BLAST marked conserved positions in the sequence as a guide in aligning.

The approach parallels the sequence alignment device: ScanPS [4], ParAlign [5], Both of them increase speed by parallelizing the Smith-Waterman algorithm and using heuristic help in accelerating the search for solutions. Another approach by using an embedded graphics processing unit (GPU), CUDA is offered for pairwise sequence alignment and MSA [6], to parallel Clustal [7] parallelize MAFFT [8].

Bit-parallelism provides a different approach, increasing speed while maintaining accuracy. Bit-parallelism has been successfully applied to string matching cases, such as LCS [8]and Edit Distance [9]. LCS and Distance Edit use simple weights on Bit-parallelism computing. General Integer Scoring is a computational score of bit-parallelism for integer weights Loving [10]. Loving assumes an integer weight to represent the same pair of characters. By using the Loving approach, this paper proposes a computational scheme of Bit-parallelism scores with multi integer weights. This is an approach to alignment needs with a more complex weight such as protein.

The basic material for bit-parallelism computing in general and the general integer scoring we present in the chapter 2 Solution ideas and mathematical formulations which form the basis for compiling computational algorithms are presented in the chapter 3, the implementation for simple case into pseudocode is in the chapter 4, the result in the last chapter.

2. Bit-Parallelism & General Integer Scoring

A. Bit-Parallelism

Bit-parallelism was first developed for the exact search for sub-simple sub-string patterns Baeza, Gonet [11]. Whereas for approximate string-matching needs with a maximum error k, with the concept of finite automaton, with the regular expression Wu Mamber. The value of k is calculated from the number of substituted operations or inserted / deleted to convert one string to another. The representation of the score matrix in the form of neighboring matrix cell differences is proposed by Myer [12], computation is proposed in an iterative column for simple problem string matching. Navaro [1] proposes integer weights for each substitution condition, insert or delete to find the regular expression occurrence that has the maximum edit weight number k.

The implementation of Bit-parallelism in the case of Longest Common Subsequence (LCS) was proposed by Allison, Dix [13], representing the vertical difference vector of the score matrix and running algorithm iterations based on columns. As the initialization stage, the masking bit is filled with match position status, vertical difference vector in column 0 is filled with gap weight value. This algorithm uses a reduction operation for iterations per column. Chrochemore [14], with the CIPR algorithm using the principle of partial order relations to manipulate the match set (k-dominant match). LCS problems are defined as whole *k*-dominant match computing. CIPR uses the add operation as a substitute for subtract operations per column. Hyro [8] improved CIPR by limiting the seasrch area around diagonally.

The implementation of Bit-parallelism in the simple edit distance by Hyrro & Navaro (2005, Hyrro et all (2005), by making changes to Myer's proposal [12]. Iterative computed algorithms per column, using boundaries on diagonal areas applied to bit-parallelism with cut -off.

B. General Integer Scoring

General Integer Scoring applies a general bit-parallelism computing scheme to integer weights. This algorithm is proposed by Loving [10]. The algorithm assumes there are 1 integer weights for 3 conditions (match, mismatch and gap insertion / deletion). Loving identifies the dependencies of vertical differences between columns. Through the equation of the similarity function for filling the score matrix, the relationship of a vertical difference value to the vertical difference of the previous column and horizontal difference in the previous row. All possible integer values identified are then mapped in a functional table. This functional table states that there is a dependence on the appearance of an integer value from another integer value, whose appearance can be represented in a logical relationship.

The score computation algorithm is formed from the relation of the functional table. Based on the functional table, computing starts from an integer value with the least number of dependencies, namely the largest integer, then a smaller value and so on until the biggest negative value is the gap weight. There is a propagation pattern horizontally when the value of the maximum value appears. This pattern appears because the score function will tend to choose the largest value. Based on the equation obtained there is a relationship between horizontal values with vertical values. Horizontal difference values are obtained from transpose in the input sequence.

3. Desain Multi Integer Computation

A. Similarity Function

Taking into account the above case example protein sequence alignment where there is a group of protein residues that have a degree of similarity, it is interpreted that there are more than 1 positive values for some match conditions. Thus, we can consider the similarity function for match and mismatch as an R value, so that the similarity function equation becomes:

Definition 1 : If there are two sequences S_1 and S_2, of length m and n repectively, and the set of weight match/mismatch $= R$ and insertion/deletion gap $= G$, the similiarity function $D_{i,j}$ defined as

$\forall\, i \in \{i, \dots, n\}, \forall\, j \in \{1, \dots, m\}$

$$D_{i,j} = max\left(\left(D_{i,j} = D_{i-1,j-1} + R\right), \left(D_{i,j} = D_{i-1,j} + G\right), \left(D_{i,j} = D_{i,j-1} + G\right)\right) \quad (1)$$

R is the value obtained from the reference table, while G is the weight representation for the insertion of the gap. Referring to the difference equation from Myer [12], we can write the horizontal and vertical difference equations of a matrix cell D to the surrounding cells, as follows:

Horizontal difference : $\Delta H(i,j) = D(i,j) - D(i,j-1)$ (2)

Vertikal difference : $\Delta V(i,j) = D(i,j) - D(i-1,j)$

The writing of the score matrix then uses the score matrix difference cell value, then using equations 1), and 2) the equation value $\Delta V(i,j)$ is defined from its adjacent difference value.

In match or mismatch conditions, it can be illustrated as follows:

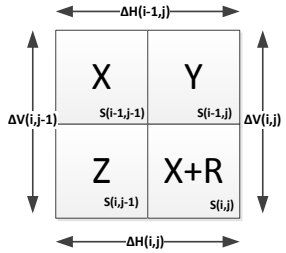


Figure 1. Illustration of the contents of the score matrix $D(i,j)$ on match or mismatch conditions

