Parallel Needleman-Wunsch Algorithm for Grid

Tahir Naveed[1], Imitaz Saeed Siddiqui[2], Shaftab Ahmed[3]
tahir.naveed@gmail.com, imtiaz.saeed@gmail.com, shaftab@bci.edu.pk

[1, 2, 3] Department of Computer Sciences & Engineering, Bahria University Islamabad, Pakistan

Abstract

A fast computation solution is required to analyze and infer large amount of data in various applications. In BioInformatics, DNA sequence information is critical to understanding genetic variations. DNA sequencing is the process of determining the exact order of the chemical building blocks in a sample. This computation requirement is the greatest technical challenge in the Human Genome Project. The Needleman-Wunsch algorithm is used for alignment of DNA Sequences under Global Alignment category. A fast computation solution is proposed through a parallel version of this algorithm and use of the Alchemi Grid as the processing engine.

Keywords—DNA Sequence, Global Alignment, Grid, Needleman-Wunsch, Parallel Algorithm

1. Introduction

The length of a normal DNA Sequence makes about 40KB to 60KB long string in FASTA format [5]. Aligning two DNA sequences [2][3][4][6] requires a long time on a single processor. The algorithm has a complexity of O(NxM) where N is length and M is depth of 2D array. The Parallel Needleman-Wunsch algorithm proposed is based on Needleman-Wunsch algorithm [9] to globally align [7] [8] two DNA Sequences using multiple processors, it reduces the time to O(N+M).

Grid Computing [1] is an emerging technology to provide high performance computing in a virtual organization composed of a large number of computers connected through web based technologies. We have implemented a parallel version of the Needleman-Wunsch algorithm for handling the DNA matching and alignment problem. The Alchemi grid has been used to the run the algorithm in grid environment.

2. Algorithm

A sequence alignment is a scheme of writing one sequence on top of another where the residues in one position are deemed to have a common evolutionary origin. If the same letter occurs in both sequences then this position is conserved in evolution. If the letters differ it is assumed that the two derive from an ancestral letter (which could be one of the two or neither). Homologous sequences may have different lengths. Thus, a letter or a stretch of letters may be paired up with dashes in the other sequence to signify such an insertion or deletion. An insertion in one sequence can always be seen as a deletion in the other one, we use the term *idel* for such operation.

In such a simple evolutionarily motivated scheme, an alignment mediates the definition of a distance for two sequences. One generally assigns 0 to a match, some negative number to a mismatch and a larger negative number to an indel. By adding these values along an alignment one obtains a score for an alignment. A distance function for two sequences can be defined by looking for the alignment, which yields the minimum score. By dynamic programming this minimization can be effected without explicitly enumerating all possible alignments of two sequences.

A. Global Alignment [15]

Global Alignment assumes that the two proteins are basically similar over the entire length of one another. The alignment attempts to match them to each other from end to end, even though parts of the alignment are not very convincing. :

```
NLGPSTKDFGKISESREFDNQ
 |      ||||    |
QLNQLERSFGKINMRLEDALV
```

B. Local Alignment [15]

Local alignment searches for segments of the two sequences that match well. There is no attempt to force entire sequences into an alignment, just those parts that appear to have good similarity, according to some criterion are considered. Using the same sequences as above, one could get:

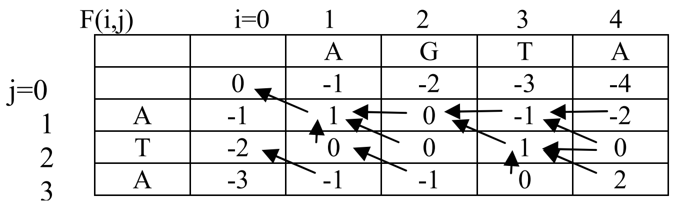
```
NLGPSTKDDFGKILGPSTKDDQ
          ||||
QNQLERSSNFGKINQLERSSNN
```

Most commonly used algorithm for local alignment is Smith-Waterman algorithm [16].

