1. INTRODUCTION

Sequence alignment refers to the search of similarity regions within biological sequences of DNA, RNA, or proteins. Besides the biological significance and interpretation of the results, the main problem of sequence alignment from the computer science point of view is the very large number of residue-to-residue comparisons that are needed when searching for similarities.

Thus, a computer program can therefore intensively search for regions of similarity between sequences to detect such relationships.

Alignment could be global or local. Global alignment is to find the best match between the entire sequences, whereas local alignment must find the best match between certain regions of the sequences.

Global alignment algorithms such as Needleman–Wunsch (NW) [1], which attempt to align every residue in every sequence, are most useful when the sequences in the query set are similar and of roughly equal size. Local alignments instead, like in Smith–Waterman (SW) [2], are more often used for dissimilar sequences that are suspected to contain regions of similarity within their larger sequence context.

Fig. 1 shows a global and a local alignment of the same two sequences. It shows that if sequences are not sufficiently similar, a global alignment tends to spread gaps that hide possible regions of similarity. Residue mismatches are called...
substitutions (mutations in genetic terminology), and the character “-” denotes gaps that may be insertions or deletions depending on the point of view.

A. Substitution Matrix
Biologists have learned that some mutations are more likely than others (e.g., a DNA mutation from cytosine to adenine is more common than cytosine to guanine) and they need alignment algorithms to reflect this property. For this purpose, substitution matrices have been built using statistical data from known sequences and mutations. Fig. 2 shows the BLOSUM62 [3], a common substitution matrix for amino acids alignments.

Definition 1: A substitution matrix \( (\Sigma) \) on an alphabet \( \Sigma = \{a_1, a_2, \ldots, a_n\} \) has \( n \times n \) entries, where each entry \((i,j)\) assigns a score for a substitution of the letter \( a_i \) by the letter \( a_j \) in an alignment.

The elements on the main diagonal have the highest values to encourage matching of identical residues in alignment algorithms. Amino acids are divided into colored groups according to the chemistry of the side group.

B. Gap Penalty
Besides having substitution matrices for mutations, it is also desirable to score insertions and deletions gaps differently. First, to avoid having gaps all over the alignment, gaps have to be given penalties, just like unmatched amino acids. This penalty cannot be derived from the database alignments used to create the substitution matrices such as BLOSUM since these matrices were derived from ungapped alignments. The score given to an insertion/deletion is commonly called a gap penalty.

There are two schemes used, namely, linear gap \( (g) \) penalty and affine gap penalty. Having only one score for any gap inserted is called a linear. However, insertions and deletions often involve a longer stretch of sequence in a single event. For this reason, two different gap penalties are usually included in the alignment algorithms: One penalty for having a gap at all (gap opening penalty \( (g0) \)) and another smaller penalty for extending already opened gaps \( (g) \). This is called an affine and is actually a compromise between the assumptions that the insertion or deletion is created by one or more events [4].

Definition 2: Given a sequence \( S \) over the alphabet \( \Sigma \) of length \( L \), a sequence \( S' \) of length \( L' \) over \( \Sigma \cup \{\text{“-”}\} \) is called a gapped sequence of \( S \) if \( L' \geq L \) and there exist a transformation \( T(S) = S' \) such that:
1. \( \forall 1 \leq i \leq L; \exists 1 \leq j \leq L; S(i) = S'(j) \)
2. If \( S'(p) = \text{“-”} \) and \( S(q) = S'(j); \) and \( i < j \) then \( p < q \).

2. PAIR-WISE ALIGNMENT
Pair-wise sequence alignment is defined as an alignment of two sequences to determine how similar they are. In most sequence similarity calculations, a similarity score is inferred from the alignment. Gap insertions are allowed until the resulting sequences are of the same size, and the alignment must obey the restriction that gaps cannot appear in the same position in both sequences. This score is determined based on a substitution matrix and specific penalties for the insertions and deletions gaps.

In the following, the two most common pair-wise alignment algorithms used to compute the similarity matrix \( H \) for each a pair of sequences \( S \) and \( T \) with their length \( L_S \) and \( L_T \), respectively, are explained in detail.