The value of the matrix cell $D(i,j)$, is determined by 3 values positions, namely $D(i-1,j-1), D(i-1,j)$ and $D(i,j-1)$. If it is assumed that the values in each position are X, Y and Z, then the match / mismatch condition will occur,

$$(X+R>Z+G)\ \&\ (X+R>Y+G) \tag{3}$$

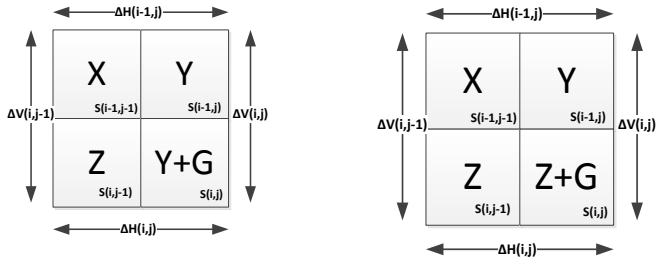
The maximum value option is in the match or mismatch condition $D(i,j)=X+R=D(i-1,j-1)+R$. Bit-parallelism utilizes iterative computation of horizontal and vertical difference values so that the vertical difference obtained from equations 2) becomes:

$$\Delta V(i,j)=R-\Delta H(i-1,j) \tag{4}$$

Whereas for requirement 3) if the match condition occurs if given algebraic manipulation (subtracted by *Y*) it will get the difference of *R* with *G* against $\Delta H(i,j-1)$ and $\Delta V(i,j-1)$

$$R-G>\Delta H(i-1,j) \text{ and } R-G>\Delta V(i,j-1) \tag{5}$$

The visualization when the score matrix is filled with the value of upper gap insertion is as follows



(a) (b)

Figure 2. Illustration of the contents of the score matrix $D(i,j)$ on upper (a) & left (b) gap insertion conditions

Figure 2 illustrates the conditions in which the gap is inserted from above (a) and the insertion gap from the left (b). At the insertion gap from above, score matrix cell $D(i,j)$ and the vertical difference contains $(i,j)=Y+G$.

$$\Delta V(i,j)=(Y+G)-Y=G. \tag{6}$$

These conditions can occur with the following conditions:

$$(Y+G>X+R)\ \&\ (Y>Z) \tag{7}$$

by modifying equation (7), $(Y+G>X+R)$ by reducing each side by X, then moving the side, and subtracting the equation $Y>Z$ with $D(i-1,j-1)$, then we obtained:

$$R-G<\Delta H(i-1,j) \text{ and } DH(i-1,j)>\Delta V(i,j-1) \tag{8}$$

Left gap insertion will make the score matrix cell $D(i,j)$ will contains $D(i,j)=D(i,j-1)+G$, reduce both sides by $D(i-1,j)$, then by adding $D(i-1,j-1)-D(i-1,j-1)$ causing the equation to be:

$$\Delta V(i,j)=\Delta V(i,j-1)+G-\Delta H(i-1,j) \tag{9}$$

the requirement to choose the value from $D(i,j-1)$

$$(Z+G>X+R)\ \&\ (Z>Y) \tag{10}$$

To form a horizontal or vertical difference, then the two sides of the initial equation above are reduced by X, and rearranging the side of the equation then subtracting both sides with $D(i-1,j-1)$, then we got

$$R-G<\Delta V(i,j-1)\ \text{ and } \Delta V(i,j-1)>\Delta H(i-1,j) \tag{11}$$

Summarizing all the vertical difference equations and their respective requirements, can be written as follows

$$\Delta V(i,j) = \begin{cases} R-\Delta H(i-1,j) & if\ \Big(\big((R-G)\geq \Delta H(i-1,j)\big)\&\big((R-G)\geq \Delta V(i,j-1)\big)\Big) \\ G & if\ \Big(\big((R-G)> \Delta H(i-1,j)\big)\&\big(\Delta H(i-1,j)\geq \Delta V(i,j-1)\big)\Big) \\ \Delta V(i,j-1)+G-\Delta H(i-1,j) & if\ \Big(\big(\big((R-G)> \Delta V(i,j-1)\big)\&\big(\Delta V(i,j-1)> \Delta H(i-1,j)\big)\big)\Big) \end{cases}$$

B. Mapping the Integer Value Relationship

From the resume of the equation above it is obtained that $\Delta V(i,j)$ is determined by $\Delta V(i,j-1)$, $\Delta H(i-1,j)$ and R. Furthermore, this relationship is translated in logical relations. Logic relations represent the logical relationship between its variable. Variables are the boolean status of the existence of the integer value (in binary). Variables area created for all integer value on range [maximum integer, minimum integer]. For a set of integer weights W_i , i is *1..b*, the maximum integer = $\max(W_i)-G$, and the minimum = *G*. Logic relations between the integer variables are mapped in a functional-table. Tables are formed for each W_i, so there will be *b* functional tables created. Next we solved the multi integer weight with the multi-functional tables.

Almost all table content has 3 area, except for tables of maximum weight. Unlike the single functional table, the multi-functional table combines the match area with mismatch into a diagonal area. A functional table has the left gap insertion area only if the table is not derived from the maximum weight. The insertion of the gap from the left occurs in two conditions: match, or meet a greater weight, as an impact of these two conditions there is a value propagation. Multi-functional table content has the pattern shown as follows:

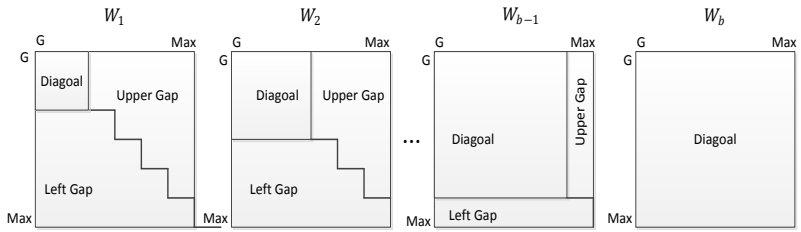


Figure 3. Pattern of multi-functional table for a set weights, with range weight $[W_1, W_b]$

The greater the weight value, the wider the diagonal area and reduce the left gap and upper gap area. At the maximum weight, there is no longer a value greater than this weight, so there is no possibility of inserting a gap from the left.

3. Desain Algorithm.

A. Operation in Single-Functional Table

The main problem of computing the bit-parallelism score is how is the relationship between the $DV(i,j-1)$ variable with $DH(i-1,j)$ in the functional table, can be computed in parallel

and iterative. We got $DH(i-1,j)$ values, from the previous row, while the $DV(i,j-1)$ values from the left shifted current DV. Identify the logical relation of an integer value can be ilustrated as the process of getting an index of the integer contain from a look-up table. We got the pair index $DV(i,j-1)$ and $DH(i-1,j)$ for each integer value position, and labeled the position as AND operation between the pair index. When the integer appear on several positions, the multi -pair this united by logical OR operations.

