ORIGINAL RESEARCH ARTICLE UHD JOURNAL OF SCIENCE AND TECHNOLOGY

An Improved Parallel Multiple Sequence Alignment Algorithm on Multi-core System

Mohammed W. Al-Neama[1], Salwa M. Ali[2] and Kasim A. Al-Salem[3]

[1]*Department of Computer Science, Education College for Girls, Mosul University, Mosul, Iraq*
[2]*Department of Mathematics, Faculty of Science, Ain Shams University, Cairo, Egypt*
[3]*Department of Computer Science, University of Cihan/Sulaimanya, Kurdistan Region, Iraq*

ABSTRACT

In this paper, we introduce an improved parallel algorithm for computing the number of exact matches nid (*S*,*T*) in the local alignment of two biological sequences *S* and *T*. This number is used in the first stage of progressive alignment to compute the distance between two sequences. The distance computations are usually its most computationally intensive part. Therefore, this work concentrates on improving an algorithm for this stage using vectorizing technique and running on multi-core. Our program is able to compute *nid* (*S*,*T*) between very long sequences, up to 34 k residues by C + + with OpenMP library on an Intel Core-i7-3770 quad-core processor of 3.40 GHz and main memory of 8 GB. It outperforms ClustalW-MPI 0.13 with 2.9-fold speedup, and the efficiency reached 0.35. Furthermore, a higher speedup with improved efficiency can be accomplished. Its 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.

Index Terms: Bioinformatics, Distance Computation, Multi-cores, Multiple Sequence Alignment, Parallel Programming

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.

The biological definition of the problem makes it time-consuming taking in consideration the fact that in practice, a single genome may contain in the order of 10^9 residues.

Access this article online

DOI: 10.21928/uhdjst.v1n2y2017.pp13-24

E-ISSN: 2521-4217
P-ISSN: 2521-4209

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

Corresponding author's e-mail: mwneama@uomosul.edu.iq

Received: 10-03-2017 Accepted: 25-03-2017 Published: 29-08-2017

```
    --T--CC-C-AGT--TATGT-CAGGGGACACG--A-GCATGCAGA-GAC
      |  || |  ||  | | | |||     || |  | |  | ||||   |
    AATTGCCGCC-GTCGT-T-TTCAG----CA-GTTATG--T-CAGAT--C

                        tccCAGTTATGTCAGgggacacgagcatgcagagac
                           ||||||||||||
aattgccgccgtcgttttcagCAGTTATGTCAGatc
```

Fig. 1. A global and a local alignment of the same two sequences

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 (*sbt*) on an alphabet $\Sigma = \{a_1, a_2..., a_n\}$ 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 unmatching 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 (go)) and another smaller penalty for

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

Fig. 2. The BLOSUM62 substitution matrix

extending already opened gaps (ge). 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 Σ of length L, a sequence S^g of length L^g over $\Sigma \cup$ "-" is called a gapped sequence of S if $L^g \geq L$ and there exist a transformation $T(S) = S^g$ such that:

1. $\forall\ 1 \leq i \leq L; \exists\ 1 \leq j \leq L^g; S(i) = S^g(j)$
2. If $S^g(p) = S(i)$, and $S^g(q) = S(j)$; 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 Σ 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 $n = \max(L_s^g, L_s^g, L_t^g)$ with the following properties: $\forall\ 1 \leq i \leq n$.

1. $M(s, i) = S^g(i)$ and $M(t, i) = T^g(i)$
2. If $M(s, i)$ = "-" then $M(t, i) \neq$ "-" and vice versa.

Definition 4: Given two sequences S and T on an alphabet Σ. A similarity score function of an alignment.

Score: $M \rightarrow R$

Score $(M) \in R$

Assigns a similarity score to each pair of characters in M.

Definition 5: A score of alignment is a real value function (score) that associate for each alignment $M(S,T)$ a real value function Score $(M(S,T))$.

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,

Score $(M(S,T)) \leq$ Score $(M_o(S,T))$ for all alignments $M(S,T)$.

