Grid Computing-based Sequence Alignment

M. Shalaby[1] **and Y. Kamal**[2]
[1] Department of Information system, Modern university for information and technology, El-Hadaba El Wosta, Zone 5, Cairo, Egypt
[2] Department of Communication and Electronics, Giza Higher Institute of Engineering & Technology Giza, Egypt

Abstract - *Sequence alignment plays an important role in bioinformatics. There are two types of sequence alignment, global and local alignment. Needleman-Wunsch algorithm is the most famous example of global alignment. It is based on sequential computing so it has a problem of being slow. In this paper we present a new global alignment algorithm that can be implemented using parallel computing (such as grid computing) rather than sequential computing. The grid computing-based sequence alignment has the advantage of being as fast as heuristic algorithms and as sensitive as dynamic programming algorithms. We also present a comparative study between our approach and Needleman-Wunsch algorithm according to time and spaces complexity*.

Keywords: Sequence Alignment, Dynamic Programming, Grid Computing

1 Introduction

Sequence alignment is used to detect the functional, structural, or evolutionary similarities between two DNA, RNA, or protein sequences [1]. Detection of similarities in viruses and bacterium can help scientists find treatments for diseases.

There are several methods that have been implemented to solve the sequence alignment problem. The sequence alignment methods are either global or local. Global alignment conducts an end to end alignment between the query sequence and the subject sequence and is used to compare homologous genes, while local alignment aligns a substring (or the whole string) of the query sequence to a substring (or the whole string) of the subject sequence and is used to discover homologous portions in non-homologous genes.

Needleman-Wunsch algorithm [2] is the most famous example of global alignment. This algorithm is based on dynamic programming, it gives accurate results, however it is time consuming. Smith-Waterman algorithm [3] is another dynamic programming algorithm which is very similar to Needleman-Wunsch algorithm, however, it is used for local alignment. Other examples of local alignment algorithms are FASTA [4] and BLAST [5] which are heuristic-based pairwise local aligner. Heuristic methods are much faster than dynamic programming methods, however they are less sensitive.

In this paper we present a pairwise aligner algorithm that aims at obtaining as accurate results as Needleman-Wunsch and Smith-Waterman algorithms and is as fast as FASTA and BLAST. Unlike Needleman-Wunsch algorithm which is sequential in nature, we present an algorithm that can be implemented using grid computing and hence it overcomes the time consumption problem of dynamic programming methods.

In section 2, we briefly describe the Needleman-Wunsch algorithm. In section 3, we introduce the concept of grid computing. In section 4, we present the proposed grid computing-based sequence alignment algorithm. In section 5, we discuss the results of applying our approach and present a comparative study between this approach and the traditional Needleman-Wunsch algorithm. Finally in section 6, we conclude this work.

2 Global alignment using Needleman-Wunsch algorithm

Needleman-Wunsch algorithm (also called optimal matching algorithm) is a dynamic programming algorithm that is used to align globally two Protein sequences. Let $SEQ1$ be a sequence of residues (symbols) with length M and $SEQ2$ be a sequence of residues with length N, then the algorithm constructs a two dimensional matrix Q of dimension $(M+1) \times (N+1)$. We denote the entry of row m and column n as Q_{mn}, where Q_{mn} is calculated according to the scoring matrix [6-10] and the gap penalty scheme. Here, S denotes the scoring matrix, $S(i,j)$ denotes the similarity score between the residue of index i in $SEQ1$ and the residue of index j in $SEQ2$, and d denotes the gap penalty. The elements Q_{mn} are calculated as follows

$$Q_{0n} = d \times n \quad (1)$$

$$Q_{m0} = d \times m \quad (2)$$

$$Q_{mn} = max \begin{cases} Q_{(m-1)(n-1)} + S(i,j) \\ Q_{(m-1)n} + d \\ Q_{m(n-1)} + d \end{cases} \quad (3)$$