Iteration start with the vertical difference value, then the horizontal one. Horizontal difference value is obtained after vertical difference computing. The horizontal operation to obtain this value is like reading a functional anti-lookup table, with transpose input sequences. Computing starts from the biggest value, because it has the smallest level of dependency, or the least computational requirements, and then iteratively followed by computation of values that are less than maximum integer value.

The maximum integer propagation could not identify through the indeks of functional table, it need special treatment because the propagation even depend on the sequence position of integer value. Propagation happened when a integer x followed by the other integer less than x. To handle the maximum propagation even in parallel way, we adoting Myer's technique [12] : handling carry in addition operation. In addition operation when carry come up then the carry will continue to propagate there are no more carry found.

B. *Operation in Multi- Functional Table*

Multi-Functional Tables are operated on character pairs according to their weight values. In each row the computation will be written for the active/used functional tables, based on the weights. In comparing a character pair there is only 1 functional table that is active. The functional table used for each position is obtained by AND operation with the weight position. The location of the weights is identified at the preprocessing stage. We integrate all functional tables with logical OR operations

As in the single functional table, the representation and identification of the position of weights are carried out at the preprocessing stage. Representation and identification of weight positions adjust computational iterations. Representation of the position of weights identified as many integer values of different weights. The weight position of representation is then stored in an integer value. Some illustration of weight representation is in figure 4.

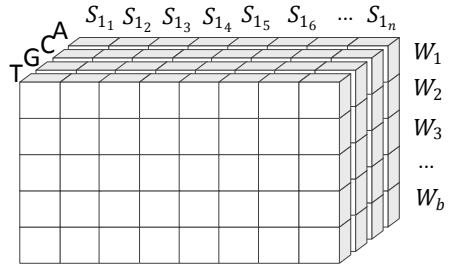


Figure 4. Illustration of the weight set representation for

Figure 4 showed when there are only four possibility character that is used in a sequence, such as DNA character. The match positin will be store in 3 tuple : $(i, char, W_j)$, i for row identification, *char* for each possible character and W_j for the each integer in set of weights.

C. Algorithm

Algorithm for bit-parallelism using multi-integer weight constructed from the computational analysis of multi-functional tables. The logical relation between the integer value identified from all functional tables. Starting from the largest integer (maximum integer value) value formed from the set of weights: $\max(W_i) - G$, up to the minimum value, namely *G*. Computing for an integer value is seen from the position of the value on each functional table. Starting from the Left Gap area, then the Diagonal area and finally the Upper Gap area.

Step 1: Maximum Vertical Difference variable $\Delta \boldsymbol{V}$ = max Identification

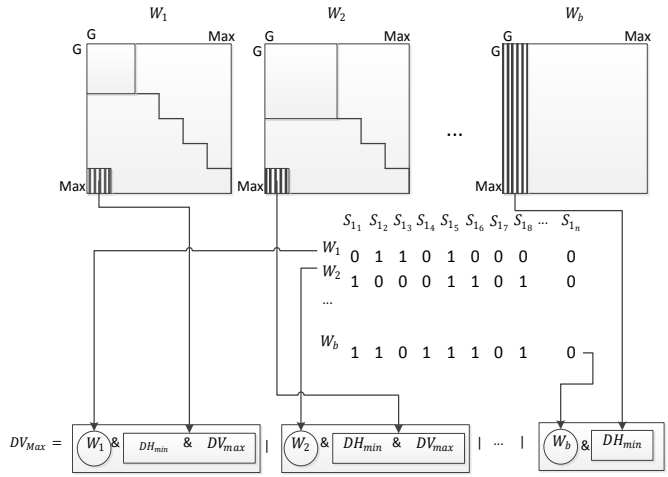


Figure 5. Illustration of the integer maximum value for all possible weights.

The maximum value is the largest integer of the functional table. Maximum value = $\max(W_i) - G$. The maximum integer is located in the most left column, the minimum integer of horizontal difference. The illustration in the figure 5. shows the location of the maximum integer value on each functional table. One functional table represents one integer weight W. The set of weights consists of $[W_1, W_b]$, where W_b is the maximum value of the set of weights. The location of the integer value is indicated in the shaded area. As in a single functional table operation, the position of the maximum integer value is identified such as finding the loop-up index of the table in each table. Then each row and column index are operated by an AND operation, and all functional table indexes are summarized by OR operations.

There are all maximum integer value in minimum horizontal differece colum at maximum weight table functional. Logical operations for maximum integer values in the functional table with the greatest weight (W_b) being as follows

$$DVmax = DHmin$$

while for the functional table with weight $< W_b$ has the following logical relations

$$DVmax = DHmin \,\&\, DVmax$$

The logical operation to integrate all functional tables is

$$DVmax = (W_1 \,|W_2|W_3 \ldots W_{B-1})\& \,(DHmin \,\&\, DVmax)|(DHmin \,\&\, W_B)$$

Propagation handling is also needed in multi-functional tables which also adopt carry handling operations in addition operations, with ADD and XOR operations. There is only one integer propagation that occurs on each row, i.e start from the the largest value position into the end of position score matrix.

$$DVmaxShift = (DVmax + DHmin)\hat{}DHmin\hat{}DVMax$$

The remaining computation of the bit position after being used to position the maximum integer value and its propagation is as follows:

$$RemainDHmin = DHmin\hat{}(DVmax \gg 1)$$

Step 2: Left Gap and Diagonal Vertical Difference Integer Identification

The next analysis is the position of the integer value in the functional table which is greater than the minimum integer value and less than maximum integer value. There are two possible positions of integer values as illustrated in the figure 6. The position in the Left Gap area only (green) and in some areas of the Left Gap is also the Diagonal area (red).

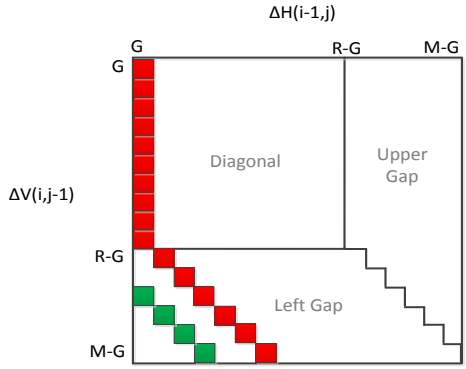


Figure 6. Illustration of the integers value < integer max for Left Gap area and Upper Gap area.