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 $H(0,0)$ 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 formula:

$$H_L(s,t) = max\begin{Bmatrix} H_L(s-1,t-1) + sbt(S(s),T(t)), \\ H_L(s-1,t) + g, \\ H_L(s,t-1) + g \end{Bmatrix}$$

or

$$H_A(s,t) = max\begin{Bmatrix} H_A(s-1,t-1) + sbt(S(s),T(t)), \\ E(s,t) + g, \\ F(s,t) + g \end{Bmatrix}$$

$$E(s,t) = max\begin{Bmatrix} H_A(s,t-1) + go, \\ E(s,t-1) + ge \end{Bmatrix}$$

$$F(s,t) = max\begin{Bmatrix} H_A(s,t-1) + go, \\ E(s,t-1) + ge \end{Bmatrix}$$

Where, sbt is a substitution matrix and (g, go, and ge) 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:

$$H_L(s,t) = max\begin{Bmatrix} 0, \\ H_L(s-1,t-1) + sbt(S(s),T(t)), \\ H_L(s-1,t) + g, \\ H_L(s,t-1) + g \end{Bmatrix}$$

or

$$H_A(s,t) = max\begin{Bmatrix} 0, \\ H_A(s-1,t-1) + sbt(S(s),T(t)), \\ E(s,t) + g, \\ F(s,t) + g \end{Bmatrix}$$

$$E(s,t)=max\begin{Bmatrix} H_A(s,t-1)+go, \\ E(s,t-1)+ge \end{Bmatrix}$$

$$F(s,t)=max\begin{Bmatrix} H_A(s-1,t)+go, \\ F(s-1,t)+ge \end{Bmatrix}$$

Where, *sbt* is a substitution matrix and (*g*, *go*, and *ge*) 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:

$g = -1$,

$$sbt(S(s),T(t))=\begin{cases} 2 & if\ S(s)=T(t) \\ -1 & \text{Otherwise} \end{cases}$$

From the highest score ($H(8,11) = 10$), a procedure of trace-back 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.

	\	A	T	C	T	C	G	T	A	T	G	A	T
\	0	-2	-4	-6	-8	-10	-12	-14	-16	-18	-20	-22	-24
G	-2	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12
T	-4	-2	1	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
C	-6	-3	0	3	2	1	0	-1	-2	-3	-4	-5	-6
T	-8	-4	-1	2	5	4	3	2	1	0	-1	-2	-3
A	-10	-5	-2	1	4	4	3	2	4	3	2	1	0
T	-12	-6	-3	0	3	3	3	5	4	6	5	4	3
C	-14	-7	-4	-1	2	5	4	4	4	5	5	4	3
A	-16	-8	-5	-2	1	4	4	3	6	5	4	7	6
C	-18	-9	-6	-3	0	3	3	3	5	5	4	6	6

The best global alignment would be: ATCTCGTATGAT

where "|" = match ; "-" = gap G--TC-TATCAC

Fig. 3. Local alignment using Smith–Waterman

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]:

$$Dist(S,T)=1-\frac{nid(S,T)}{min\{L_s,L_t\}} \quad (2)$$

Where, *nid(S,T)* indicates the number of exact matches using SW algorithm to align S and T. For instance, the *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.

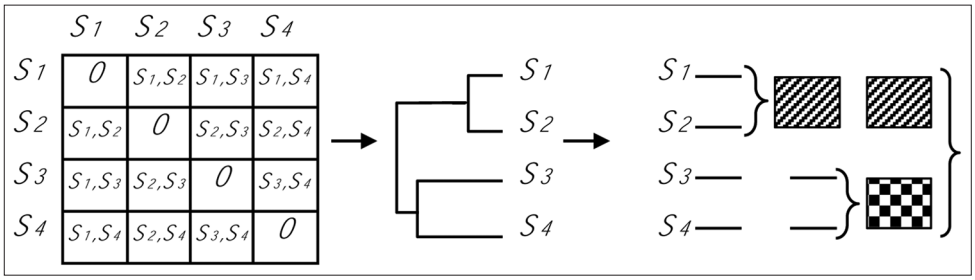

Fig. 4. Multiple sequence alignment produced with ClustalW

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 trace-back, 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:

$$N_L(s,t)=\begin{cases} 0 & if\ H_L(s,t)=0 \\ N_L(s-1,t-1) & if\ H_L(s,t)=H_L(s-1,t-1) \\ +m(s,t) & +sbt(S(s),T(t)) \\ N_L(s,t-1) & if\ H_L(s,t)=H_L(s,t-1)+g \\ N_L(s-1,t) & if\ H_L(s,t)=H_L(s-1,t)+g \end{cases} \quad (3)$$

Where,

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

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

$$N_A(s,t)=\begin{cases} 0 & if\ H_A(s,t)=0 \\ N_A(s-1,t-1) & if\ H_A(s,t)=H_L(s-1,t-1) \\ +m(s,t) & +sbt(S(s),T(t)) \\ N_E(s,t-1) & if\ H_A(s,t)=E(s,t) \\ N_F(s-1,t) & if\ H_A(s,t)=F(s,t) \end{cases} \quad (4)$$