Definition 3: Given a pair of sequences \( S \) and \( T \) over the alphabet \( \Sigma \) with their lengths \( L_S \) and \( L_T \), respectively. Let \( S^g \) and \( T^g \) be two-gapped sequences with lengths \( L_S^g \) and \( L_T^g \), respectively. A pair-wise sequence alignment of \( S \) and \( T \) is defined to be a matrix \( M \) of size \( (2 \times n) \) with \( n = \max (L_S^g, L_T^g) \) with the following properties: \( \forall 1 \leq i \leq n \).
The problem of pair-wise alignment \( S \) and \( T \) is to find an alignment \( M_o(S,T) \) with optimal score \( (M_o(S,T)) \), that is,

\[
\text{Score} (M(S,T)) \leq \text{Score} (M_o(S,T)) \quad \text{for all alignments} \quad M(S,T).
\]

**A. NW Algorithm**

The NW algorithm [1], introduced in 1970, as the first dynamic programming tool to compute a global alignment for any pair of biological sequences. The NW algorithm achieves its goal by going through the following three phases:

- **The initialization phase:** Initiates the \( H(i,0) \) matrix element by 0. The first row and column are initialized with the costs of gaps of lengths \( s \) and \( t \).

  - i.e., \( H(s,0) = g.s \) and \( H(0,t) = g.t \) \( \forall 1 \leq s \leq L_s, 1 \leq t \leq L_t; g \) is a gap penalty.

- **The score computation phase:** Computes all other values of matrix \( H(s,t) \) using one of the following recursive formulas:

  \[
  H_L(s,t) = \max \left\{ \begin{array}{ll}
  H_L(s-1,t-1) + \text{sbt}\left(S(s),T(t)\right), \\
  H_L(s-1,t) + g, \\
  H_L(s,t-1) + g
  \end{array} \right.
  \]

  or

  \[
  H_A(s,t) = \max \left\{ \begin{array}{ll}
  H_A(s-1,t-1) + \text{sbt}\left(S(s),T(t)\right), \\
  E(s,t) + g_s, \\
  F(s,t) + g_e
  \end{array} \right.
  \]

  or

  \[
  E(s,t) = \max \left\{ \begin{array}{ll}
  H_A(s,t-1) + g_o, \\
  E(s,t-1) + g_e
  \end{array} \right.
  \]

  \[
  F(s,t) = \max \left\{ \begin{array}{ll}
  H_A(s-1,t) + g_o, \\
  E(s,t) + g_e
  \end{array} \right.
  \]

Where, \( \text{sbt} \) is a substitution matrix and \( (g, g_o, \text{and } g_e) \) are the gaps penalty.

- **The trace back phase:** Recovers the alignment by tracing back the path starting from the last element \( H(L_s+1; L_t+1) \).

**A. SW Algorithm**

From the evolution perspective, two-related sequences could evolve independently with many independent mutations lowering the similarity between the sequences. Aligning the sequences with noised information often fails to produce a biologically meaningful alignment. In these cases, the local alignment, proposed by Smith and Waterman [2], identifies the longest segment pair that yields the best alignment score is more preferable. In the SW’s algorithm, the longest segment pair between two aligning sequences that yield the optimal alignment is identified by comparing all possible segments of all lengths between the two sequences through dynamic programming technique.

The main difference between this technique and NW’s is that negative scores are set to zeroes. This modification produces an alignment score matrix with positive scores. Thus, the backtracking procedure of the algorithm starts at the highest positive score cell and proceeds until it encounters a cell with zero score. The longest segment pair identified in these backtracking steps is the optimal scored local alignment of the two sequences.

The similarity matrix score \( H \) is filled using one of the following recurrence formula:
\[
E(s,t) = \max \left\{ H_A(s,t-1) + g_o, E(s,t-1) + g_e \right\}
\]

\[
F(s,t) = \max \left\{ H_A(s-1,t) + g_o, F(s-1,t) + g_e \right\}
\]

Where, \( s_{bt} \) is a substitution matrix and \((g, g_o, and g_e)\) are the gaps penalty.

Fig. 3 shows the example of calculating the pair-wise local alignment \( S = \{ATCTCGTATGAT\} \) and \( T = \{GTCTATCAC\} \) using the SW algorithm.

The similarity matrix \( H \) is shown for:

\[
g = -1, \quad s_{bt}(S(t),T(t)) = \begin{cases} 
2 & \text{if } S(t) = T(t) \\
-1 & \text{Otherwise}
\end{cases}
\]

From the highest score \( H(8,11) = 10 \), a procedure of traceback carries out the corresponding alignment as shown in Fig. 3.