The operation logic for positions only in green occurs to the integer values represent with x, $(\min(W_i) - G) < x < \max(W_i - G)$, where i represents for all integer weight. The logic relations for only Left Gap area in a functional table are as follows

$$DV_x = \left((DV_{max}\ \&DH_{min+(max-x)})\right| \left(DV_{max-1}\&\ DH_{min+(max-x-1)}\right)|DV_{max-2}\&DH_{min+(max-x-2)}| \ldots$$
$$\ldots |(DV_x \& DH_{min})$$

while for red computing location consists of green computing by adding logical OR operations for all $DV_{(R-G)}$ until DV_{min}, so that the complete computing becomes

$$DV_x = \left((DV_{max}\ \&DH_{min+(max-x)})\right| \ldots$$
$$\ldots\ |(DV_{R-G+1}\&DH_{(R-G+1)-(x-min)})|((DV_{R-G}| \ldots |DV_{min})\&DH_{((R-G+1)-(x-min\,))+1}$$

Each functional table has differerent green and red area pattern. Computation of an integer x contruct by joining all tabel functional with logical operation OR. Illustrations for all functional tables are as figure 7.

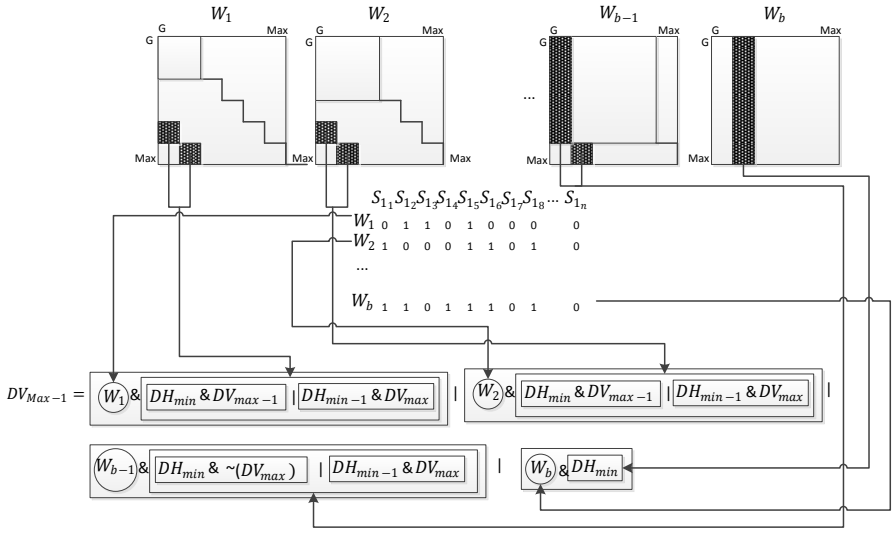


Figure 7. Illustration of the integers value < integer max for Left Gap area and Upper Gap area for all functional table.

All computations of vertical difference variables require the form of $DV(i, j-1)$ so that the variable content is shifted one bit to the left.

$DVxShift = ((DVx \ll 1) + RemainDHmin)\text{^}RemainDHmin$

Handling the propagation of values is applied to each integer value $(W_i - gap\_insertion\_weight), i > 0$, because the propagation will occur when this positive integer value appears in a row computing i iteration.

Step 3: minimum vertical difference $\Delta \boldsymbol{V = DVmin}$ *identification*

The last computation of the vertical difference is *DVmin*. *DVmin* is obtained from the remain of all positions greater than *DHmin*.

$$DVmin = all\_ones\text{^}(DV_{max}Shift|DV_{max-1}Shift|DV_{max-2}Shift|\dots|DV_{min+1})$$

Step 4: horizontal difference DH identification

Horizontal difference computing uses the same principle as a functional single table.

$$Group = \sim(DH_{max}|DH_{max-1}|DH_{max-2})$$
$$DH_x = (DH_{max}\ \&DV_x)|(DH_{max-1}\&DV_{x-1})|(DH_{max-2}\ \&DV_{x-2})|(Group\ \&\ DV_{x-3})$$

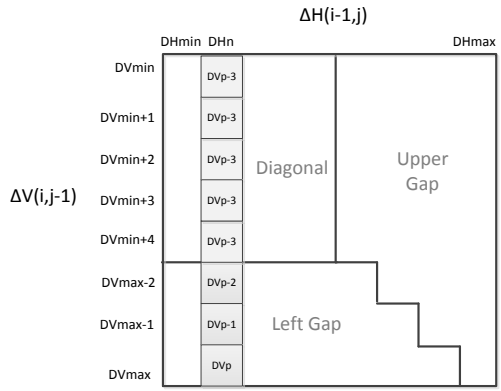


Figure 8. Illustration of Horizontal Difference computation

Take it all the analytical computation into an complete algorithm we get

Algorithm 1. Multi Integer Preprocessing ($S_1, n, W_s, ref\_weight[char_s][char_s], char_s) \rightarrow vector\_match[char_s][W_s]$

1. $iterate \leftarrow S_1$
2. $bitmask \leftarrow 0000000000000001$;
3. for $j = 1{:}n$ j do
4. $idx\_w \leftarrow ref\_weight[Iterate][char_s]$;
5. $vector\_match[char_s][idx\_w] = vector\_match[char_s][idx\_w]|bitmask$;
6. $bitmask \ll 1$; $iterate = iterate + 1$;
7. end for

Algorithm 1 is a preprocessing that produces a vector match for all possible characters in sequence ($char_s$) and for all integer weights (ref_weight). Algorithms require sequence 1 input as a reference sequence of parallel row operations and integer weights reference table. Reference Table integer weights contain weights for matching character pairs. The vector match has dimensions equal to the character list and integer weights.

Computing preprocessing requires an iteration of a number of characters in sequence reference (S1), with length n, for each possible character in the character set. Computational results are stored in an integer vector. Computing requires $O(n)$ time for preprocessing and has the same space for the same character set and integer set weights.