Where,

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

$$N_E(s,t)=\begin{cases} 0 & if\ t=1 \\ N_A(s,t-1) & if\ E(s,t)=H_A(s,t-1)+go \\ N_E(s,t-1) & if\ E(s,t)=E(s,t-1)+ge \end{cases}$$

$$N_F(s,t)=\begin{cases} 0 & if\ t=1 \\ N_A(s,t-1) & if\ E(s,t)=H_A(s-1,t)+go \\ N_F(s,t-1) & if\ E(s,t)=E(s-1,t)+ge \end{cases}$$

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_E, D_{E1}, D_F, and D_{F1} 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, D_E is replaced by D_{E1}, and D_F is replaced by D_{F1}.

In the subsequent iteration, this cyclic method is used to replace the six buffers D_1, D_2, D_E, D_{E1}, $D_{F,}$ and D_{F1}. The values of all cells in D are computed in terms of its diagonal neighbor stored at D_1, with its left and upper neighbors stored at D_2 in addition the cells in D_E and D_F, and the maximum value is selected indicating the highest score.

Fig. 5 shows the vectorization approach when aligning pair of sequences S = {*GCTACTCAC*} and T = {*GCTAGG TATGAT*} with their lengths are 9 and 12, respectively. It illustrates the calculation of the elements of H_A, using affine gap ($go = 7$, $ge = -1$) and a substitution cost of (2) if the characters are match and (−1) otherwise, and how it is replaced by D, using D_1, D_2, D_E, D_{E1}, D_F, and D_{F1}.

This figure shows the dependence relationship of the elements of the matrices, which is visualized by considering its antidiagonals D_1, D_2, D_E, D_{E1}, $D_{F,}$ and D_{F1} dependencies. It is clear that antidiagonal D in iteration $i = 9$ computations depend on the four previously computed antidiagonals D^7, D^8, $\mathrm{D}_{\mathrm{E}}^{8}$, and D^8_F. Therefore, all other computed antidiagonals can be neglected.

Furthermore, it shows the dependence relationship of cell $D(5)$ with its left neighbor $D_E(5) = -9$, upper neighbor $D_F(4) = -8$, and upper left neighbor $D_1(4) = 5$. Where $D_2(5) + go$ is a maximum value of ($D_{E1}(5) + ge$ and $\mathrm{D}_{\mathrm{E}}^{1}(5) + go$); $D_2(4) + go$ is a maximum value of ($D_{F1}(4) + ge$ and $D_F(4) + go$). Using this way, all elements along vector D are computed in parallel from all elements in vectors D_1, D_2, D_E, and D_F.

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 (go for opining gap and ge for extending gap), substitution matrix (sbt), in iteration (i), and with element (j), the equation:

$$D^i(j) = H_A(j, i-j+1) \tag{5}$$

Gives a vectorization D^i of the matrix H_A and the following relations hold:

$$N^i(j) = N_A(j, i-j+1) \tag{6}$$

$$nid(S,T) = \max_i N^i(i_{max})$$

Where $max(2, s-L_s) \le i \le min((s+1), (L_s+1))$ and (i_{max}) indicates the position of the maximum value in the vectors D^i.

Proof: From Equation (1), we get:

$$H_A(j, i-j+1) = max\begin{cases} 0, \\ H_A(j-1, i-j+stb(S(j), \\ T(i-j+1)), \\ E(j, i-j+1), \\ F(j, i-j+1) \end{cases}$$

$$E(j, i-j+1) = max\begin{cases} H_A(j, i-j)+go, \\ E(j, i-j)+ge \end{cases}$$

$$F(j, i-j+1) = max\begin{cases} H_A(j-1, i-j)+go, \\ F(j-1, i-j+1)+ge \end{cases}$$

and from Equation (5), we get:

$$D^i(j) = max\begin{cases} 0, \\ D^{i-2}(j-1)+stb(S(j), T(i-j+1)), \\ D_E^i(j), \\ D_F^i(j) \end{cases}$$