3. Needleman-Wunsch algorithm [9][10][11]

All possible pairs of residues (DNA bases or protein amino acids) - one from each sequence - are represented in a 2-dimensional array. The sequences are written across the top and down the left side of the matrix, except that an extra row (row #0) and column (column #0) are added to allow the alignment to begin with a gap of any length in either sequence. The gap rows are filled with penalty scores for gaps of increasing lengths. Maximum possible values are calculated for all other boxes below, to the right of the top row and left column using the above scoring functions. All possible alignments are represented by pathways through this matrix. Each cell is the maximum possible score for an alignment ending at that point. For each cell, look at all possible pathways back to the beginning of the sequence (allowing gaps) and give that cell the value of the maximum scoring pathway.

Figure 1 shows the matrix filled with values and pointers. In implementation there are two matrices, one to store the calculated values and another is used to store the pointers which will be used later to trace back for the optimal alignment



F(i,j)		i=0	1	2	3	4
			A	G	T	A
j=0		0	-1	-2	-3	-4
1	A	-1	1	0	-1	-2
2	T	-2	0	0	1	0
3	A	-3	-1	-1	0	2

Figure 1 - Filled Needleman-Wunsch Matrix and Traceback Pointers

Every non-decreasing path from (0, 0) to (M,N) corresponds to a global alignment of the two sequences.

1. Initialization

$F(0, 0) = 0$
$F(0, i) = -i * d$
$F(j, 0) = -j * d$

2. Main Iteration

For each i = 1 . . . M
For each j = 1 . . . N

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j), & \text{case 1} \\ F(i-1, j) - d, & \text{case2} \\ F(i, j-1) - d, & \text{case3} \end{cases}$$

$$Ptr(i, j) = \begin{cases} DIAG, & \text{if case 1)} \\ LEFT, & \text{if case 2} \\ UP, & \text{if case 3} \end{cases}$$

3. Termination

F(M,N) is the optimal score, and from Ptr(M,N), we can trace back the optimal alignment. The optimal global alignment is:

```
A G T A
A - T A
```

4. Performance

Time: O(NxM) (We need to fill out the whole matrix)
Space: O(NxM) (We need a matrix to store all the trace back pointers)

Needleman-Wunsch algorithm had used dynamic programming [11] approach to solve the problem in a small memory space, instead of making all combinations of DNA matching and using a big memory space.

4. Parallel Needleman-Wunsch algorithm

A parallel version of Needleman-Wunsch algorithm [9] has been developed; which uses multiple processors for initializing, Calculating and filling the DataMatrix (Stores the DNA Sequences and their calculated values) and the PointerMatrix (Stores DNA Sequences and the pointer values to be used later in backtracking). Our algorithm doesn't include backtracking process to keep record for values to be calculated in each iteration on parallel machines. This algorithm has been implemented on Grid using Alchemi Framework [14].

All the matrices in parallel version of Needleman-Wunsch algorithm are places in global memory space so that all available processors can access them at the same time to perform initialization and other calculations.

A. Parallel Initialization of Matrices using different CPUs

In our example we will show the implementation of our algorithm on 3 CPUs. One of which contains global memory and rest of two are used for calculations.

The DataMatrix and PointerMatrix are initialized with DNA Sequences and the gap values are inserted as shown in Figure 2 and Figure 3, mathematically. This step is handled in parallel on the participating machines.

It is noticeable that this initialization of both matrices with DNA sequences can be performed in parallel, provided there are four CPUs available, which means that Step 1.1 and Step 1.2 can be further performed in parallel.

After above step, same is the case in initializing the two matrices with Gap values. Which means Step 1.4 and Step 1.6 can be performed in parallel as well, provided four CPUs are available.

1.1

Parallel For-Loops to fill DataMatrix with two sequences Seq1 and Seq2 using two different CPUs executing each For-Loop

```
For i=2 to Length of DataArray
    DataArray [0,i] = Seq1[i-2]

For j=2 to Depth of DataArray
    DataArray [j,0] = Seq1[i-2]
```

1.2

Parallel For-Loops to fill PointerMatrix with two sequences (seq1 and seq2) using two different CPUs executing each For-Loop

```
For i=2 to Length of PointerArray
    PointerArray [0,i] = Seq1[i-2]

For j=2 to Depth of PointerArray
    PointerArray [j,0] = Seq1[i-2]
```