3. ALIGNMENT FOR MULTIPLE SEQUENCE

Multiple sequence alignment (MSA) refers to the alignment of more than two biological sequences (DNA, RNA, or protein). It is considered as an extension of pair-wise sequence alignment as discussed in the previous section. It helps in many criteria such as identifying diagnostic patterns or motif to characterize protein families, demonstrating homology between new sequences and existing families of sequences.

MSA is NP-complete problem [5] since the computational cost grows exponentially with the expansion of biological datasets. This leads to the development of many algorithms aiming at reaching the most accurate and efficient alignment. Most commonly used algorithms are classified into two categories, progressive and iterative.

Recent studies have shown significant progress in enhancing the quality, accuracy, and speed of MSA tools. However, the big dataset of sequences of biologically relevant length can be difficult and time-consuming to align. Thus, many MSA tools have been proposed. In this section, only important MSA paradigms are introduced. They are used as a base for various MSA distinguished tools [6]. A progressive method most widely used MSA tools utilize the progressive method that was first introduced in 1987 [7]. For aligning \( N \) sequences, it goes through the following three main stages (Fig. 4):

- Stage 1: Generates all possible \( N(N-1)/2 \) pair-wise sequence alignment to construct a distance matrix computing the similarity between of each two sequences
- Stage 2: Creates a guide-tree using all the pair-wise distances using a clustering method such as unweighted pair-group method with arithmetic [8] or neighbor-joining [9]
- Stage 3: Builds-up the final multiple alignments by the progressive inclusion of the \( N \) sequences alignment based on the range given by the guide tree.

Stage 1 is calculated using the match between the residues of the two sequences found by the local alignment. The number of exact match is computed by counting the identical residues appearing in the same column in the local alignment excluding gaps, using the equation [10]:

\[
\text{Dist}(S, T) = 1 - \frac{\text{nid}(S, T)}{\min\{L_s, L_t\}}
\]  
(2)

Where, \( \text{nid}(S, T) \) indicates the number of exact matches using SW algorithm to align \( S \) and \( T \). For instance, the \( \text{nid} \)-value in Fig. 3 is 6. Actually, this method will run-out storing the similarity matrix \( H \), which is not practical for the datasets with long sequences.
Liu et al. [11] presented new recurrence relations equations (3) and (4) for the nid-value calculation that is compatible for parallel systems such as GPUs. They facilitate nid-computations without calculation of the actual traceback, to reduce the storage space using the matrices \( N_L(s,t) \) and \( N_A(s,t) \) for linear and affine gap penalty, respectively. The computations of these matrices are given by following recurrences:

\[
N_L(s,t) = \begin{cases} 
0 & \text{if } H_L(s,t) = 0 \\
N_L(s-1,t-1) & \text{if } H_L(s,t) = H_L(s-1,t-1) \\
+m(s,t) & \text{if } H_L(s,t) = H_L(s-1,t) + \text{gap(s,t)} \\
N_L(s,t-1) & \text{if } H_L(s-1,t) = H_L(s,t) + \text{gap(s,t)} \\
N_L(s-1,t) & \text{if } H_L(s,t-1) = H_L(s,t) + \text{gap(s,t)} 
\end{cases}
\]

where,

\[
m(s,t) = \begin{cases} 
1 & \text{if } s = T(t) \\
0 & \text{otherwise} 
\end{cases}
\]

For linear gap penalty, and for affine gap penalty, we use:

\[
N_A(s,t) = \begin{cases} 
0 & \text{if } H_A(s,t) = 0 \\
N_A(s-1,t-1) & \text{if } H_A(s,t) = H_A(s-1,t-1) \\
+m(s,t) & \text{if } H_A(s,t) = H_A(s-1,t) + \text{gap(s,t)} \\
N_E(s,t-1) & \text{if } H_A(s,t) = E(s,t) \\
N_E(s-1,t) & \text{if } H_A(s,t) = F(s,t) 
\end{cases}
\]

where,

\[
m(s,t) = \begin{cases} 
1 & \text{if } s = T(t) \\
0 & \text{otherwise} 
\end{cases}
\]

4. PROPOSED METHOD

This section propose vectorized and parallelized algorithm for computing the sequence alignment. The aim of this work is switching of all the previous matrices to vectors and computing it by parallel. The main goal for it is to reduce used storage and speed-up runtime, for aligning long biological sequences fast.