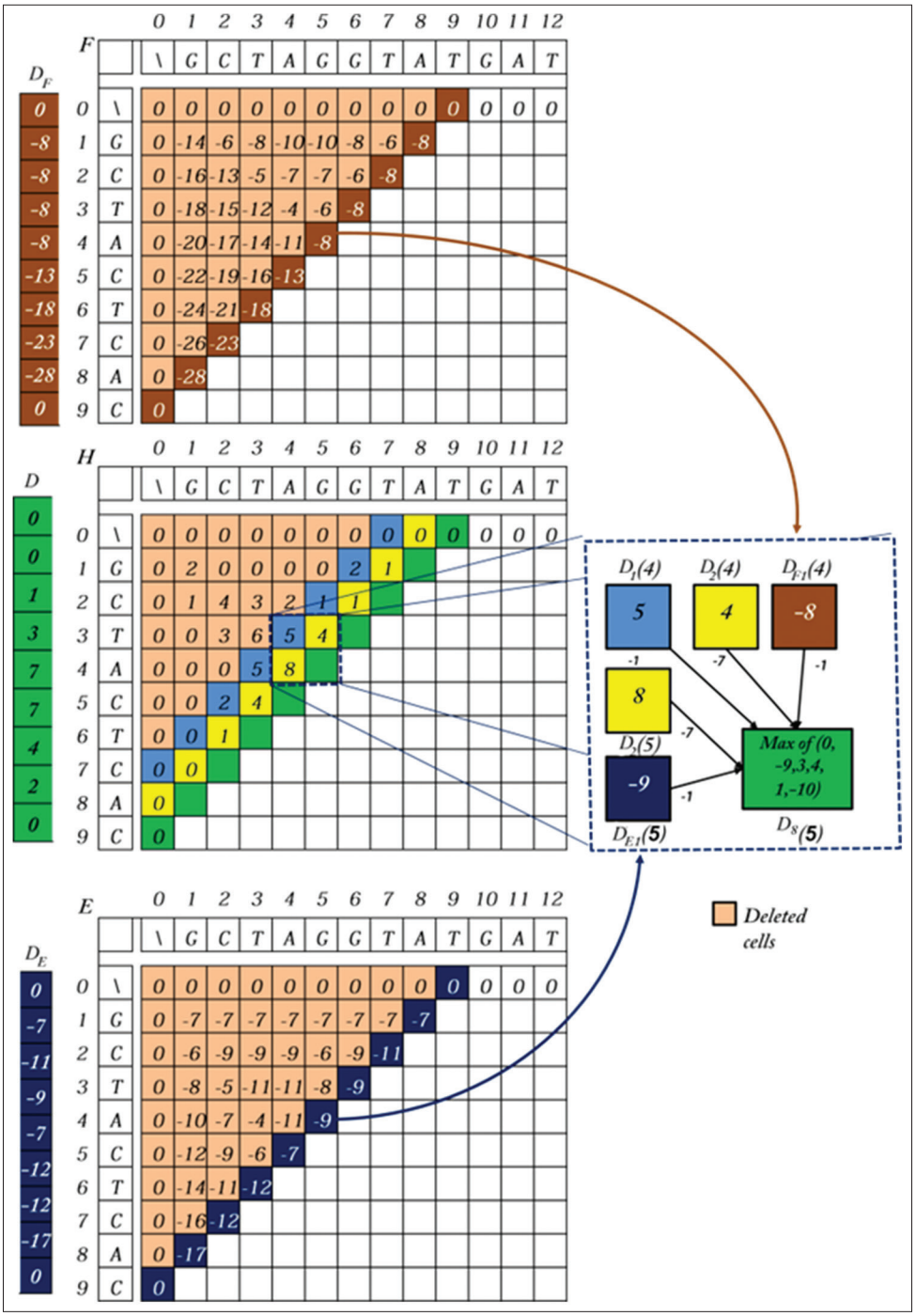

Fig. 5. Relation between *H*, *D*, D_1, D_2, D_E, and D_F

$$D_E^i(j) = max\begin{cases} D^{i-1}(j-1) + go, \\ D_E^{i-1}(j) + ge \end{cases}$$

$$D_F^i(j) = max\begin{cases} D^{i-1}(j) + go \\ D_F^{i-1}(j-1) + ge \end{cases}$$