Algorithm 2. Multi-Integer Score Computation (S_2, m, $vector\_match$) : DV_s, DH_s

1. for $i = 1{:}\,m$ do
2. $DV_{max} \leftarrow$ Compute Maximum Vertical Diff. Integer Variable ($vector\_match$, DV_s, DH_s)
3. $DV_s \leftarrow$ Compute Left Gap Only Vertical Diff. Integer Variables ($vector\_match$, DV_s, DH_s)
4. $DV_s \leftarrow$ Compute Left Gap and Diagonal Vertical Diff. Integer Variables ($vector\_match$, DV_s, DH_s)
5. $DV_{min} \leftarrow$ Computer Minimum Vertical Diff. Integer Variable ($vector\_match$, DV_s, DH_s)
6. $DH_s \leftarrow$ Computer Horizontal Diff. Variables ($vector\_match$, DV_s, DH_s)
7. end for

Algorithm 2 produces a matrix score in the form of vertical and horizontal differences. Iterations are carried out per row as many characters as in sequence 2. The computational sequence starts from the computation of the vertical difference then computes the horizontal difference. In accordance with the relationship in the functional table, then for vertical differences stored in the form of shifted 1 bit to the left.

Computational matrix scores in the form of vertical and horizontal differences require time $O(m)$ and $O(b)$, where m is the length of sequence 2 and b is the integer range of the highest integer weight with the lowest integer weight. Logic relations algorithms are built by the given set of weights, different sets of weights will generate different algorithms.

4. Implementation

In the implementation example is given with a simple set of weights applied to DNA. According to the Nucleotide 44 reference table, DNA has a set of weights that are different match = 5, mismatch = -4 and insertion gap = -8. For this implementation we choose the simplest set weights for simplify ilustration. Given two different match weights, then a mismatch weight and a gap insertion weight. Weight of pairs (A, A) and (C, C) = 2, while (G, G) and (T, T) = 1, mismatch = -1 and insertion gap = -2.

By using the weighting attributes before, we have the integer range is [-2.4]. Vertical and horizontal integer variables formed are as much as the maximum and minimum value difference = 4 - (- 2) + 1 = 7. Integer variables for vertical differences are: *DVp4, DVp3, DVp2, DVp1, DVz, DVn1, DVn2*. Horizontal difference integer variables are: *DHp4, DHp3, DHp2, DHp1, DHz, DHn1* and *DHn2*.

The functional table formed for the above 3 weights (2.1, -1) is as follows:

	W = -1	DH(i-1,j)							W = 1	DH(i-1,j)							W = 2	DH(i-1,j)						
		-2	-1	0	1	2	3	4		-2	-1	0	1	2	3	4		-2	-1	0	1	2	3	4
	-2	1	0	-1	-2	-2	-2	-2	-2	3	2	1	0	-1	-2	-2	-2	4	3	2	1	0	-1	-2
	-1	1	0	-1	-2	-2	-2	-2	-1	3	2	1	0	-1	-2	-2	-1	4	3	2	1	0	-1	-2
	0	1	0	-1	-2	-2	-2	-2	0	3	2	1	0	-1	-2	-2	0	4	3	2	1	0	-1	-2
DV(i,j-1)	1	1	0	-1	-2	-2	-2	-2	DV(i,j-1) 1	3	2	1	0	-1	-2	-2	DV(i,j-1) 1	4	3	2	1	0	-1	-2
	2	2	1	0	-1	-2	-2	-2	2	3	2	1	0	-1	-2	-2	2	4	3	2	1	0	-1	-2
	3	3	2	1	0	-1	-2	-2	3	3	2	1	0	-1	-2	-2	3	4	3	2	1	0	-1	-2
	4	4	3	2	1	0	-1	-2	4	4	3	2	1	0	-1	-2	4	4	3	2	1	0	-1	-2

Figure 9. Illustration of multi-functional table from weight set (A,A) = (C,C) = 2, (G,G) = (T,T) = 1, mismatch = -1 and insertion gap = -2.

Computational algorithms with logical relations develop into 5 part : (1) compute maximum integer *DV*; (2) compute the *DV* Left Gap area only : integer positive 3 and integer positive 2; (3) compute *DV* Left Gap and *DV* Diagonal area : integer positive 1, zero and negative 1; (4)

minimum integer DV; (5) compute the integer DH : integer positive 4, positive 3, positive 2, positive 1, zero, negative 1 and negative 2.

Algorithm 3 : Multi-Integer $21m1m2$ Score Computation $(S_2, m, vector\_match)$: $DV_4, DV_3, DV_2, DV_1, DV_0, DV_{-1}, DV_{-2}, DH_4, DH_3, DH_2, DH_1, DH_0, DH_{-1}, DH_{-2}$

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

// (1) compute DVmax

2. $InitDV_4 = (W_{-1} \,|W_1)\& \,(DH_{-2} \,\&\, DV_4)|(DH_{-2} \,\&\, W_2)$
3. $DV_4Shift = (InitDV_4 + DH_{-2})\hat{}DH_{-2}\hat{}InitDV_4$
4. $RemainDH_{-2} = (DH_{-2} \,\&\, DV_4) \gg 1$
5. $DV_4ShiftOrMatch = DV_4Shift \mid W_2$

// (2) compute DV in Left Gap area only : DVp3 & DVp2

6. $DV_3 = (DH_{-2} \,\&\, W_2)| \left((DH_{-2} \,\&\, \sim(DV_4))\right|((DH_{-1} \,\&\, DV_4)) \,\&W_1)|$
$((DH_{-1} \,\&\, DV_4ShiftOrMatch)|(DH_{-2} \,\&\, DV_3))\&W_{-1}$

7. $DV_3Shift = (DH_{-2} + \, DV_3)\hat{}DH_{-2}\hat{}DV_3$
8. $DV_3ShiftOrMatch = DV_3Shift \mid W_1$

9. $DV_2 = (DH_z \,\&\, W_2)| \,(DH_{-1} \,\&\, W_1)|$
$((DH_z \,\&\, DV_4ShiftOrMatch)|(DH_{-1} \,\&DH_3ShiftOrMatch)|(DH_{-2} \,\&\, DV_2))\&W_{-1})$
10. $DV_2Shift = ((DV_2 \ll 1) + RemainDH_{-2})\hat{}RemainDH_{-2}$
11. $DV_2ShiftNotMatch = (\, DV_2Shift \,\&\, \sim(W_2|W_1))$

12. // (3) compute DV in Left Gap & Diagonal area : DVp1, DVz, DVn1
13. $DV_1 = (DH_1 \,\&\, W_2)| \,(DH_z \,\&\, W_1)|$
$((DH_{-2} \,\&\, \sim(DV_4ShiftOrMatch \mid DV_3ShiftOrMatch \mid DV_2Shift))|$
$(DH_1 \,\&DV_4ShiftOrMatch)| \,(DH_z \,\&\, DV_3ShiftOrMatch)|$
$(DH_{-1} \,\&\, DV_2ShiftNotMatch))\&W_{-1})$
14. $DV_1Shift = DV_1 \ll 1$

