Local Sequence Alignment Using Parallel Memory-Efficient Dynamic Programming

Lukáš Pichl[1], Misako Arai[2], Ken Hanabusa[3], Takafumi Hayashi[2]

[1]Division of Natural Sciences, International Christian University, Mitaka, Tokyo, 181-8585 Japan

[2]Department of Computer Software, University of Aizu, Ikki, Aizuwakamatsu, 965-8580 Japan

[3]HIT Corporation, 4-33-4 Nishi-Shinjuku, Shinyuku-ku, Tokyo, 160-0023 Japan

Abstract

Alignment of biological sequences is a generalization of the classical computer science problem of the longest common subsequence of two strings, which allows for the possibility of character mismatches, string gaps in alignment, and typically deals with extremely large strings on the order of genome sizes. In spite of a rigorous alignment solution based on dynamic programming - which also posses a high sensitivity in practical biological applications - faster but less sensitive heuristics is used up to date (FASTA, BLAST, etc.). This works elaborates on the memory-efficient version of the dynamic programming algorithm by designing and testing a parallel variant of this algorithm. The local alignment problem is first suitably partitioned in parallel and then distributed among all processors independently. With such an approach, it is demonstrated that a lookup of an ALU repeat in entire human genome can take as low as 5 minutes on customarily available cluster machines, for instance.

Keywords: pairwise local sequence alignment, linear memory, optimal scaling, distributed computing.

1 Introduction

Since the famous discovery of DNA double helix structure in 1953 by Watson and Crick, molecular biology produced large amounts of data and gave rise to new fields in computer science, such as bioinformatics [1, 2] or advanced database technologies. The need to analyze DNA sequence data on computers resulted in the development of new sequence alignment algorithms. Owing to the length of DNA sequences, optimization of these algorithms for time and memory space complexity is critical. A simple version of the problem, the longest common subsequence of two strings, can be solved with the dynamic programming algorithm (Bellman 1950 [3]); both the computation time and memory space required are proportional to the product of the length of the two sequences, the former being a bottleneck for human genome processing on modest cluster computers.

The space-efficient algorithms for pairwise string alignment including gap scoring schemes were published in 1990s [4, 5] with memory space required being proportional to the sum of sequence lengths. Still, they did not become customary in bioinformatics because of the quadratic time complexity of the underlying dynamic programming (DP [3, 6]) approach. Faster heuristics at various level of sophistication (FASTA [7], BLAST [8] etc.) has been used in practice instead. Recently, Rajko and Aluru [9] proposed the first parallel algorithm for *global* alignment (two strings of length m and n), which is optimally scalable *both* in space as $O((m+n)/p)$ and time as $O(mn/p)$ and p is the number of processors. It is therefore expected that the distributed DP algorithms will be used in practice in near future. Here we deal with the local comparison case left which was not analyzed by Rajko and Aluru [9]. Our approach is based on a similarity with quicksort [10] and mergesort [11] strategies and results in a non-recursive partitioning, simpler than the algorithm in [9]. The efficiency of the proposed distributed algorithm is demonstrated by finding all ALU repeats (gene positioning marks) in the entire human genome for several matching thresholds. The parallel program takes about 5 min on 128 P4 CPUs (P4HT, 2.8GHz), proving that a rigorous local sequence matching in human DNA is practically tractable.

In Section 2 we review the memory-efficient algorithm for local sequence alignment. Section 3 designs the distributed algorithm and provides benchmarks and results. Concluding remarks in Section 4 close the paper.

2 Local Alignment in $O(M)$

A general pairwise sequence alignment is one of the fundamental operations in computational biology which has evolved from the common sequence comparison problems in computer science. Instead of letter-wise identity, alignment scoring systems which consist of an alphabet Σ (and a gap γ) are used. A special case of sequence alignment is string

editing, a letter-wise transform of one sequence to another which consists of character insertion, deletion, and substitution operations. The local alignment score of two sequences $s(1,\cdots,i)$ and $t(1,\cdots,j)$ ($i \le m$ and $j \le n$) obeys

$$a[i,j] = MAX \left\{ \begin{array}{l} a[i-1,j-1] + p(i,j) \\ a[i,j-1] + g \\ a[i-1,j] + g \\ 0 \end{array} \right\} \quad (1)$$