Since D^{i-1} is computed in the previous iteration of D^i, D_E^i, and D_F^i, that is, in one iteration before D^i, D_E^i, and D_F^i, hence D_2^i is computed in the second iterations before D^i, let us denote D^{i-1}, D^{i-2}, D_E^{i-1}, and D_F^{i-1} by D^{i-1}, D^{i-2}, D_{E1}^i, and D_{F1}^i, respectively. Then, we get:

$$D^i(j) = max\begin{cases}0,\\ D_2^i(j-1) + stb(S(j), T(i-j+1)),\\ D^i{}_E(j),\\ D^i{}_F(j)\end{cases}$$

$$D_E^i(j) = max\left\{D_1^i(j-1) + go, D_{E1}(j) + ge\right.$$

$$D_F^i(j) = max\left\{D_1^i(j) + go, D_{F1}(j-1) + ge\right. \quad (7)$$

To proof of $N_D^i(j)$ let the Equation (6), gives the vectorization $N_A(s,t)$:

$$NA(j, i-j+1) =$$
$$\begin{cases}0 & if\ H(j,i-j+1)=0\\ N_A(j-1,i-j) & if\ H(j,i-j)=H(j-1,i-j)\\ +m(j,i-j+1) & +sbt(S(j),T(i-j+1))\\ N_E(j,i-j+1) & if\ H_A(j,i-j+1)=E(j,i-j+1)\\ N_F(j,i-j+1) & if H_A(j,i-j+1)=F(j,i-j+1)\end{cases}$$

$$N_E(j, i-j+1) =$$
$$\begin{cases}0 & if\ i-j=0\\ N_A(j,i-j) & if\ H_A(j,i-j+1)=H_A(j,i-j)+go\\ N_E(j,i-j) & if\ H_A(j,i-j+1)=E(j,i-j)+ge\end{cases}$$

$$N_F(j, i-j+1) =$$
$$\begin{cases}0 & if\ k=1\\ N_A(j-1,i-j+1) & if\ H_A(j,i-j+1)=H_A(j-1,i-j+1)+go\\ N_F(j-1,i-j+1) & if\ H_A(j,i-j+1)=F(j-1,i-j+1)+ge\end{cases}$$

After vectorizing the matrix N_A as shown in Fig. 5.

$$N_D^i(j) = \begin{cases}0 & if\ D^i(j)=0\\ N_D^{i-2}(j-1) & if\ D^i(j)=D_1^i(j-1)+\\ +m(j) & +sbt(S(j),T(i-j+1))\\ N_E^i(j) & if\ D^i(j)=D_E^i(j)+g\\ N_F^i(j-1) & if\ D^i(j)=D_F^i(j)+g\end{cases}$$

$$N_E^i(j) = \begin{cases}0 & if\ j=1\\ N_A^i(j) & if\ D(j)=D^i(j)+go\\ N_E^i(j) & if\ D(j)=D_E^{i-1}(j)+ge\end{cases}$$

$$N_F^i(j) = \begin{cases}0 & if\ j=1\\ N_A^i(j-1) & if\ D^i(j)=D^i(j-1)+go\\ N_F^i(j-1) & if\ D^i(j)=D_F^{i-1}(j-1)+ge\end{cases}$$

Then, we get:

$$N_D^i(j) = \begin{cases}0 & if\ D^i(j)=0\\ N_1^i(j-1) & if\ D^i(j)=D_1^i(j-1)\\ +m(j) & +sbt(S(j),T(i-j+1))\\ N_2^i(j) & if\ D^i(j)=D_E^{i-1}(j)+g\\ N_2^i(j-1) & if\ D^i(j)=D_F^i(j)+g\end{cases} \quad (8)$$

Where,

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

$$N_E^i(j) = \begin{cases}0 & if\ i-j=0\\ N_A^i(j) & if\ E^i(j)=D_i(j)+go\\ N_E^i(j) & if\ E^i(j)=D_{E1}^i(j)+ge\end{cases}$$

$$N_F^i(j) = \begin{cases}0 & if\ j=1\\ N_A^i(j-1) & if\ D^i(j)=D(j-1)+go\\ N_F^i(j-1) & if\ D^i(j)=D_{F1}^i(j-1)+ge\end{cases}$$

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

Case 1: *D(j) = 0*. The alignment is empty. Hence, $N_A(j)=0$.

Case 2: *D(j) = D¹(j−1)+sbt(S(j),T(i−j+1))*. The alignment ends with *S(j)* aligned to *T(i−j+1)*, which contributes *m(S(j)*, *T(i−j+1))* to the *nid*-value. The residual number is than equal to the *nid*-value got in the optimal *j−1* suffix alignment. Hence,

$N_A(j) = N_1(j-1) + m(S(j), T(i-j+1))$

Case 3: *D(j) = D²(j−1) + go*. 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 *D²(j−1)* suffix alignment. Hence,

$N_A(j) = N_2(j-1)$

Case 4: $D(j) = D^2(j) + ge$. The alignment ends with $T(i-j+1)$ aligned to a gap, which contributes to zero exact matches. The remaining number is equal to the number found in the optimal $D^2(j)$ suffix alignment. Hence,

$N_A(j) = N_2(j)$

The increase in N_D^i occurs only at the vectors D_i which has matching in its elements. Hence,

$N_A(x_{max}, y_{max}) = \text{nid}(S,T)$ is obtained by maximization $N_D^i(i_{max})$.