To calculate the elements Q_{mn} where $0 < m \leq M, 0 < n \leq N$, the elements $Q_{(m-1)(n-1)}, Q_{(m-1)n}, Q_{m(n-1)}$ (source elements) must exist and hence the algorithm is executed sequentially. After the calculation of all elements Q_{mn}, the algorithm starts from the bottom right element (Q_{MN}) and traces back the elements Q_{mn} which leaded to this score. Let $SEQ1(m)$ be the residue of index m in $SEQ1$ and $SEQ2(n)$ be the residue of index n in $SEQ2$. If the source element equals to $Q_{(m-1)(n-1)}$ the residue $SEQ1(m)$ is aligned to $SEQ2(n)$, if the source element equals to $Q_{(m-1)n}$ the residue $SEQ1(m)$ is aligned with gap, if the source element equals to $Q_{m(n-1)}$ the residue $SEQ2(n)$ is aligned with gap. Fig. 1 shows the pseudo code of Needleman-Wunsch algorithm, the inputs of this algorithm are $SEQ1$ and $SEQ2$, and the outputs are the aligned sequences A_SEQ1 and A_SEQ2.

```
Needleman-Wunsch algorithm
Input: SEQ1, SEQ2
for i=0 to length(SEQ1)
  Qi0 ← d*i
for j=0 to length(SEQ2)
  Q0j ← d*j
for i=1 to length(SEQ1)
  for j=1 to length(SEQ2)
  {
    Match ← Q(i-1)(j-1) + S(i,j)
    Delete ← Q(i-1)j + d
    Insert ← Qi(j-1) + d
    Qij ← max(Match, Insert, Delete)
  }
A_SEQ1 ← ""
A_SEQ2 ← ""
i ← length(SEQ1)
j ← length(SEQ2)
while (i > 0 or j > 0)
{
   If(i>0 and j>0 and Qij == Q(i-1)(j-1) +
S(i,j))
  {
    A_SEQ1 ← SEQ1(i) + A_SEQ1
    A_SEQ2 ← SEQ2(j) + A_SEQ2
    i ← i - 1
    j ← j - 1
  }
  elseif(i>0 and Qij==Q(i-1)j+d)
  {
    A_SEQ1 ← SEQ1(i) + A_SEQ1
    A_SEQ2 ← "-" + A_SEQ2
    i ← i - 1
  }
  else (j>0 and Qij==Qi(j-1) + d)
  {
    A_SEQ1 ← "-" + A_SEQ1
    A_SEQ2 ← SEQ2(j) + A_SEQ2
    j ← j - 1
  }
}
Output: A_SEQ1, A_SEQ2
```

Figure 1: Needleman-Wunsch algorithm.

3 Grid computing

Grid computing is software and hardware structure that provides high computing capabilities by sharing distributed resources like computers, storage space, and software applications [11]. The resources of grid computing structure must be scheduled [12]. The scheduling process has three stages: resources discovery, resource selection, and job submission [13].

Scheduling is either static or dynamic. In the static mode, the information of all resources and the tasks of the application are available when the application is scheduled [14, 15]. Unlike static mode, the dynamic scheduler allocates tasks to resources while the application executes. This case is useful when the number of iterations in a loop cannot be determined and jobs arrive in real time [16-20].

The overhead cost is the time required to schedule tasks and communicate the results of finished distributed tasks. The overhead cost should be minimized so that it does not negate the benefits of using grid computing [21].

In section 5 we show the results of applying grid computing to the concurrent tasks of our proposed approach.

4 Proposed approach

This approach aligns the two sequences $SEQ1$, and $SEQ2$. It assumes that length of $SEQ1$ is greater than or equal length of $SEQ2$. The algorithm searches for the maximum length consecutive match in $SEQ1$ and $SEQ2$. Here, the consecutive match is defined as the consecutive sequence of residues in $SEQ1$ and $SEQ2$ such that the cumulative score increases as long as one residue of the first consecutive sequence is compared with its corresponding residue of the second consecutive sequence. After finding the maximum length consecutive match, $SEQ1$ and $SEQ2$ are split into three sub-sequences each, the left side of consecutive match (L_SEQ1, L_SEQ2), the maximum consecutive match (M_SEQ1, M_SEQ2), and the right side of consecutive match (R_SEQ1, R_SEQ2). Again, search for the maximum consecutive match is applied recursively for the (L_SEQ1, L_SEQ2) and (R_SEQ1, R_SEQ2). For each recursive iteration, we add the gaps required to keep the maximum consecutive match in each part aligned, then we concatenate these three parts.

4.1 Proposed Approach Sub-modules

The proposed approach can be divided into three main sub-modules: Consecutive match sub-module (CM), split sequence sub-module (SS), and combine sub-sequences sub-module (CS).