The proposed algorithm will implement on the optimal local alignment (SW) algorithm because of the following features:

- It has been trusted by biologists for almost two decades with quality that is still comparable to more recent algorithms
- Its alignment results are similar to biologists expectations
- It is relatively fast, simple, understandable, and provides fairly good alignments across a diverse range of sequence types
- It has highly cited aligner, especially for big dataset of sequences as mentioned in recent studies [12]-[15].

SW algorithm [2] compares two sequences by computing a distance that represents the minimal cost of transforming one segment into another, with respect to the given scoring system.

As previously mentioned, there are two approaches to compute SW algorithm, based-on-gap penalty, linear, and affine gap penalties. Gap penalties prefer more continuous
gaps to opening new gaps. Therefore, it encourages that gaps occur in loop regions instead of in highly structured regions.

The background biological meaning for this is that biologically divergence is often less likely in highly structured regions, which are commonly very important to the fold and function of a protein. In this paper, the affine gap penalty will be used.

A. Vectorization Approach

Vectorizing all matrices presented in the previous section is the main contribution of the proposed method. It was used when computing the aligning sequences of long lengths, aiming at accelerating computations, and used less space. This approach was based on the calculation of each elements of antidiagonal \( D \) in the similarity matrix (\( H \)) with affine gap penalty is based only on the computed elements of the four antidiagonals; two from (\( H \)) matrix with one from both \( E \) and \( F \) matrices.

This postulate that just one vector \( D \) for current antidiagonal, with six buffers previously computed \( D_1, D_2, D_{12}, D_{123}, D_{1234}, \) and \( D_{12345} \) are enough to calculate the elements of \( D \). After the newly \( D \) is computed, \( D_1 \) is replaced by \( D_2, D_2 \) is replaced by \( D_3, D_3 \) is replaced by \( D_4, D_4 \) is replaced by \( D_5 \), and \( D_5 \) is replaced by \( D_{123} \).

In the subsequent iteration, this cyclic method is used to replace the six buffers \( D_1, D_2, D_3, D_4, D_5, \) and \( D_{12345} \). The values of all cells in \( D \) are computed in terms of its diagonal neighbor stored at \( D_5 \), with its left and upper neighbors stored at \( D_4 \) in addition the cells in \( D_3, D_2, \) and \( D_{123} \), and the maximum value is selected indicating the highest score.

Fig. 5 shows the vectorization approach when aligning pair of sequences \( S = \{GCTACTCA\} \) and \( T = \{TATGGATT\} \) with their lengths are 9 and 12, respectively. It illustrates the calculation of the elements of \( H_4 \) using affine gap (\( g_0 = 7, g_e = -1 \)) and a substitution cost of (2) if the characters are match and (−1) otherwise, and how it is replaced by \( D_i \), using \( D_1, D_2, D_3, D_4, D_5, \) and \( D_{123} \).

This figure shows the dependence relationship of the elements of the matrices, which is visualized by considering its antidiagonals \( D_1, D_2, D_{12}, D_{123}, \) and \( D_{1234} \) dependencies. It is clear that antidiagonal \( D \) in iteration \( i = 9 \) computations depend on the four previously computed antidiagonals \( D_1, D_2, D_{12}, \) and \( D_{123} \). Therefore, all other computed antidiagonals can be neglected.

Furthermore, it shows the dependence relationship of cell \( D(5) \) with its left neighbor \( D_1(5) = -9, \) upper neighbor \( D_{12}(4) = -8, \) and upper left neighbor \( D_{123}(4) = 5, \) Where \( D_{1234}(5) + g_0 \) is a maximum value of \( (D_{12}(5) + g_e \) and \( D_{123}(5) + g_0); \) \( D_{1234}(4) + g_0 \) is a maximum value of \( (D_{123}(4) + g_e \) and \( D_{1234}(4) + g_0). \)

Using this way, all elements along vector \( D \) are computed in parallel from all elements in vectors \( D_1, D_2, D_{12}, \) and \( D_{123}. \)

To verify these postulates, authors have proposed the following new recurrence and theorem.