B. Parallelization Approach

Another critical challenge that must be dealt with is the gigantic explosion in the amount of molecular data which makes the ability to align a huge number of long sequences becoming even more essential.

For example, the Ribosomal Database Project Release 10 [16] consists of more than million sequences. This leads to the massive number of calculations. Even if the sequences are short, and pair-wise calculations can be done relatively quickly, say at a rate of 5000^{-1} s, then their alignment still requires almost 12 days of CPU time. Another difficulty is how to store the similarity matrix elements, as it will take up to 40 GB of memory. This leads to the need of new approaches to parallelize the calculations using sort of sophisticated parallel and distributed systems such as multi-cores.

Recent studies refer some attempts have made to accelerate computation of similarity matrix. Ying *et al.*, uses GPU's in [17]. They show speedup comparing with the serial CPU program. Wirawan *et al.*, [18] introduced a parallel algorithm on the cell broadband engine multi-core system for the calculation by taking benefit of the 128-bit SIMD vectorization registers of each SPEs and used half word values (16 bits) for the computation. Their results show a good speedup comparing with sequential ClustalW program.

Recently, Al-Neama *et al.* [19] proposed a new parallel algorithm of distance matrix computations of ClustalW is based on OpenMP system. It achieves speedup of about 2.39 on 50 sequences of the average length of 9200 nucleotide; tested on Core-i7 Intel Xeon 2.83GHz of the processor.

The second contribution of the proposed algorithm is parallelizing the computations to align the long-sequences dataset. The multithreads technique is used to apply the 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's 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_1, D_2, D_E, D_{E1}, D_F, D_{F1}, and D are accessible through the core's shared memory.

The maximum value for each elements of D using Equation (7) and N_D^i 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, with D_E and D_F, 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 $P0$, $P1$, $P2$, and $P3$. 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