15. $DV_z = (DH_2 \,\&\, W_2)| \,(DH_1 \,\&\, W_1)|$
$((DH_{-1} \,\&\, \sim(DV_4ShiftOrMatch \mid DV_3ShiftOrMatch \mid DV_2Shift))$
$(DH_2 \,\&DV_4ShiftOrMatch)|DH_1 \,\&DV_3ShiftOrMatch|$
$(DH_z \,\&\, DV_2ShiftNotMatch))\&W_{-1}$
16. $DV_zShift = DV_z \ll 1$
17. $DV_{-1} = (DH_3 \,\&\, W_2)| \,(DH_2 \,\&\, W_1)|$
$((DH_z \,\&\, \sim(DV_4ShiftOrMatch \mid DV_3ShiftOrMatch \mid DV_2Shift))$
$(DH_3 \,\&DV_4ShiftOrMatch)|DH_2 \,\&\, DV_3ShiftOrMatch|$
$(DH_1 \,\&\, DV_2ShiftNotMatch))\&W_{-1}$
18. $DV_{-2}Shift = DV_{-1} \ll 1$

// (4) Compute DVmin : Upper Gap area : DVn2

19. $DV_{-2}Shift = (DV_4Shift \mid DV_3Shift \,|DV_2 \,Shift \mid DV_1Shift \mid DV_zShift \mid$
$DV_{-1}Shift)\hat{}(2^{word} - 1)$

// inisialisasi DH

20. $DH_2 = (DH_2 \,\&\, \sim(W_2|W_1))$
21. $DH_3OrMatch = (DH_3 \mid W_1)$
22. $DH_4OrMatch = (DH_4 \mid W_2)$
23. $DH_{-2}top1 = (DH_{-2} \,|DH_{-1}|DH_z|DH_1)$

// (5) compute DHn1, DHz, DH1, DH2, DH3, DH4

24. $DH_{-1} = (DH_3Shift \,\&\, W_2)|DH_2Shift \,\&\, W_1|\left((DV_3Shift \,\&\, DH_4OrMatch)\right|$
$(DV_2Shift \,\&\, DH_3OrMatch)|(DV_1Shift \,\&\, DH_2)(DV_z \,Shift \,\&\, DH_{-2to1})) \,\&\, W_{-1}$

25. $DH_z = (DH_2Shift \,\&\, W_2)|DH_1Shift \,\&\, W_1|\left((DV_2Shift \,\&\, DH_4OrMatch)\right|$

$(DV_1 Shift \,\&\, DH_3 OrMatch)|(DV_z Shift \,\&\, DH_2)(DV_{-1}\ Shift \,\&\, DH_{-2to1})) \,\&\, W_{-1}$

26. $DH_1 = (DH_1 Shift \,\&\, W_2)|DH_z Shift \,\&\, W_1|\big((DV_1 Shift \,\&\, DH_4 OrMatch)|$
$(DV_z Shift \,\&\, DH_3 OrMatch)|(DV_{-1} Shift \,\&\, DH_2)(DV_{-2}\ Shift \,\&\, DH_{-2to1})) \,\&\, W_{-1}$

27. $DH_2 = (DH_z Shift \,\&\, W_2)|(DH_{-1} Shift \,\&\, W_1)|$
$\big((DV_z Shift \,\&\, DH_4 OrMatch)|DV_{-1} Shift \,\&\, DH_3 OrMatch|(DV_{-2} Shift \,\&\, DH_2)\big)\& W_{-1}$

28. $DH_3 = (DH_{-1} Shift \,\&\, W_2)|(DH_{-2} Shift \,\&\, W_1)|$
$\big((DV_{-1} Shift \,\&\, DH_4 OrMatch)|(DV_{-2} Shift \,\&\, DH_3 OrMatch)\big)\& W_{-1}$

29. $DH_4 = (DH_4 OrMatch \,\&\, (DV_{-2} Shift \,\&\, W_2))|(DH_4 OrMatch \,\&\, (DH_{-2} Shift \,\&\, W_1))|$
$(DHp4OrMatch \,\&\, (DHn2Shift \,\&\, W_{-1}))$

30. $DH_{-2} = (DH_4 \mid DH_3 \mid DH_2 \mid DH_1 \mid DH_z \mid DH_{-1})\ \hat{}\ (2^{word} - 1)$

31. end for

Using the same principle, this algorithm promises to apply to cases of more complex weight sets. The more complex weight sets have more integer weights, which has an impact on the number of functional tables involved in computing. Adding a functional table means increasing the logical operation term. The integer weight range will affect the number of integer variables, the wider the integer range, the more integer variables will be used.

Result and Analysis

We compare the algorithm with the classical DP by Needleman Wunch [15], for score computation result and the running time. Using the same DNA data set as the data set used by Loving [10], data sets are regenerated to produce varying sequence lengths (20, 30, 40, 50 and 60). Algorithms implemented in the C programming language environment. The result is the average of the 30 time running from each algorithm.

For all sequence size, the Multi-Integer algorithm give the same score computational results with the DP score matrix. The average of multi-integer score running time for each sequence size as showed at figure 9.