where $a[i,j]$ is similarity score of $\{s_k\}_{k=1}^{m}$ and $\{t_l\}_{l=1}^{n}$, $p(i,j)$ is the matching score for the letters $s(i)$ and $t(j)$ in alphabet Σ, and g is the penalty score for the gap γ. A matrix of reversed links from $a[i,j]$ to $a[i-1,j-1]$ (alignment of $s(i)$ and $t(j)$), or $a[i,j-1]$ (alignment of $t(j)$ with γ in s) or $a[i-1,j]$ (alignment of $s(i)$ with γ in t) determines the traceback starting from the highest score value in the matrix a, say $a[i_\mu, j_\mu]$. Let us note that the recursion is initialized without a penalty for gap alignment, $a[0,i] = a[i,0] = 0 \times i$. The total time complexity of this algorithm behaves as $O(mn)$. The similarity score for two sequences s and t, based on the dynamic programming algorithm in Eq. (1) also requires $O(mn)$ memory for storing optimal alignment score in $a[m,n]$. The design of the first phase of a memory efficient DP algorithm [4, 5] is straightforward - it builds on the recursive column-by-column fill of the matrix a: in order to obtain $a[-,j]$, only the previous column, $a[-,j-1]$, suffices. Therefore these scores can be stored in linear memory (1D array $a[i] = a[i,-]$) which iteratively combines in a-scores columns j and $j-1$ for $j = 1,2,\cdots,n$. Hence the memory efficient version requires only $O(m+n)$ memory for storing the scores [1],

Algorithm 1 Score(s,t,a[])
$Input : s,t$ $(sequence)$
$Output : a$ $(vector)$
$m \leftarrow |s|$
$n \leftarrow |t|$

$for(j = 0; j \le n; j++)\{$ $a[j] \leftarrow j \times 0$ $\}$
$for(i = 1; i \le m; i++)\{$
$old \leftarrow a[0]$
$a[0] \leftarrow i \times g$
$for(j = 1; j \le n; j++)\{$
$temp \leftarrow a[j]$
$a[j] \leftarrow MAX \left\{ \begin{array}{l} old + p(i,j) \\ a[j-1] + g \\ a[j] + g \\ 0 \end{array} \right\}$
$old \leftarrow temp\}\}$

The principal idea of sequence alignment in linear memory, the recursive traceback, is due to Hirschberg [12]. It resembles the partitioning procedures common in quicksort [10] and mergesort [11]. While the mergesort algorithm uses a constant data partition, quicksort is based on a data-dependent one. Also the optimal alignment of the two sequences is found by recursive subdivision of the problem into smaller substances. The partition points define position of aligned (or sorted) data. The first sequence (s) is divided exactly in the middle (mergesort), while the partitioning position in the second sequence (t) must be computed based on the data in both sequences. The procedure then recurs in the left and right parts of the partitioned sequence pair. In fact, the parallel implementation of quicksort [13] provides a useful insight how to parallelize the alignment problem. The binary partitioning of sequence s results in an alignment of character $s(i)$ either with some character $t(j)$ or gap γ. The optimal alignment score O thus recurs as:

$$O \left| \begin{array}{c} s_{1,m} \\ t_{1,n} \end{array} \right| = O \left| \begin{array}{c} s_{1,i-1} \\ t_{1,x_j} \end{array} \right| + r_{i,j} + O \left| \begin{array}{c} s_{i+1,m} \\ t_{j+1,n} \end{array} \right| \quad (2)$$

where either $x_j = j-1$ and $r_{ij} = p(i,j)$ in case of (mis)match, or $x_j = j$ and $r_{ij} = g$ for gap insertion. Thus either character-character or character-gap matching point j is computed by Algorithm 2 ($Align$). The Algorithm 1 ($Score$) is referred to as Pre_Score. If the indicates are inverted, $i \to m-i$ and $j \to n-j$, it is referred to as Suf_Score. Following [1], we assume $m \ge n$.

Algorithm 2 Align(x,y,z,w,i,f)
$Input : s,t$ $(sequence)$
x,y,z,w,i $(index)$
$Output : o\_s, o\_t$ $(vector)$
i,f $(index)$

$if(s[x..y] = empty$ or $t[z..w] = empty)\{$
$f \leftarrow i + max(y-x, w-z)\}$
$else\{$
$i \leftarrow (x+y)/2$
$Pre\_Score(s[x..i-1], t[z..w], p\_sim)$
$Suf\_Score(s[i+1..y], t[z..w], s\_sim)$
$pos \leftarrow z-1$
$type \leftarrow SPACE$
$v \leftarrow p\_sim[z-1] + g + s\_sim[z-1]$
$for(j = z; j \le w; j++)\{$
$if(p_sim[j-1] + p(i,j) + s\_sim[j] > v)\{$
$pos \leftarrow j$
$type \leftarrow SYMBOL$
$v \leftarrow p\_sim[j-1] + p(i,j) + s\_sim[j]$
$\}$
$if(p\_sim[j] + g + s\_sim[j] > v)\{$
$pos \leftarrow j$
$type \leftarrow SPACE$
$v \leftarrow p\_sim[j] + g + s\_sim[j]$
$\}$
$\}$
$if(type = SPACE)\{$
Align$(x, i-1, z, pos, i, mid)$