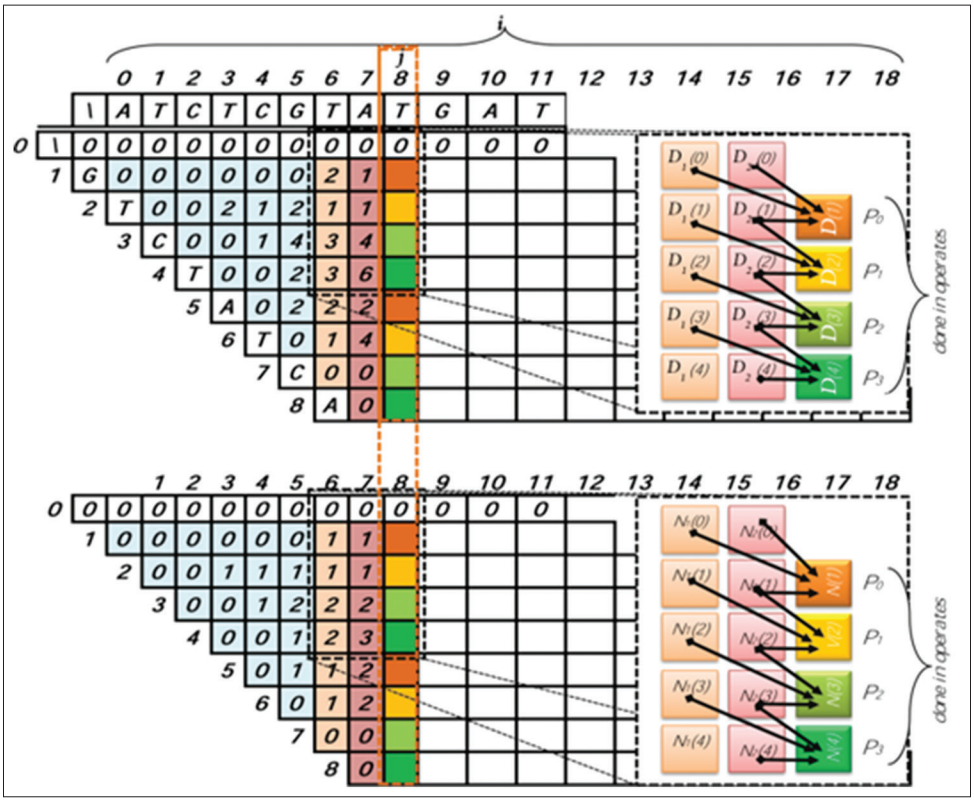

Fig. 6. Scheduling *D*'s computations on 4 cores

TABLE I
Used Benchmark Dataset Specifications

Sequences' No.	Sequences' length	Standard deviation
50	34,500	5.307
100	19,700	1.417
500	9200	514
400	856	7
1,000	858	8
400	408	3
2,000	266	2
4,000	247	2

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 II
Sequential Performance Measurements of Our Algorithm versus ClustalW-MPI

Sequences' No.	Sequences' length	Our algorithm	CW-MPI	Speedup
50	34,500	79,027	164,613.66	2.083
100	19,700	51,203	96,876.08	1.892
500	9,200	47,140	80,892.24	1.716
1000	858	2572	3569.94	1.388
400	856	382	626.86	1.641
400	408	163	187.12	1.148
2000	266	903	1,008.65	1.117
4000	247	2,631	2767.81	1.052

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 III
Parallel Performance Measurements of Our Algorithm versus ClustalW-MPI

Sequences' No.	Sequences' Length	Our Alg.	CW-MPI	Speedup
50	34,500	33,270	93,157.03	2.92
100	19,700	25,602	67,075.93	2.62
500	9,200	28,803	67,109.92	2.33
1000	858	240.66	517.42	2.15
400	856	115.73	223.59	1.93
400	408	1,955	3,67.98	1.72
2000	266	677.25	819.47	1.21
4000	247	2026	2329.75	1.15

TABLE IV
Efficiency Comparisons Using 8 Cores

Sequences' No.	Sequences' length	CW-MPI
50	34,500	0.35
100	19,700	0.33
500	9200	0.29
1000	858	0.27
400	856	0.24
400	408	0.22
2000	266	0.15
4000	247	0.14

TABLE V
Performance Comparison (in GCUPS) for Scanning the Datasets

Sequences' No.	Sequences' length	Our algorithm	CW-MPI
50	34,500	0.438	0.157
100	19,700	0.750	0.286
500	9200	3.667	1.574
1000	858	2.435	1.133
400	856	1.153	0.597
400	408	1.886	1.094
2000	266	2.104	1.739
4000	247	2.428	2.111

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

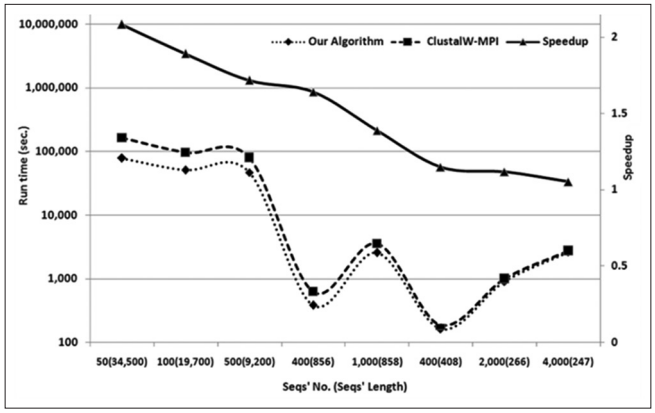

Fig. 7. Performance comparison between sequential our algorithm, ClustalW-MPI

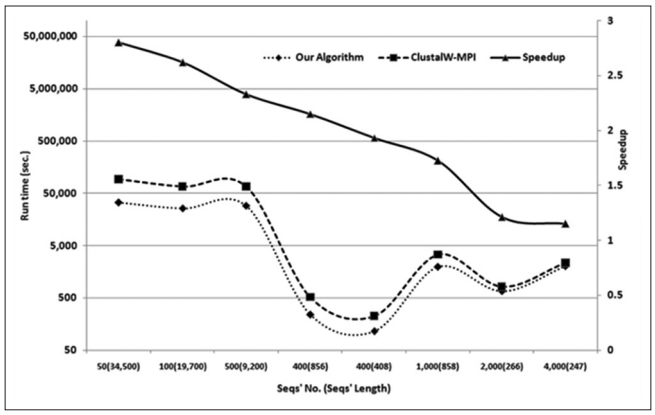

Fig. 8. Performance comparison between parallel our algorithm, ClustalW-MPI

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:

$$E = \frac{S}{P} \tag{8}$$

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

[1] S. B. Needleman and 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.