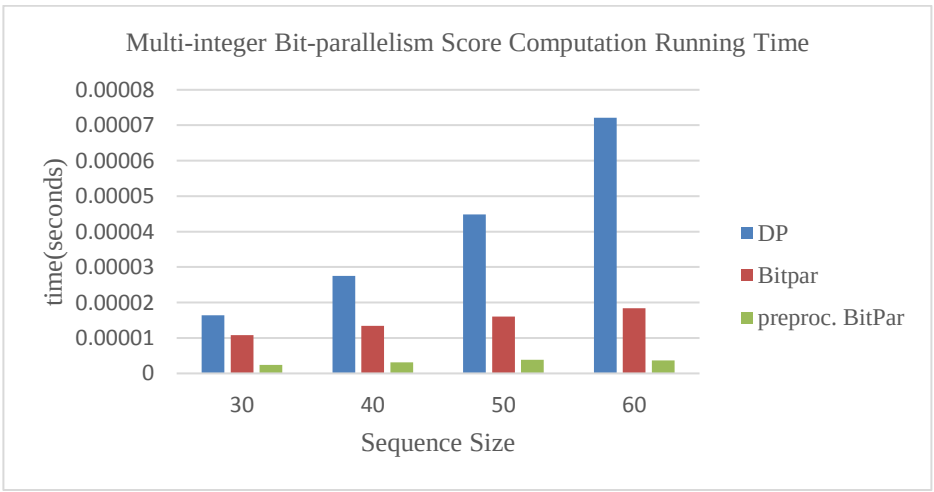
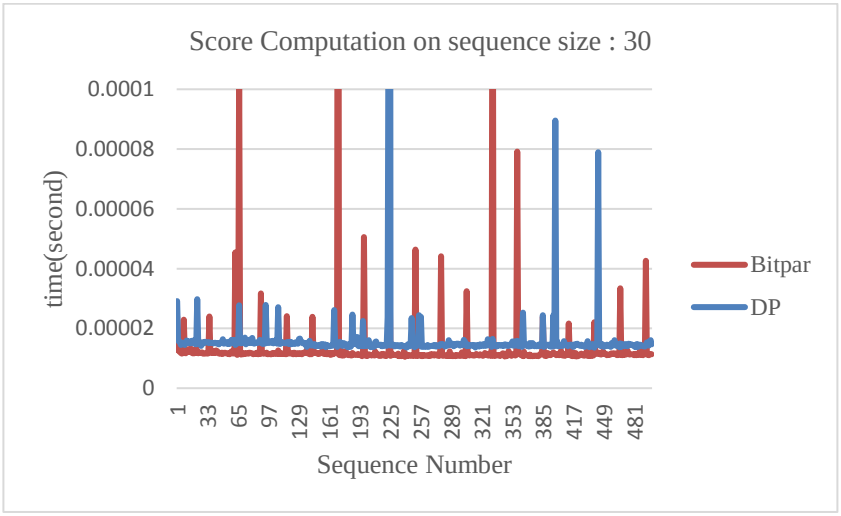


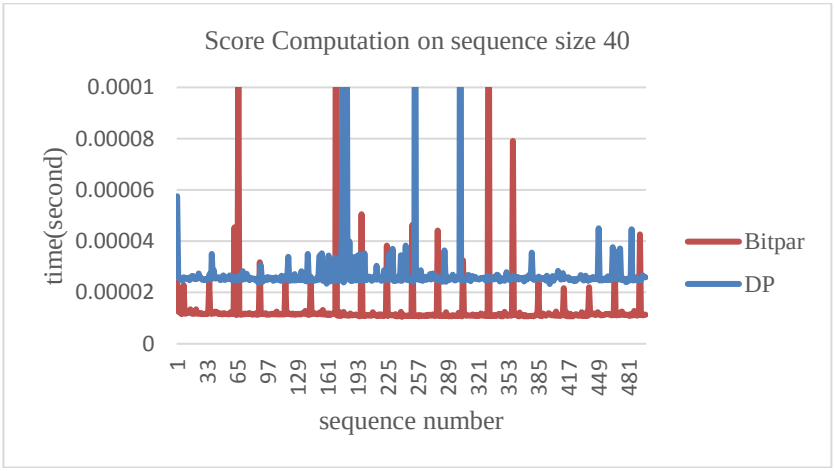
Figure 10. Multi-integer Score Computation Running time compare to Classical DP Score computation running time

The increasing sequence size give $O(mn)$ running process in classical DP, but only $O(m)$ on multi-integer bit-parallelism score computation. Multi-integer need not much time in preprocessing stage.

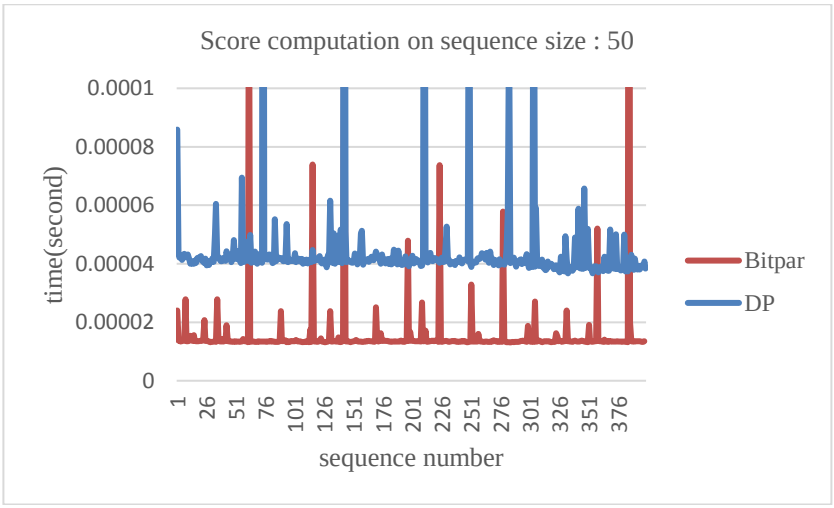
The detail of running time in each sequence size described on the figure 11. The greater sequence size give longer distance between the two graphics. For average sequence pair, multi-integer score computation give slow increasing time. The fluctuation on the running time showed the behind proses on virtual machine C environment Both of the DP and Bit-parallelism have the close variance result of time computation (in 10^{-9}).



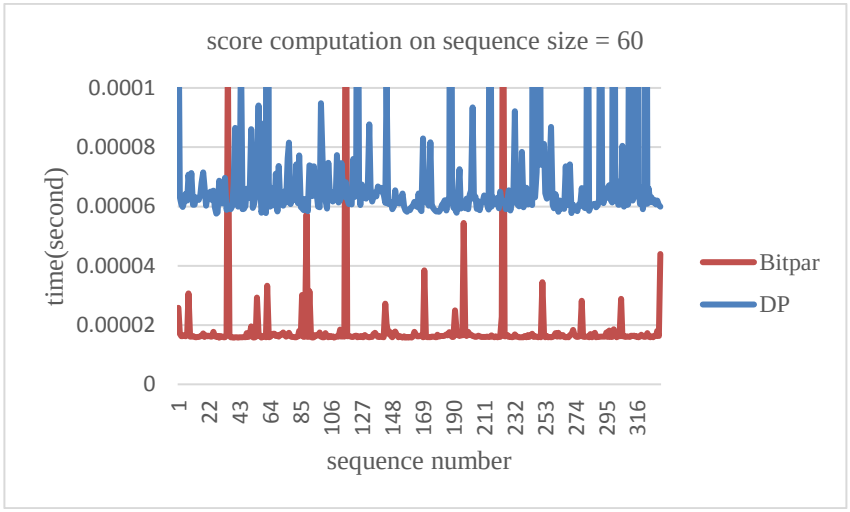
(a)



(b)



(c)



(d)

Figure 11. Detail Multi-integer Score Computation Running time compare to Classical DP Score computation running time (a) sequence size = 30, (b) sequence size = 40, (c)sequence size = 50, and (d)sequence size = 60,

5. Conclusion

The algorithm applies multi integer weights by combining functional-tables. The algorithm succeeds in providing the same score matrix with the classic DP score matrix. Although this algorithm less flexible, because one code is created for a set weight, this algorithms give faster computation result in $O(m)$ in time and only need O(b) space requirements, with b according to the set range of weights.

