A Parallel Needleman – Wunsch – Hirschberg Bio-sequence Alignment Algorithm.

Conference Paper · January 2010

Source: DBLP

CITATIONS
3

READS
288

2 authors:

Luis de la Torre
Universidad Metropolitana
13 PUBLICATIONS **19** CITATIONS
SEE PROFILE

J. Seguel
University of Puerto Rico at Mayagüez
52 PUBLICATIONS **138** CITATIONS
SEE PROFILE

A Parallel Needleman – Wunsch – Hirschberg Bio-sequence Alignment Algorithm

Luis de la Torre[1], **and Jaime Seguel**[2]
[1]Department of Mathematics and Physics
University of Puerto Rico at Cayey, Cayey, Puerto Rico
[2]Electrical and Computer Engineering Department
University of Puerto Rico at Mayaguez, Mayaguez, Puerto Rico

Abstract - *The Needleman – Wunsch (NW) algorithm is a dynamic programming method for aligning bio-sequences. Although the method is exact, bioinformatics practitioners often prefer to perform their sequence alignment with a heuristic method, the well-known BLAST. The arguments against Needleman-Wunsch are its high memory demands, a relatively slower response time, and sometimes, difficulties in producing a load-balanced parallelization. Memory demands have been alleviated with a variant based on Hirschberg longest common sequence identification algorithm, which we refer as the Needleman–Wunsch–Hirschberg (NWH) algorithm. NWH trades memory by computations, but preserving the O(n^2) time complexity bound of NW. Its parallelization however, is even more challenging than that of the NW algorithm. This paper discusses an intermediate approach, which we called parallel Needleman-Wunsch-Hirschberg (PNWH). PNWH keeps memory space within reasonable bounds, eliminates the extra computational burden of NWH, and yields an exact load-division and balancing formula for its effective parallelization.*

Keywords: Sequence Alignment, Parallel Algorithm, Needleman – Wunsch

1 Introduction

Let S_1 and S_2 be sequences over an alphabet A. An *alignment* of S_1 and S_2 is a pair S_1' and S_2' of the sequences over A U { – }, where " – " is the so – called gap symbol. A pair S_1', S_2' is an alignment of S_1 and S_2 if and only if it satisfies

i. Length(S_1') = Length(S_2'), and
ii. when the gap symbol is removed from S_j' we get back S_j, j = 1, 2.

This work concerns strings over the protein alphabet {A, R, N, D, C, Q, E, G, H, I, L, K, M, F, P, S, T, W, Y, V}. Similar sequences are assumed to have diverged from a common ancestor by *mutation* and/or *selection*. Mutational processes alter strings by performing on them:

i. *Insertions*: Addition of residues (letters)
ii. *Deletions*: Removal of residues
iii. *Substitutions*: Replacement of residues.

Insertions and deletions are commonly referred as *gaps*. *Selection*, in turn, preserves mutations that have resulted in favorable genotype traits.

The sequence alignment problem (SAP) is an *optimization problem* whose objective is maximizing a score function. SAP is formulated on the basis of a scoring scheme, this is, a pair consisting of a substitution matrix and a gap penalty. The substitution matrix is a two – dimensional array containing all score(a, b), where a and b are letters in the protein alphabet. Substitution matrices are thus, statistically sound statements of the odds (likelihood ratio) of observing a and b in an alignment. The best known substitution matrices for the protein alphabet are: BLOSUM and PAM matrices. A gap penalty is a mapping that assigns a penalty to the insertion of a gap for producing an alignment. Each scoring scheme determines a unique *score function*. The string alignment problem (SAP) can be stated as follows: Given two sequences S_1 and S_2 over an alphabet and a scoring scheme; what is the alignment pair S_1', S_2' that maximizes the Score function? A Brute force approach to SAP involves comparing $O(2^{2n})$ alignments between sequences of length n. Fortunately, SAP can be solved in quadratic time with the *Needleman – Wunsch Algorithm* [5]**.** This method is based on the next recursive formula. Given a scoring scheme consisting of a substitution matrix [s(x_i, y_j)] and a linear gap penalty mapping $\varphi(g) = -\lambda g$, where g is the number of gap symbols inserted, the score of aligning the x_i character in the first sequence with the y_j character in the second , denoted $S_{i,j}$, is obtained by:

$$\text{Basis: } (\forall i, j), S_{0,j} = \lambda j; S_{i,0} = -\lambda i$$

$$\text{Recursion: } (\forall i, j > 0), S_{i,j} = \max \begin{cases} S_{i-1,j-1} + s(x_i, y_j) \\ S_{i-1,j} - \lambda \\ S_{i,j-1} - \lambda \end{cases}$$

After completing matrix of scores [$S_{i,j}$], the method traces back the results to discover the optimal alignment(s). Trace back starts from the bottom right corner of [$S_{i,j}$] and runs column by column back to the upper left corner of the matrix of scores. At each cell in the matrix, the process creates a pointer to either the cell to the left, the cell immediately up, or the cell up in the diagonal direction. In order to decide the direction of pointer the method asks: With which of the three neighboring entries (the one up, the one to the left, or the one across a diagonal) was the maximum achieved?. Thus, NW creates a matrix of pointers [P[k, j]] together with the computation of the scores. NW recursion perform $O(m)$ steps, each consisting of $O(n)$ operations.