[2] T. F. Smith and M. S. Waterman. "Identification of common molecular subsequences." *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195-197, 1981.

[3] S. Henikoff, J. G. Henikoff and S. Pietrokovski. "Blocks+: A non-redundant database of protein alignment blocks derived from multiple compilations." *Bioinformatics*, vol. 15, no. 6, pp. 471-479, 1999.

[4] B. Schmidt. *Bioinformatics: High Performance Parallel Computer Architectures*. Florida: CRC Press, 2010.

[5] L. Wang and T. Jiang. "On the complexity of multiple sequence alignment." *Journal of Computational Biology*, vol. 1, no. 4, pp. 337-348, 1994.

[6] R. C. Edgar and S. Batzoglou. "Multiple sequence alignment." *Current Opinion in Structural Biology*, vol. 16, no. 3, pp. 368-373, 2006.

[7] D. F. Feng and R. F. Doolittle. "Progressive sequence alignment as a prerequisitetto correct phylogenetic trees." *Journal of Molecular Evolution*, vol. 25, no. 4, pp. 351-360, 1987.

[8] P. Sneath and R. Sokal. "Unweighted pair group method with arithmetic mean," in *Numerical Taxonomy*, San Francisco: W.H. Freeman and Company, pp. 230-234, 1973.

[9] N. Saitou and M. Nei. "The neighbor-joining method: A new method for reconstructing phylogenetic trees." *Molecular Biology and Evolution*, vol. 4, no. 4, pp. 406-425, 1987.

[10] K. Chaichoompu and S. Kittitornkun. "Multithreaded clustalw with im-proved optimization for intel multi-core processor," in *Communications and Information Technologies, 2006. ISCIT'06. International Symposium on. IEEE*, 2006, pp. 590-594.

[11] W. Liu, B. Schmidt, G. Voss and W. Muller-Wittig. "Streaming algorithms for biological sequence alignment on gpus." *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 9, pp. 1270-1281, 2007.

[12] F. F. Ghaleb, N. M. Reda and M. W. Al-Neama. "An overview of multiple sequence alignment parallel tools." in *Proc. CSCCA '13,* Dubrovnik, Croatia, 2013, pp. 91-96.

[13] Y. Liu, B. Schmidt and D. L. Maskell. "Msaprobs: Multiple sequence alignment based on pair hidden markov models and partition function posterior probabilities." *Bioinformatics*, vol. 26, no. 16, pp. 1958-1964, 2010.

[14] J. D. Thompson, B. Linard, O. Lecompte and O. Poch. "A comprehensive benchmark study of multiple sequence alignment methods: Current challenges and future perspectives." *Plosone*, vol. 6, no. 3, pp. 2101-2113, 2011.

[15] J. Daugelaite, A. O. Driscoll and R. D. Sleator. "An overview of multiple sequence alignments and cloud computing in bioinformatics," ISRN Biomathematics, 2013.

[16] J. R. Cole, Q. Wang, E. Cardenas, J. Fish, B. Chai, R. J. Farris, A. S. Kulam-Syed-Mohideen, D. M. McGarrell, T. Marsh, G. M. Garrity and J. M. Tiedje. "The ribosomal database project: improved alignments and new tools for rRNA analysis." *Nucleic Acids Research*, vol. 37, no. 1, pp. D141-D145, 2009.

[17] Z. Ying., X. Lin., S. C. W. See and M. Li. "GPU-accelerated DNA distance matrix computation," in *Proc. ChinaGrid 2011*, Dalian, Liaoning, China, 2011, pp. 42-47.

[18] A. Wirawan, C. K. Kwoh and B. Schmidt. "Multi threaded vectorized distance matrix computation on the Cell/BE and x86/SSE2 architectures." *Bioinformatics Advance*, vol. 26, no. 10, pp. 1368-1369, 2010.

[19] M. W. Al-Neama, N. M. Reda and F. F. Ghaleb. "Multiple sequence alignment on multi-cores." *International Journal of Biomathematics*, vol. 8, no. 6, p. 1550084, 2015.

[20] National Center for Biotechnology Information (NCBI); 2017. Available: https://www.ncbi.nlm.nih.gov/. [Last Accessed on 2017 Aug 24].

[21] G. Hager and G. Wellein. *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton, FL: CRC Press, 2010.