$o\_s[mid] \leftarrow s[i]$
$o\_t[mid] \leftarrow SPACE$
Align$(i+1, y, pos+1, w, mid+1, f)\}$
$else\{$
Align$(x, i-1, z, pos-1, i, mid)$
$o\_s[mid] \leftarrow s[i]$
$o\_t[mid] \leftarrow t[pos]$
Align$(i+1, y, pos+1, w, mid+1, f)\}\}$

3 Distributed Algorithm

The parallel algorithm for global alignment of two strings (length m and n) proposed at ICCP'2003 [9] optimally scales in memory as $O((m+n)/p)$ and time as $O(mn/p)$, where p is the number of processors. This algorithm is built on a generalization of the alignment problem at the endpoints by computing parameterized associative balanced partitions, after which each processors solves its task in only $O(mn/p^2)$ time. It is the parallel computation of balanced partition in $O(mn/p)$, which accounts for the total complexity of the algorithm.

As for the local alignment case, the computation of $O(mn/p)$ must be again distributed at the same time because of the quadratic time complexity. We propose the following procedure:

1. Partition ansatz: divide s to p equidistant parts and compute the alignment score of each substring against t using Algorithm 1 (independent processes). Update a recursion level counter x_i whenever $a[]$ stays below a threshold value during the column update process. $O(mn/p)$.

2. Partition update: communicate the maximal local partition points x_i among processors i ($x_i \times m/p + O(n)$) for $i = 1, \cdots, p$). Time complexity $O(1)$.

3. Subproblem alignment: compute the balanced local alignment on each processor restricted to $s(x_i, \cdots, x_{i+1})$ using Algorithm 2. Time complexity $O(mn/p)$.

4. Solution of the local alignment acquired as a union of subproblem alignments in $O(1)$.

The threshold score above is determined for each shorter string t of length n as the mean value of the score matrix of t vs. a random string of length n, i.e. the noise level corresponding to zero-similarity alignment. Sample threshold values (maxima in columns of DP scoring matrix) are shown in Fig. 1 for an example discussed below.

3.1 Implementation and results

In case of a human genome [14], the number of base pairs is about $m =\sim 3.2G$bp while the average length of practical interest gene is $n =\sim K$bp.

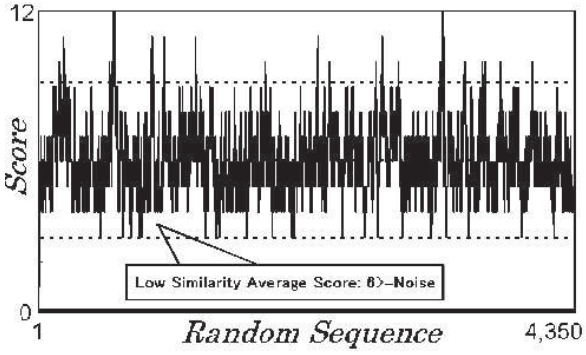

Figure 1: Random sequence alignment with ALU

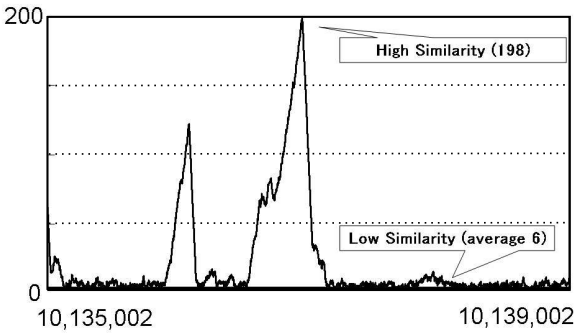

Figure 2: Maximum col. score (C_1 & ALU)

It is therefore the distributed *linear* memory algorithm for local alignment that is indispensable. This is demonstrated by the following computational experiment which uses all chromosomes of human genome except for X and Y (chromosome 1,...,22, sizes given in Table 3.1). The total size of the data is approximately $2.7G$bp ($m =\sim 2.7G$). We select a simple human repeat sequence, ALU, to be searched for in the genome. ALU size equals 434bp ($n = 434$) [15]. The column-wise maximum matching scores in chromosome 1 are shown in Fig. 2. Table 3.1 shows the number of ALU matches (possibly overlapping) in the 22 chromosomes, using three preset sensitivity thresholds. Because of the large negative contribution of the gaps to the similarity score, a very good level of functional similarity is achieved with thresholds as low as 45% of complete match. Figure 3 illustrates one example of such alignment.