		A	R	D	H	H	G
	0	-1	-2	-3	-4	-6	-6
A	-1	5	4	3	2	1	0
A	-2	4	3	2	1	0	1
D	-3	3	2	11	10	9	8
H	-4	2	3	10	21	20	19
H	-5	1	2	9	20	31	30

Three alignments:	(i) ARDHHG AADHH-	(ii) A-RDHHG AA-DHH-	(iii) AR-DHHG A-ADHH-

Fig. 1: Optimal alignments for S_1=AADHG and S_2=ARDHHG using BLOSUM50 and the linear penalty φ(g)=-g

Backtracking, in turn, involves $O(n+m)$ operations. Therefore, their execution times are bounded above by the square of the length of the longest protein sequence. Thus, in terms of time complexity, NW is a relatively fast method. In terms of storage requirements, it is clear that [P[*k, j*]] can be constructed either row by row, or column by column. Therefore, NW *does not require keeping all values of* matrix [$S_{i,\ j}$] but the entries of the previous row or column. Nonetheless, NW requires the storage of *all pointers* [P[*k, j*]] for performing the trace back phase. Therefore, the space complexity of NW is also quadratic. This may be problematic in some instances of SAP. Yeast proteins are on average 466 amino acids (i.e characters) long [2], but the largest known proteins are the titins, a component of the muscle, with a total length of almost 27,000 amino acids. And there are some other 10,000 or more letter long proteins. So, although current estimates for the average protein length is around 300 characters [1], there are instances of SAP in which storage demands of NW are too high. Fig. 1 is an illustration of Needleman – Wunsch method. Hirschberg [4,6] designed an $O(n+m)$ space algorithm for identifying the longest common subsequence of a pair of sequences. Central to this design is the fact that the length of the longest common sequence is the same when measure with both sequences in reversed order. Although the theory behind Hirschberg's algorithm is cast in terms of sequence lengths and character-to-character comparisons without insertions; the principles are extensible to score optimization. A concept that is key in this extension is that of path is a score matrix. A *path* in a score matrix S = S[*k, j*] is a sequence of entries of S: S(0, 0), … ,S (k_1, j_1), S(k_2, j_2), … ,S(n – 1, m – 1) whose indices satisfy the rule $k_2 = k_1$ and $j_2 = j_1 + 1$; or $j_2 = j_1$ and $k_2 = k_1 + 1$; or $k_2 = k_1 + 1$ and $j_2 = j_1 + 1$. Using this concept, we can prove the following theorem:

Theorem 1: Let S = S[*k, j*] be a score matrix of a pair of sequences and let S*[*k, j*] be the score matrix of the same sequences but in reversed order. Then,

i. For each $j_1 = 0, \dots, n$ there is a pair of entries S[k_1, j_1] and S*[k_2,$m - j_1$] such that *Score* = S[k_1, j_1] + S*[k_2,$m - j_1$]=$\max_k$\{ S[k, j_1] + S*[k,$m - j_1$]\}

ii. The values S[k_1, j_1] , $k_1 = 0, \dots, n$ form paths in S which are in one – to – one correspondence with the results of backtracking

This result allows a *trade – off of space for computations*. Each pair of column S[: ,j}, and S*[: ,m – j] are computed and the pair of indices [*k,j*] that satisfies i in Theorem 1 is found and stored. This is done for each pair of columns S[,j}, and S*[,m – j]. At each of these computations, the space saving method *stores only the coordinates of a path through S*. Because the paths traverse the score matrices moving either to the right, diagonal, or down, at each computation, some sub-blocks are eliminated. This allows for keeping the time complexity of the NWH method within the $O(n^2)$ bound.