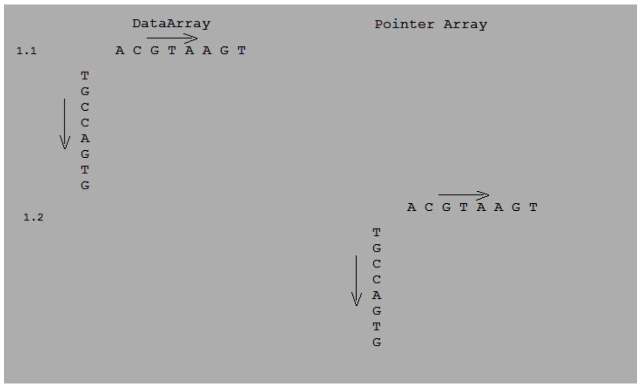


Figure 2 - Initializing with DNA Sequences

Figure 2 shows the initialization of DataMatrix and PointerMatrix with DNA sequence graphically.

1.3

Initializing the anchor point of the DataMatrix

```
DataArray [1,1] = 0
```

1.4

Parallel For-Loops to fill DataMatrix with GAP values using two different CPUs executing each For-Loop

```
Temp = 0
For i=2 to Length of DataArray
    Temp = Temp + GAP
    DataArray [1,i] = Temp

Temp = 0
For j=2 to Depth of DataArray
    Temp = Temp + GAP
    DataArray [j,1] = Temp
```

1.5

Initializing the anchor point of the PointerMatrix

```
PointerArray [1,1] = 0
```

1.6

Parallel For-Loops to fill PointerMatrix with GAP values using two different CPUs executing each For-Loop

```
Temp = 0
For i=2 to Length of PointerArray
    Temp = Temp + GAP
    PointerArray [1,i] = Temp
Temp = 0
For j=2 to Depth of PointerArray
    Temp = Temp + GAP
    PointerArray [j,1] = Temp
```

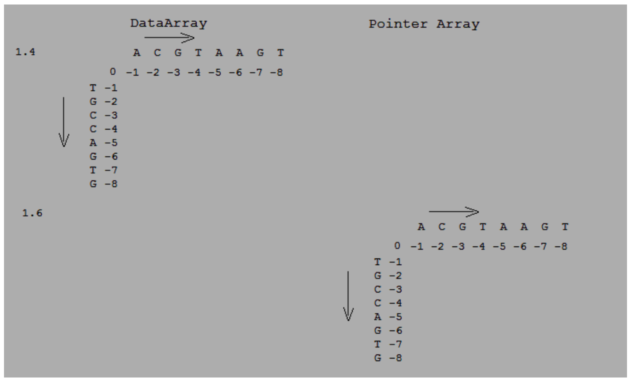


Figure 3 - Initializing with Gap Values

Figure 3 shows the initialization of DataMatrix and PointerMatrix with Gap values graphically.

B. Preparation for Parallel Computing

DataArray

		A	**C**	**G**	**T**	**A**	**A**	**G**	**T**
	0	-1	-2	-3	-4	-5	-6	-7	-8
T	-1	X	X	X	X	X	X	X	X
G	-2	X	X	X	X	X	X	X	X
C	-3	X	X	X	X	X	X	X	X
C	-4	X	X	X	X	X	X	X	X
A	-5	X	X	X	X	X	X	X	X
G	-6	X	X	X	X	X	X	X	X
T	-7	X	X	X	X	X	X	X	X
G	-8	X	X	X	X	X	X	X	X

Figure 4 - Illustrating values to be calculated in parallel [12]

Figure 4 shows the technique to calculate values in parallel. In first iteration, values at 2,2 will be calculated in parallel. In 2nd iteration 2,3 and 3,2 will be calculated in parallel and so on. Another matrix named MyMatrix, has been introduced which will keep track of indexes of DataMatrix to be calculated in parallel.

```
duration1 = 1
For (loop1 = 0 ; loop1 < duration1 ; loop1++)
    itemp = 2
    jtemp = duration1
    For a = 0 to loop1
        str = itemp+,+jtemp
        newArr[loop1, a] = str
        itemp++
        jtemp--
    if (duration1 < length)
        duration1++

iitemp = length/2 + 1
duration2 = length/2
newI = length

For ( loop2 = duration2 ; loop2 >= 0 ; loop2--)
    itemp = iitemp
    jtemp = length

    For (int a = loop2 ; a >= 0 ; a--)
        str = itemp+,+jtemp
        newArr[newI-1, a] = str
        itemp++
        jtemp—

    newI++
    iitemp++
    if (duration2 >= length)
        duration2—
```