Theorem: The pair-wise local alignment of the sequences \( S \) and \( T \), with an affine gap penalty (\( g_0 \) for opening gap and \( g_e \) for extending gap), substitution matrix (\( stb \)), in iteration (\( i \)), and with element (\( j \)), the equation:

\[
D' (j) = H_A (j, i-j + 1)
\]

Gives a vectorization \( D' \) of the matrix \( H_A \) and the following relations hold:

\[
N(i) = H_A (j, i-j + 1)
\]

\[
\text{max}(S, T) = \max_i N(i)
\]

Where \( \text{max}(S, T) \leq i \leq \text{min}(S + 1, T + 1) \) and \( (i) \) indicates the position of the maximum value in the vectors \( D' \).

Proof: From Equation (1), we get:

\[
H_A (j, i-j + 1) = \max \{T(i-j+1), E(j, i-j+1), F(j, i-j+1) \}
\]

\[
E(j, i-j+1) = \max \{H_A (j, i-j) + g_0, E(j, i-j) + g_e \}
\]

\[
F(j, i-j+1) = \max \{H_A (j-1, i-j) + g_0, F(j-1, i-j+1) + g_e \}
\]

and from Equation (5), we get:

\[
D'(j) = \max \{0, D'^{(i-2)}(j-1) + stb(S(j), T(i-j+1)), D_E(j), D_F(j) \}
\]
Fig. 5. Relation between $H$, $D$, $D_0$, $D_1$, $D_2$, and $D_E$.

$$D_E^i(j) = \begin{cases} 
D_{E}^{i-1}(j-1) + g_o, \\
D_{E}^{i-1}(j) + g_E
\end{cases}$$

$$D'_E(j) = \begin{cases} 
D_{E'}^{i-1}(j) + g_o, \\
D_{E'}^{i-1}(j-1) + g_E
\end{cases}$$
Since $D_{i,j}^k$ is computed in the previous iteration of $D_{E}^k$, $D_{E}^{k-1}$, $D_{F}^{k}$, $D_{E}^{k-1}$, and $D_{F}^{k-1}$, that is, in one iteration before $D_{E}^k$, $D_{E}^{k-1}$, and $D_{F}^{k}$, hence $D_{2}^k$ is computed in the second iterations before $D_{F}^{k}$, let us denote $D_{1}^k$, $D_{2}^k$, $D_{E}^{k-1}$, and $D_{F}^{k-1}$ by $D_{1}^k$, $D_{2}^k$, $D_{E}^{k-1}$, and $D_{F}^{k-1}$, respectively. Then, we get:

$$D_{E}^{0}(j) = \begin{cases} 0, & \text{if } j = 1 \\ D_{E}^{k}(j-1) + \text{sub}(S(j), T(i - j + 1)), & \text{otherwise} \end{cases}$$

Then, we get:

$$N_{E}^{0}(j) = \begin{cases} 0, & \text{if } j = 1 \\ N_{E}^{k}(j-1) + \text{sub}(S(j), T(i - j + 1)), & \text{otherwise} \end{cases}$$

(7)

To proof of $N_{D}^{i}(j)$ let the Equation (6), gives the vectorization $N_{A}(i,t)$:

$$N_{A}(j,i-j+1) = \begin{cases} 0, & \text{if } H(j,i-j+1) = 0 \\ N_{A}(j-1,i-j), & \text{if } H(j,i-j) = H(j-1,i-j) \\ +m(j,i-j+1), & \text{otherwise} \\ N_{E}(j,i-j+1), & \text{if } H(j,i-j+1) = E(j,i-j+1) \\ N_{E}(j,i-j+1), & \text{if } H(j,i-j+1) = F(j,i-j+1) \end{cases}$$

$$N_{D}^{0}(j) = \begin{cases} 0, & \text{if } j = 1 \\ N_{D}^{i}(j-1) + \text{sub}(S(j), T(i - j + 1)), & \text{otherwise} \end{cases}$$

After vectorizing the matrix $N_{A}$ as shown in Fig. 5.

$$N_{D}^{0}(j) = \begin{cases} 0, & \text{if } D^{k}(j) = 0 \\ N_{D}^{k}(j-1), & \text{if } D^{k}(j) = D^{k}(j-1) \end{cases}$$

$$N_{E}^{0}(j) = \begin{cases} 0, & \text{if } j = 1 \\ N_{E}^{k}(j-1) + \text{sub}(S(j), T(i - j + 1)), & \text{otherwise} \end{cases}$$

$$N_{E}^{0}(j) = \begin{cases} 0, & \text{if } j = 1 \\ N_{E}^{k}(j-1) + \text{sub}(S(j), T(i - j + 1)), & \text{otherwise} \end{cases}$$

(8)

Where,

$$m(j) = \begin{cases} 1, & \text{if } S(j) = T(i - j + 1) \\ 0, & \text{otherwise} \end{cases}$$

We now show that for a given $i$, $N_{i}(j)$ is equal to the number of exact matches in the optimal $(i)$ suffix alignment.