The CM algorithm obtains all possible consecutive matches (we denote the consecutive matched residues of $SEQ1$ as $WORD1$ and the consecutive matched residues of $SEQ2$ as

$WORD2$) and saves the parameters of each match in a parameters table (P_TABLE). These parameters include $WORD1$ start index, $WORD2$ start index, the consecutive match length (W_LENGTH), and the consecutive match score ($SCORE$). CM algorithm starts with finding i, j such that the similarity score between $SEQ1(i)$ and $SEQ2(j)$ is positive. Therefore $WORD1$ is initialized to $SEQ1(i)$ with start index i, and $WORD2$ is initialized to $SEQ2(j)$ with start index j. After that, the next residues of index $(i+1)$ and $(j+1)$ are checked, they are added to $WORD1$ and $WORD2$ respectively if the similarity score between them are positive. The last step is repeated as long as the last residues similarity check is positive. The parameters table (P_TABLE) is initially empty and constructed only in the first run of CM algorithm, and hence $FIRST\_TIME$ parameter is initialized to TRUE. Eventually, P_TABLE is searched to obtain the maximum consecutive match (M_SEQ1, M_SEQ2) as well as other maximum consecutive match parameters (M_SEQ1 start index, M_SEQ2 start index, W_LENGTH). Figure 2 shows the pseudo code of CM algorithm.

```
CM algorithm
Input: SEQ1, SEQ2, P_TABLE, FIRST_TIME
If FIRST_TIME
{
 FIRST_TIME ← FALSE
 For k=0 to length(SEQ1)
 {
   SCORE ← 0
   For j=0 to length(SEQ2)
   {

 SEQ1_INDEX ← j + k (modulo length(SEQ1))
    If S(SEQ1_INDEX,j) > 0
    {
       If W_LENGTH=0
     Get WORD1_START_INDEX,
           WORD2_START_INDEX
        SCORE ← SCORE+
S(SEQ1_INDEX,j)
        Add SEQ1(SEQ1_INDEX) to
WORD1
        Add SEQ2(j) to WORD2
        WLENGTH ← WLENGTH+1
        Add the match parameters to
     P_TABLE
     }
    Else
    {
        SCORE ← 0
        WORD1 ← ""
        WORD2 ← ""
        W_LENGTH ← 0
     }
   }
 }
}
Get the maximum consecutive match
parameters from match parameters table
Output: P_TABLE , Maximum
consecutive match parameters
```

Figure 2: The pseudo code of CM algorithm.

The SS algorithm obtains (L_SEQ1, L_SEQ2) and (R_SEQ1, R_SEQ2) and hence it exploits the match parameters table (P_TABLE) to generate match parameter tables (LP_TABLE, RP_TABLE) for (L_SEQ1, L_SEQ2) and (RSEQ1, RSEQ2), respectively. LP_TABLE and RP_TABLE are generated such that LP_TABLE only considers the consecutive matches in L_SEQ1 and L_SEQ2, and RP_TABLE only considers the consecutive matches in R_SEQ1 and R_SEQ2. We note that, the start index and word length parameters of any consecutive match may be modified if it overlaps with the maximum consecutive match. Figure 3 shows the pseudo code of SS algorithm.

```
SS algorithm
Input: Maximum consecutive match
parameters, P_TABLE
Get L_SEQ1,L_SEQ2
Get R_SEQ1, R_SEQ2
For each record in P_TABLE
  If match parameters belong to
L_SEQ1, L_SEQ2
   {
     If (L_SEQ1 overlaps M_SEQ1) or
   (L_SEQ2 overlaps M_SEQ2)
       Modify parameters
     Add these parameters to
   LP_TABLE
 }
  else if match parameters belong
to LSEQ1, LSEQ2
   {
     If (RSEQ1 overlaps MSEQ1) or
   (RSEQ2 overlaps MSEQ2)
       Modify parameters
     Add these parameters to
   RP_TABLE
    }
Output: L_SEQ1, L_SEQ2, R_SEQ1,
R_SEQ2,LP_TABLE, RP_TABLE
```

Figure 3: The pseudo code of SS algorithm.

CS algorithm simply concatenates (L_SEQ1, M_SEQ1, R_SEQ1) and concatenates (L_SEQ2, M_SEQ2, R_SEQ2). For each iteration, gaps might be added to either L_SEQ1 or L_SEQ2 to make their length equal and keep maximum length consecutive match aligned. Similarly, gaps might be added to either R_SEQ1 or R_SEQ2 for the same reason.

4.2 Example

As an example, we use Blosum50 scoring matrix [22] and consider the two sequences,

$SEQ1$: *BHYYXALKRHHQWWHHQWW*

$SEQ2$: *HYYQCBBALKRHHQXHQWY*

In the first iteration, the algorithm fetches the maximum length consecutive match and obtains the following results,

M_SEQ1: *ALKRHHQ*, L_SEQ1: *BHYYX*, R_SEQ1: *WWHHQWW*

M_SEQ2: *ALKRHHQ*, L_SEQ2: *HYYQCBB*, R_SEQ2: *XHQWY*

In the second iteration, the algorithm fetches the maximum consecutive match in both left side and right side sub-sequences. For the left side sub-sequence, we obtain the results,