2 Parallelization Strategy and Load Balancing Analysis

Two observations lead to the parallelization of NWH. First is the fact that sub-blocks of the matrix of scores can be computed in parallel, as soon as their "predecessor block" computes their corresponding border column and row. We assume that computation and communication time are affine functions in the size of the block of the form of: $w(t)=wt+O_w$ and $c(t)=ct+O_c$, respectively. We also assume that communication cost is always less than computation cost

Fig. 2.a represents a Score matrix decomposed in blocks of x rows and y columns. The computation of light colored blocks precedes that of darker ones. The numbers correspond to processors, being 1 the master and 2 – 4 the workers. Fig. 2.b is a Gantt chart for the schedule of these computations for a Score matrix S in a master and three workers.

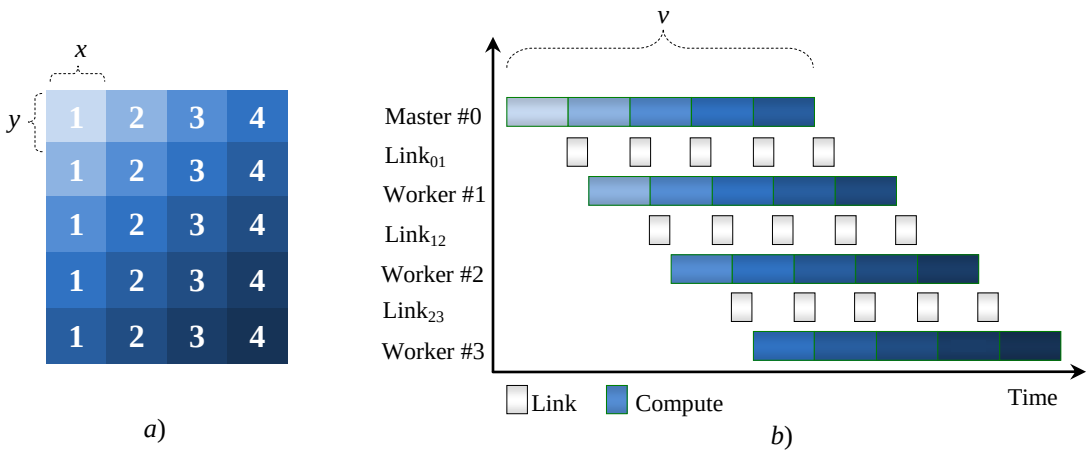

Fig. 2: a) Score matrix decomposition, b) Gantt Chart for the execution of Score matrix with a master and three workers

Thus, the problem of minimizing the execution time can be written as:

$$\text{Minimize } \mu(y) = (v + p - 1)w(xy) + pc(y)$$

$$\text{subject to } \quad vy = m$$

$$px = n$$

$$y, x, p \geq 1, v \geq 1$$

where v is number of vertical partitions of matrix S, x and y are the number of columns and rows of a sub-block, respectively; and p the number of processors in the system. The constraints imposed, turn the objective function into a decreasing mapping. As a consequence, we may set $x=n/p$. By applying Lagrange multipliers [8] we get the equation

$$mpO_w(pwn + p^2c - 1)^2 - (pwn + p^2c - wn)(wnm - p^2O_w) = 0$$

which is solved numerically. With the optimal values of x and y, we derive the following parallel Needleman-Wunsch-Hirschberg method:

Given p processors, x, y and v, processor i computes v blocks of size yx of matrix S and v blocks of size yx of matrix S*; where the columns in the blocks of matrix S* are in one to one correspondence with the columns in the blocks of matrix S through the relation i in Theorem 1. All computations are stored. Processor i identifies the pairs of indices whose addition achieves the value of Score as in i of Theorem 1; and reports the corresponding path segment to the master.

2.1 Parallel Algorithm

Algorithm PNWH

```
Input: A substitution matrix [s(xk, yj)], a linear penalty map φ(g) = -g, and sequences S1, S2, S1* and S2*, n = Length(S1), m = Length(S2).

[p,x,y,v]=optimalvalues(n,m,w,c,Ow,Oc)
broadcast(S1,S2)

Master (id==0)
 For j = 1 to v
  S[0: x,(j-1)y:jy)]= ComputeMatrix(x,y,j,"")
  Send(S[x,(j-1)y:jy)],j,'') to 1
 End For
 For i= 1 to p-1
  recv(sol[ix:(i+1)x],i,'',i)
 End For

Master (id==p-1)
 For j = 1 to v
  S*[0: x,(j-1)y:jy)]= ComputeMatrix(x,y,j,*)
  Send(S*[x,(j-1)y:jy)],j,'*',p-2)
 End For


Worker (1<=id<=p-2)
 For j = 1 to 2*v
  Recive(temp,i,rever,idtemp);
  temp[:,iy:(i+1)y]=ComputeMatrix(x,y,i,rever)
  if rever==*
    S*←temp// Actualize S* whit temp values
    if (0<id<p-1) Sent(temp,i,'*',id-1)
  Else
    S←temp //Actualize S whit temp values
    if (0<id<p-1) Sent(temp,i,'',id+1)
  End if
 End for
 Max=ComputeMax(S,S*);
 Sent(Max,id,'',0)
```

The function `optimalvalues(n,m,w,c,Ow,Oc)`, has the following parameter as input: the size of sequences and the parameter for the computation and communication function, and returns the optimal parameter obtained by solving the minimization execution time problem.