Case 1: $D^{k}(j) = 0$. The alignment is empty. Hence, $N_{i}(j) = 0$.

Case 2: $D^{k}(j-1) + \text{sub}(S(j), T(i - j + 1))$. The alignment ends with $S(j)$ aligned to $T(i-j+1)$, which contributes $m(\text{sub}(S(j), T(i-j+1)))$ to the $nidd$-value. The residual number is than equal to the $nidd$-value got in the optimal $D^{k}$ suffix alignment. Hence,

$$N_{i}(j) = N_{i}(j-1) + m(\text{sub}(S(j), T(i-j+1)))$$

Case 3: $D^{k}(j) = D^{k}(j-1) + \text{sub}(S(j), T(i-j+1))$. The alignment ends with $S(j)$ aligned to a gap, which contributes zero exact matches. The residual number is than equal to the number got in the optimal $D^{k}$ suffix alignment. Hence,
parallelism that reduces runtime necessary for repeated tasking synchronization and exchanging data. In addition, it makes efficient the scheduling through a task allocation policy that prefers the distribution according to the location of data. Each core (P) in the processor has a thread that its responsibility is calculation the maximum value of \( D \) and all threads runs in parallel. Calculations of the vector \( D \) are distributed over the total number of available cores \( P \). The elements of all vectors \( D_P, D_{E_P}, D_{E_P}, D_{E_P}, D_{E_P}, D_{E_P}, \) and \( D \) are accessible through the core’s shared memory.

The maximum value for each elements of \( D \) using \( N^D_D \) using Equation (8) are calculated in parallel. The value of each cell is evaluated in terms of its diagonal neighbor stored at \( D_1 \), with its left and upper neighbors stored at \( D_2 \), \( D_2 \), and then the maximum value is selected indicating the highest score.

Fig. 6 shows the parallelization approach. It displays the scheduling of calculations of the elements of \( D \) and distributed them on the available cores labeled \( P_0, P_1, P_2, \) and \( P_3 \). They run in parallel on each four sequent elements of \( D \), then they sequentially run on the second sequent 4 rows, and so on.

5. PERFORMANCE EVALUATION

The performance of the conceived parallel implementation of the proposed algorithm was extensively evaluated using different processing parameters. The evaluation methodology that was followed to correctly study the results obtained by the described solution is presented. The analysis goes from pair-wise alignment methods (SW). The improvements achieved due to introduced optimizations for multi-core system.

In this section, all needed information about the experimental setup for performance measurements is illustrated. It includes specifications of used platforms, details of experimented biological datasets, and characteristics of other programs used during comparisons.

A. Platform

As obviously clear from the previous section, the presented algorithm was designed to parallelize computations on a multi-core-based environment. Thus, to correctly evaluate the performance of the both original and proposed methods, the platform was considered as specified: An Intel quad-core-i7-3770, with 3.40 GHz processor and main memory of 8 GB; implementing on 64-bit Linux.
OS (Ubuntu) and running using C++ with OpenMP library.

### B. Datasets

The tests have been conducted using a variety of data sets. These data sets sequences including long, medium-length, and short sequences. These lengths are ranging from 400 to 34,500 residues to study the solution’s overall performance against multiple different sizes. The data sets consist of sequences selected from NCBI [20].

Table I shows the used data set with the number of sequences and average their length with a numerical measure of the scatter of a data set (standard deviation).