General algorithm of multi integer score computation promise computational logic relations with more complex weights set (need more difference integer weights), with still maintained in $O(m)$ time computation. By adding an algorithm for recovery alignment, this method has the potential to be an alternative to faster computing scores on sequence alignment tools.This method also promises faster computation for other optimization problems that use the DP algorithm with multi integer weights.

6. Reference

[1]. G. Navarro, "A guided tour to approximate string matching," *ACM computing surveys (CSUR)*, vol. 33, pp. 31-88, 2001.

[2]. E. Pertsemlidis dan J. W. F. Iii, "Tutorial Having a BLAST with bioinformatics (and avoiding BLASTphemy)," 2001

[3]. W. R. Pearson, "Finding protein and nucleotide similarities with FASTA," *Current protocols in bioinformatics*, vol. 53, pp. 3-9, 2016.

[4]. T. P. Walsh, C. Webber, S. Searle, S. S. Sturrock dan G. J. Barton, "SCANPS: a web server for iterative protein sequence database searching by dynamic programing, with display in a hierarchical SCOP browser," *Nucleic acids research*, vol. 36, pp. W25--W29, 2008

[5]. T. Rognes, "ParAlign: a parallel sequence alignment algorithm for rapid and sensitive database searches," *Nucleic acids research*, vol. 29, pp. 1647-1652, 2001.

[6]. E. F. O. Sandes dan A. C. Melo, "CUDAlign: using GPU to accelerate the comparison of megabase genomic sequences," dalam *ACM Sigplan Notices*, 2010.

[7]. C.-L. Hung, Y.-S. Lin, C.-Y. Lin, Y.-C. Chung dan Y.-F. Chung, "CUDA ClustalW: An efficient parallel algorithm for progressive multiple sequence alignment on Multi-GPUs," *Computational Biology and Chemistry*, vol. 58, pp. 62-68, 2015.

[8]. H. Hyyr{\"o}, "Bit-parallel LCS-length computation revisited," dalam *Proc. 15th Australasian Workshop on Combinatorial Algorithms (AWOCA 2004)*, 2004

[9]. H. Hyyr{\"o}, "Explaining and extending the bit-parallel approximate string matching algorithm of Myers," 2001.

[10]. J. Loving, Y. Hernandez dan G. Benson, "BitPAl: a bit-parallel, general integer-scoring sequence alignment algorithm," *Bioinformatics*, vol. 30, no. 22, pp. 3166-3173, 2014..

[11]. R. Baeza-Yates dan G. H. Gonnet, "A new approach to text searching," *Communications of the ACM*, vol. 35, pp. 74-82, 1992.

[12]. G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming," *Journal of the ACM (JACM)*, vol. 46, no. 3, pp. 395-415, 1999

[13]. L. Allison dan T. I. Dix, "A bit-string longest-common-subsequence algorithm," *Information Processing Letters*, vol. 23, pp. 305-310, 1986.

[14]. M. Crochemore, C. S. Iliopoulos, Y. J. Pinzon dan J. F. Reid, "A fast and practical bit-vector algorithm for the longest common subsequence problem," *Information Processing Letters*, vol. 80, pp. 279-285, 2001

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

[16]. T. F. Smith, M. S. Waterman dan W. M. Fitch, "Comparative biosequence metrics," *Journal of Molecular Evolution*, vol. 18, pp. 38-46, 1981.

[17]. J. Xiong, Essential bioinformatics, Cambridge University Press, 2006.

[18]. T. P. a. W. C. a. S. S. a. S. S. S. a. B. G. J. Walsh, "SCANPS: a web server for iterative protein sequence database searching by dynamic programing, with display in a hierarchical SCOP browser," *Nucleic acids research*, pp. 25-29, 2008.

[19]. E. Ukkonen, "Algorithms for approximate string matching," *Information and Control*, vol. 64, pp. 100-118, 1985.

[20]. T. F. Smith dan M. S. Waterman, "Comparison of biosequences," *Advances in Applied mathematics*, vol. 2, no. 4, pp. 482-489, 1981.

[21]. M. S. Rosenberg, Sequence alignment: methods, models, concepts, and strategies, Univ of California Press, 2009.

[22]. D. Kirk dan others, "NVIDIA CUDA Software and GPU Parallel Computing Architecture," dalam *ISMM*, 2007.

Setyorini, graduated a Bachelor degree in Informatics from Sekolah Tinggi Teknologi Telkom, Bandung in 2000 and Master degree in Computer Engineering from Institut Teknologi Bandung in 2004. Currently she is a doctoral student at the School of Electrical Engineering and Informatics, Institut Teknologi Bandung.

Kuspriyanto is a professor at the School of Electrical Engineering and Informatics, Institut Teknologi Bandung. He received his bachelor degree in Electrical Engineering from Institut Teknologi Bandung in 1974, D.E.A (1979) and Ph.D. (1981) in Automatic System from USTL, France. His field of interest includes Computer System, Computer Architecture, and Real Time Systems.

Dwi H. Widyantoro, is a professor at the School of Electrical Engineering and Informatics, Institut Teknologi Bandung. He graduated from his Bachelor degree program in Computer Science ITB; his Master and Doctoral program in Computer Science from Texas A&M University, College Station, TX, USA. His field of interest include Machine Learning.

Adi Pancoro, graduated from his Bachelor degree program in Biology Departement ITB; and his PhD degree from Departement Genetics and Biochemistry, Newcastle upon Tyne, England. He is now a Senior Lecturer for Undergraduate and Master graduate program in School of Life Science and Technology, ITB. His reseach interests includes Biotechnology and Bioinformatics.