After Above calculation MyMatrix will look like

2,2							
3,2	2,3						
4,2	3,3	2,4					
5,2	4,3	4,3	2,5				
6,2	5,3	4,4	3,5	2,6			
7,2	6,3	5,4	4,5	3,6	2,7		
8,2	7,3	6,4	5,5	4,6	3,7	2,8	
9,2	8,3	7,4	6,5	5,6	4,7	3,8	2,9
9,3	8,4	7,5	6,6	5,7	4,8	3,9	
9,4	8,5	7,6	6,7	5,8	4,9		
9,5	8,6	7,7	6,8	5,9			
9,6	8,7	7,8	6,9				
9,7	8,8	7,9					
9,8	8,9						
9,9							

Figure 5 – MyMatrix

The matrix MyMatrix shown in Figure 5, explains that in first iteration only 2,2 will be calculated, in 2nd iteration 3,2 and 2,3 will be calculated in parallel and in third iteration 4,2 , 3,3 and 2,4 will be calculated in parallel and so on. We can't calculate 2,2 and 9,9 in parallel because 9,9's result will not be available until 9,8 and 8,9 are not present.

This MyMatrix clearly indicates that more CPUs means more parallel assigning, which will result in parallel calculation, biggest parallel calculation that can be performed in this step required eight CPUs for 8th iteration. Which means if less than eight processors (lets assume 5) are available then 5 values will be calculated in parallel and the remaining 3 values will be calculated in parallel to complete eighth iteration.

C. *Sequential Assigning for Parallel Calculation*

This step of algorithm is assuming that 2 CPUs are available.

```
CPU1 = 0 // shows CPU 1 is free
CPU2 = 0 // shows CPU 2 is free

For i=0 to Depth of MyArray
    For j=0 to Length of MyArray
        If MyArray [i,j] < > null
            While ( CPU1 < > 0 OR CPU2 < > 0 )
            {
            If CPU1 == 0
            DataArray [i,j] = MaxofCPU1( MyArray [i,j] )
            Else
            DataArray [i,j] = MaxofCPU2( MyArray [i,j] )
            }
        Else
            Exit j Loop
```

For-i loop check for Parallel Values to be calculated
For-j assigns CPUs the indexes for which they will calculate values

D. *Instructions on each CPU - int MaxofCPUn (int i,int j)*

The variables i and j are the coordinates of the DataMatrix's value to be calculated.

```
int max(int i, int j)
    {
        //Getting previously calculated values
        Diagonal = DataArray [i-1,j-1]
        Up = DataArray [i-1,j]
        Left = DataArray [i,j-1]

        //Calculating all 3 values to compare
        If ( DataArray [i,0] == DataArray[0,j] )
            Diagonal = Diagonal + MATCH
        Else
            Diagonal = Diagonal+ NoMATCH
        Up = Up + GAP
        Left = Left + GAP

        //Returning Max value and filling Pointer Matrix
        If ( Diagonal > Left AND Diagonal > Up )
            PointerArray[i,j]="3"
            return Diagonal
        Else If( Up > Left )
            PointerArray[i,j]="2"
            return Up

        Else
            PointerArray[i,j]="1"
            return Left
    }
```

5. Implementation Problem

All the matrices DataMatrix, PointerMatrix, MyMatrix are global and are accessible to the CPUs on a Grid. While implementing the above parallel Needleman-Wunsch algorithm using Alchemi framework [14], we faced the problem of increased network traffic. For small size of matrix it is not significant. However with typical sizes of DNA sequences the network traffic overhead has to be reduced. To handle this problem two formulas as under were used:

$$\text{No. of Threads} = \text{Ceil}\left[\frac{\text{No. of values in the current diagonal}}{\text{Threshold [Upper limit]}}\right]$$

Where Threshold is the range of values from which we select the number of values to be solved per thread.

$$\text{Workload} = \text{Ceil}\left[\frac{\text{No. of values in the current diagonal}}{\text{No. of Threads}}\right]$$

Workload is the number of values to be solved per thread.