The function `ComputeMatrix(x,y,j,tag)` computes the `j` block of score matrix S or S* depending on the `tag`.

The function `Sent(data,i,tag,j)` sends the `i` corresponding border column of S or S*, depending on the `tag` identifier to the processor `j`.

The function `ComputeMax(S,S*)` computes the maximum entries in the `S+S*` respective segments

Fig 3 shows in a Gantt Chart the execution times of the proposed method in a system consisting of a master and three workers

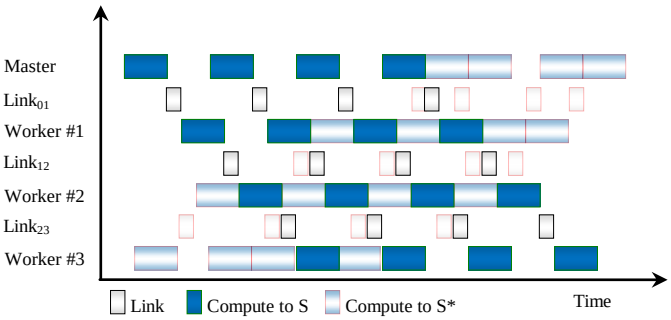

Fig 3: Gantt chart for the execution of PNWH with a master and three workers.

.

3 Experimental measurements

The next tables report measurements taken both, in single and multi-processor systems. The measurements for a single processor system are taken from runs of the NW method. In the case of multi–processor system the measurements are taken from runs of the parallel method introduced within this paper. The labels in the boot tables are Sequences Length (SL), Execution Time (ET), Memory Usage (MU), Parameters (P), and Number of Processors (NP)

Table 1. Measurements on a single processor system

S L (m × n)	E T (s)	M U (mb)
11466x11250	8.01	246.16
18000x18000	19.94	607.39
21894x21924	29.83	917.35
26134x2600	82.69	>917.35

Table 1. Measurements on a multi-processor system

S L (m × n)	P x, y, and v	E T (s)	M U (mb)	N P
11466x11250	225,182,64	1.04	23.41	45
18000x18000	250,180,100	2.38	51.19	72
21894x21924	252,179,123	2.70	43.97	87
26134x2600	252,179,146	4.85	51.86	104

4 Conclusions

A parallel algorithm for finding the optimal alignment of large protein sequences has been introduced. The method is based on variant of the Needleman-Wunsch algorithm and uses some ideas first presented by Hirschberg in a method for identifying longest common subsequences. The method introduced in this paper finds intermediate points between memory space requirements and computations, thus reconciling the two extremes: the excessive memory usage of Needleman-Wunsch and the over computations in Hirschberg method. The result obtained from an actual implementation of the method support the validity of our claims.

5 Acknowledments

The authors want to thank Mrs. Melissa N. Martinez, who is currently an undergraduate student of the Mathematics Department of the University of Puerto at Cayey, for her valuable help in the conduction of experiments that lead to the design of the parallel method presented in this paper.

6 References

[1] Brocchieri L, Karlin S (2005-06-10). "Protein length in eukaryotic and prokaryotic proteomes". Nucleic Acids Research 33 (10): 3390–3400.

[2] Lodish H, Berk A, Matsudaira P, Kaiser CA, Krieger M, Scott MP, Zipurksy SL, Darnell J. (2004). Molecular Cell Biology 5th ed. WH Freeman and Company: New York, NY.

[3] Fulton A, Isaacs W. (1991). "Titin, a huge, elastic sarcomeric protein with a probable role in morphogenesis". Bioessays 13 (4): 157–61.

[4] Hirschberg D, "A Linear Space Algorithm for Computing Maximal Common Subsequences", *Communications of ACM.*, vol. 18, No. 6, pp. 341 – 343, 1975.

[5] S.B. Needleman, C.D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins". Mol. Biol. 48(3):443-453.1970.

[6] Neil C. Jones and Pavel A. Pevzner, *An Introduction to Bioinformatics Algorithms* MIT Press, 2004.

[7] L. de la Torre and J. Seguel, "A Comparison of Two Master-Worker Scheduling Methods", HPCC'09, 2009.

[8] D. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific, Belmont, Mass., 1996.