### C. Programs

Overall, tests have been conducted on the specific platforms using various groups of data sets. To evaluate the implementation of our algorithm, it was tested in

---

**Table I**

<table>
<thead>
<tr>
<th>Sequences’ No.</th>
<th>Sequences’ length</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>34,500</td>
<td>5.307</td>
</tr>
<tr>
<td>100</td>
<td>19,700</td>
<td>1.417</td>
</tr>
<tr>
<td>500</td>
<td>9200</td>
<td>514</td>
</tr>
<tr>
<td>400</td>
<td>856</td>
<td>7</td>
</tr>
<tr>
<td>1,000</td>
<td>858</td>
<td>8</td>
</tr>
<tr>
<td>400</td>
<td>408</td>
<td>3</td>
</tr>
<tr>
<td>2,000</td>
<td>266</td>
<td>2</td>
</tr>
<tr>
<td>4,000</td>
<td>247</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table II**

<table>
<thead>
<tr>
<th>Sequences’ No.</th>
<th>Sequences’ length</th>
<th>Our algorithm</th>
<th>CW-MPI</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>34,500</td>
<td>79,027</td>
<td>164,613.66</td>
<td>2.083</td>
</tr>
<tr>
<td>100</td>
<td>19,700</td>
<td>51,203</td>
<td>96,876.08</td>
<td>1.892</td>
</tr>
<tr>
<td>500</td>
<td>9,200</td>
<td>47,140</td>
<td>80,892.24</td>
<td>1.716</td>
</tr>
<tr>
<td>1000</td>
<td>858</td>
<td>2572</td>
<td>3569.94</td>
<td>1.388</td>
</tr>
<tr>
<td>400</td>
<td>856</td>
<td>382</td>
<td>626.86</td>
<td>1.641</td>
</tr>
<tr>
<td>400</td>
<td>408</td>
<td>163</td>
<td>187.12</td>
<td>1.148</td>
</tr>
<tr>
<td>2000</td>
<td>266</td>
<td>903</td>
<td>1,008.65</td>
<td>1.117</td>
</tr>
<tr>
<td>4000</td>
<td>247</td>
<td>2,631</td>
<td>2767.81</td>
<td>1.052</td>
</tr>
</tbody>
</table>
comparison to popular and efficient MSA program named ClustalW-MPI. This program is available online at: http://www.mybiosoftware.com/alignment/3052.

The runtime and speedup are considered most common performance measurements. Runtime is the elapsed time for all calculation, including all additions, comparisons, and maximum values. Speedup is the ratio between the runtime of the two involved programs.

Table II gives the runtime (in sec) and speedup of the two sequential of the proposed program against the ClustalW-MPI program computing the distance computation to illustrate how vectorization approach accelerates calculations.

Fig. 7 illustrates that the longer sequences have the more acceleration of our algorithm calculation. Furthermore, our program achieves a speedup up to 2-fold over ClustalW-MPI.

Furthermore, parallel runtime of both is shown in Table III. Fig. 8 shows the execution time and speedup of proposed parallel program on the above-mentioned datasets. Our parallel program achieved reducing the runtime of aligning sequences of length 34,500 from 164,613 s using ClustalW-MPI to 79,027 s using the proposed algorithm using 8 cores.

The speed-up of our parallel program achieved significant speedup of almost 3 for aligning sequences of longest length of sequence. Obviously, the sequences with the short length, the decrease the overall performance.
To conform that the proposed program is most efficient than other executed programs, the parallel efficiency of the available cores was evaluated. It is the ratio of the speedup (S) with respect to the number of processors (P) [21]. It is given by the following equation:

$$E = \frac{S}{P} \quad (8)$$

Results are shown in Table IV. It is clear that the efficiency of our algorithm is exponentially increasing as the length of sequences increases. In addition, our program supreme efficiency was up to 0.35 with respect to the ClustalW-MPI for the longest sequence length up to (34 k).

There is another performance measurement used in computational biology called billion cell updates per second (GCUPS). A GCUPS represents the time for a complete computation of one entry in the similarity matrix, including all comparisons, additions, and maximum operations. Table V shows the performance comparison for the datasets.

6. CONCLUSIONS

This paper presented a new parallel algorithm for computing the nid-value in the pair-wise local alignment. The results of the proposed algorithm were used in the first stage of MSA. Since the new algorithm was implemented using vectorizing technique, we have got a significant improvement in the performance.

The program was able to calculate nid-value for sequences with length up to (34 k) residues. It surpasses ClustalW-MPI 0.13 with 2.9 speedup and the efficiency reached 0.35. A better performance can be achieved if more cores are provided. Furthermore, it can be accomplished a higher speedup with improved efficiency.

Program’s performance figures vary from a low of 0.438 GCUPS to a high of 3.66 GCUPS as the lengths of the query sequences decrease from 34,500 to 9200.

REFERENCES