M_SEQ1: *HYY*, L_SEQ1: *B*, R_SEQ1: *X*

M_SEQ2: *HYY*, L_SEQ2: , R_SEQ2: *QCBB*

At this stage, the algorithm combines these results and adds the necessary gaps. Therefore, the aligned sub-sequences are

BHYY - - - X

- HYYQCBB

For the right side sub-sequences in the second iteration and taking into account that the similarity between the two residues ('W','Y') is positive, we obtain the results

M_SEQ1: *HQWW*, L_SEQ1: *WWH*, R_SEQ1:

M_SEQ2: *HQWY*, L_SEQ2: *X*, R_SEQ1:

Again, at this stage the results are combined and the aligned sub-sequences are

WWHHQWW

X - - HQWY

Back to the first iteration, the two aligned sub-sequences are concatenated so we obtain the following results

BHYY - - - XALKRHHQWWHHQWW

- HYYQCBBALKRHHQX - - -HQWY.

5 Results

In this section we apply the grid computing trend to the proposed approach presented in section 4. We use a network of three personal computers (with dual core 2.66 GHz processor, and 3 Gigabyte RAM each), one coordinator and two agents. The network topology is star where the coordinator controls the two agents. We use User Datagram Protocol as the data communication protocol (UDP). The tasks that can be executed concurrently are processed in parallel by the two agents. The first parallel task that is dispatched by coordinator is creating the lookup table of consecutive matches. The coordinator aggregates the results obtained by the two agents to find the maximum length consecutive match. The second parallel task is done after splitting the two sequences where searching for the maximum length consecutive match in left side sub-sequences is done in parallel (by the two agents) with searching for the maximum length consecutive match in right side sub-sequences.

On one hand, we use the environment described above to align many pairs of Protein sequences of different lengths. On the other hand, we align the same pairs of sequences using the typical Needleman-Wunsch algorithm which is sequential in nature using one personal computer (with dual core 2.66 GHz processor, and 3 Gigabyte RAM each).

To present a comparative study between our Grid Computing-based approach and Needleman-Wunsch approach, we analyze the time and space complexity of their main modules (which are constructing the lookup table in the proposed approach and filling the two dimensional matrix Q in Needleman-Wunsch approach). Let M be the length of the first sequence and N be the length of the second sequence, then the time complexity of the construction of the lookup table in the proposed approach is $O(MN)$ and so does the time complexity of filling the matrix Q in Needleman-Wunsch approach. Each record of the lookup table in the proposed approach occupy 16 bytes of memory so the space complexity of constructing the lookup table is $O(16L) \approx O(L)$, where L is the number of records in the lookup table. The space complexity of the matrix Q in Needleman-Wunsch approach is $O(MN)$.

Figure 4 shows the execution time of the two approaches for different values of MN. The difference between the execution time of the two approaches increases dramatically as MN increases.

Empirical results show that the proposed grid computing environment is not efficient for small sequence lengths (where $MN < 20000$) because the non-distributed devices outperform the distributed devices as a result of the overhead cost.

6 Conclusions

We presented a grid computing-based sequence alignment algorithm that can be used for the global alignment of a pair of Protein sequences. The proposed algorithm differs from the traditional Needleman-Wunsch algorithm in that some modules can be executed concurrently, and hence it is convenient to implement such algorithm using grid computing. The proposed approach has been implemented using a star network of three computers (one coordinator that controls two agents) and UDP data communication protocol.

Unlike the time complexity which is equal in the two approaches, the space complexity of the proposed approach is less than its counterpart for Needleman-Wunsch algorithm.

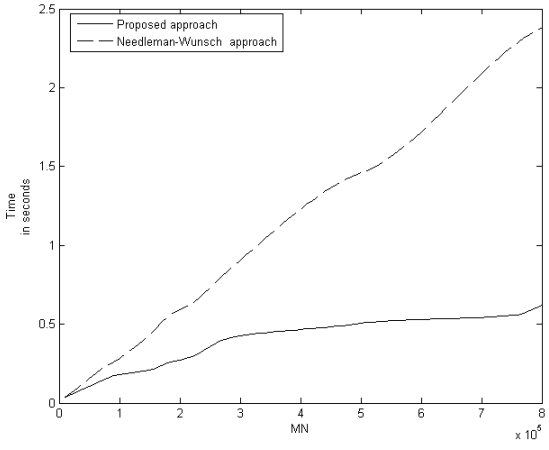

Figure 4: The execution time of the proposed approach and Needleman-Wunsch approach, where MN is the product of the length of the first sequence and the length of the second sequence.