Sessions

For each new diagonal a new session is created. Each session consists of one or more threads depending on the length of the diagonal and the threshold (range of values from which the workload is chosen with the help of formule). Each new session is dependant on the result of its previous session. As long as the threads of the a session are running, new session cannot be created.

Threads

A thread [13] is assigned a certain workload with respect to the number of values in the current diagonal and the threshold. All the threads that belong to the same session are totally independent of each other and thus can be solved in a parallel fashion.

6. Conclusion

By developing Needleman's parallel algorithm we have reduced the calculation time from O(NxM) to O(N+M) [12]. The Alchemi framework for grid computing has been used to demonstrate the usefulness of the concept for large sequences like DNA.

Initialization steps are already parallel in this new algorithm and a slight change in algorithm can enhance initiation to double of current speed provided more processors are available.

Preparation for parallel calculation step also indicates that more CPUs means more calculations to be performed in parallel as in the above example, eight CPUs are required to obtain the best of this algorithm.

REFERENCES

[1] Krishna N. and Akshay L. and Dr. Rajkumar B., 2002, Alchemi v0.6.1 Documentation [online], University of Melbourne. Available: http://alchemi.net/doc/0_6_1/index.html

[2] About the Human Genome Project [online], Oak Ridge National Laboratory. Available: http://www.ornl.gov/sci/techresources/Human_Genome/project/about.shtml

[3] The Science Behind the Human Genome Project [online], Oak Ridge National Laboratory. Available: http://www.ornl.gov/sci/techresources/Human_Genome/project/info.shtml

[4] Facts About Genome Sequencing [online], Oak Ridge National Laboratory.Available: http://www.ornl.gov/sci/techresources/Human_Genome/faq/seqfacts.shtml

[5] FASTA Format Description [online], NGFN-BLAST by Nationale Genomforschungsnetz. Available: http://ngfnblast.gbf.de/docs/fasta.html

[6] Source of DNA Sequences [online], National Center for Biotechnology Information. Available: http://www.ncbi.nlm.nih.gov/mapview

[7] Pairwise Sequence Comparison [online], Lab of Bioinformatics, Institute of Computing Technology (ICT), Chinese Academia of Sciences (CAS). Available: http://www.bioinfo.org.cn/lectures/index-13.html

[8] Introduction to Bioinformatics - Chapter 5 - Introductory Sequence Analysis [online], Human Genome Mapping Project Resource Centre (HGMP-RC) by UK Medical Research Council. Available: http://portal.rfcgr.mrc.ac.uk/Courses/Jemboss_3day/Chapter5.html#Global%20sequence%20alignment

[9] Rong X, Jan 2003, Pairwise Alignment - CS262 - Lecture 1 Notes [online], Stanford University. Available:
http://ai.stanford.edu/~serafim/cs262/Spring2003/Notes/1.pdf

[10] Bin Wang, 2002, Implementation of a dynamic programming algorithm for DNA Sequence alignment on the Cell Matrix Architecture [online], Utah State University, Logan, Utah. Available:
http://www.cellmatrix.com/entryway/products/pub/wang2002.pdf

[11] Chand T. John, April 2004, CS273: Algorithms for Structure and Motion in Biology, Stanford University. Available:
http://www.stanford.edu/class/cs273/scribing/8.pdf

[12] DNA Sequence Comparison [online], The BioWall by Swiss Federal Institute of Technology in Lausanne (EPFL). Available:
http://lslwww.epfl.ch/biowall/VersionE/ApplicationsE/SequenceE.html

[13] Krishna N. and Akshay L. and Dr. Rajkumar B., 2002, Alchemi v0.6.1 Documentation [online], University of Melbourne. Available:
http://alchemi.net/doc/0_6_1/index.html

[14] Krishna N. and Akshay L. and Dr. Rajkumar B., 2002, Alchemi v0.6.1 Documentation [online], University of Melbourne. Available:
http://alchemi.net/

[15] BioInformatics Educational Resources
Documentation [online], European Bioinformatics Institute United Kingdom. Available:
http://www.ebi.ac.uk/2can/tutorials/protein/align.html

[16] Chitta Baral, Computational Molecular Biology, CSE 591
Arizona State University, United States of America. Available:
http://www.public.asu.edu/~cbaral/cse591-s03/classnotes/seq-align.pdf