Empirical results show that the proposed approach is exponentially faster than Needleman-Wunsch algorithm. They also show that applying the proposed approach on short-length sequences using one computer is more convenient than using a grid of computers because the overhead cost negates the benefits of using grid computing in short-length sequence alignment.

7 References

[1] D. Mount. "Bioinformatics: Sequence and Genome Analysis (2nd ed.)". Cold Spring Harbor Laboratory Press: Cold Spring Harbor, NY. ISBN 0-87969-608-7, 2004.

[2] S. B. Needleman and C. D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins"; J. Mol. Biol. 48, 443-453, 1970.

[3] T. F. Smith and M. Waterman. "Identification of common molecular subsequences"; J. Mol. Biol. 147, 195-197, 1981.

[4] D. J. Lipman and W. R. Pearson. "Rapid and Sensitive Protein Simlarity Searches"; Science 227, 1435-1441, 1985.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman. "A Basic Local Alignment Search Tool"; J. Mol. Biol., 215, 403-410, 1990.

[6] T. Müller, S. Rahmann, and M. Rehmsmeier; "Non-symmetric score matrices and the detection of homologous transmembrane proteins". Bioinformatics (Ox-ford, England). 17 Suppl 1: S182–9. PMID 11473008, 2001.

[7] D. W. Rice, D. Eisenberg. "A 3D-1D substitution matrix for protein fold recognition that includes predicted secondary structure of the sequence"; Journal of Molecular Biology, 267, 4, 1026–38. doi:10.1006/jmbi.1997.0924. PMID 9135128, 1997

[8] Gong, Sungsam, Blundell, and L. Tom. "Discarding functional residues from the substitution table improves predictions of active sites within three-dimensional structures"; In Levitt, Michael. PLoS Computational Biology 4 (10): e1000179. doi:10.1371/journal.pcbi.1000179. PMC 2527532. PMID 18833291, 2008.

[9] N. C. Goonesekere, B. Lee. "Context-specific amino acid substitution matrices and their use in the detection of protein homologs"; Proteins, 71, 2, 910–9. doi:10.1002/prot.21775. PMID 18004781, 2008.

[10] Y. M. Huang, C. Bystroff. "Improved pairwise alignments of proteins in the Twilight Zone using local structure predictions"; Bioinformatics, 22, 4, 413–22. doi:10.1093/bioinformatics/bti828. PMID 16352653, 2006.

[11] I. Foster and C. Kesselman. "The Grid: Blueprint for a Future Computing Infrastructure". Morgan Kaufmann Publishers, 1999.

[12] Y. Zhu. "A Survey on Grid Scheduling Systems", Department of Computer Science, Hong Kong University of science and Technology, 2003.

[13] J. Schopf. "Ten Actions When SuperScheduling", document of Scheduling Working Group, Global Grid Forum, http://www.ggf.org/documents/GFD.4.pdf, 2001.

[14] R. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen, and R. Freund. "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems"; J. of Parallel and Distributed Computing, 61, 6,810-837, 2001.

[15] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman. "Heuristics for Scheduling Parameter Sweep Applications in Grid Environments"; Proc. of the 9th hetero-geneous Computing Workshop (HCW'00), pp. 349-363, Cancun, Mexico, 2000.

[16] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak, and J. Pukacki. "Improving Grid Level Throughput Using Job Migration And Rescheduling"; Scientific Programming, 12, 4, 263-273, 2004.

[17] A. Takefusa, S. Matsuoka, H. Casanova, and F. Berman. "A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid"; Proc. of the 10th IEEE International Symposium on High Performance Distributed

Computing (HPDC-10'01), pp. 406-415, San Francisco, California USA, 2001.

[18] H. Chen and M. Maheswaran. "Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems"; Proc. of the 16th International Parallel and Distributed Processing Symposium (IPDPS), pp. 88-97, Fort Lauderdale, Florida USA, 2002.

[19] N. Muthuvelu, J. Liu, N. L. Soe, S. Venugopal, A. Sulistio, and R. Buyya. "A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids"; Proceedings of the 3rd Australasian Workshop on Grid Computing and e-Research, Newcastle, Australia, 2005.

[20] G. Malathi, S. Sarumathi. "Survey On Grid Scheduling"; Journal of Computer Applications, 3, 3, 22-29, 2010.

[21] F. Azzedin and M. Maheswaran. "Metagrid: A scalable framework for wide-area service deployment and management"; Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRIDí02), Berlin, Germany, 2002.

[22] S. Henikoff, and J. G. Henikoff. "Amino acid substitution matrices from protein blocks"; Proc. Natl. Academy Science 89, 915-919, 1992.