Our computational benchmark with Pentium 4 processor at $2.8G$Hz yields 5 min 54 s of CPU time for 128 processors ($p = 128$). The overhead of $2K$bp for each of the about $21M$bp partitions remains within 0.01%.

4 Conclusion

We have discussed the algorithm for memory-efficient pairwise sequence alignment and features of its parallel implementation. An efficient and easy distributed algorithm was designed for the case of gene or pattern matching with local align-

C	$n\%$	*10%*	*30%*	*45%*
1	218,712,726	165,002,573	279,583	65
2	237,043,673	141,741,511	253,062	893
3	193,607,218	112,742,026	167,861	0
4	186,580,523	92,593,428	125,533	209
5	177,524,972	98,807,045	166,885	1,145
6	166,880,540	97,687,090	143,628	0
7	154,546,299	118,874,023	216,878	991
8	141,694,337	82,114,019	151,312	0
9	115,187,714	80,004,325	128,293	44
10	130,710,865	94,598,620	146,423	660
11	130,709,420	79,705,559	147,154	152
12	129,328,332	98,202,348	164,335	332
13	95,511,656	48,505,115	58,206	200
14	87,191,216	60,838,946	105,282	115
15	81,117,055	64,453,761	105,229	166
16	79,890,791	84,740,793	156,107	56
17	77,480,855	94,305,536	178,775	304
18	74,534,531	41,470,103	64,738	228
19	55,780,860	97,641,911	245,868	1,129
20	59,424,990	50,077,073	100,084	737
21	33,924,742	21,347,519	36,795	0
22	34,352,051	39,690,638	55,835	0

Table 1: The Number of matched ALU repeats

ment. Unlike from the global cases, a distributed non-recursive partition is possible. The theoretical overhead is clearly minimal as confirmed in computational experiments with the ALU repeat in human genome. Resulting ALU matching tables can also be relevant in research of DNA mutations. Since a whole genome can be aligned locally against a 10^3 base length pattern in several minutes using the present distributed algorithm on p ($p = 2^7$) processors, it should be possible to test and calibrate results of available heuristics [16] against DP, the rigorous and more sensitive method.

We acknowledge partial support by a JSPS Grant-in-Aid and HIT Corporation, Tokyo.

```
CTTTCCACTGTTTTCTTCCTTCCTCCCTCCCTCCTTCTCTCTCTCCCTCCTTCTCTCC-C-TCCCTT
-GAT-C-CT-CTTGCCT-AGGCCTCCCAAAGTGCTGGGATTACAGGCAAGAGC-CACCACGTCCC-G

TCTCTCCCTTTCTCCTTCCTTCCTTCTTTCCTTCCTTCCTTCTATCCTTCCTTCCTTCCCTCC
CCTCT-AATTTCT-C-TCC-T-CTTCTCTCC-T-C-TCC-T-T-TCC-T--TTCC-T-CTCCTCTCC

CTCCCTTTCTCTCTCTTTCTCCTTCCTTCCTTCCTTCTTTCCTTCCTTCCCTCCCTGCCTCTCCCTC
TTTCC-TT-TC-CTC-TTCT-CTT-CTTGTTTTTTTTCTTTTC-TCCCTCCCTCCCT-CC-CTCCCTC

CCTCCCCCCCCTCCCTCCCTCCCTCCCTTCCTTCCTTCCTCTCTTTCTCTCTTTCTTTCTTTCTCAGA
CCTCCCTCCCT-CCTTCCTTCCTTCCTTCCTTCCTTCCTTCCTTCCTTCCTTCCTTTTTGAGA

CA-ACTCTCACTCTGTCACCCAGGCTGGAGTGCAGTGG-CGCAATGTTGGCTCCCTGCAACCTCTAC
CAGAGTTTTATTCTGTCACCCAGACCTGAGTGCAATGGGCACAATTTTGGCTCACTGCAACCTCCAT

GTCCCAGGTTCAAGCAATTCTTGTGCTTCAGCTTCCCGAGTAGCTGGGACTACAGATGTGTGTCACC
CTCCCCGGTTCAAGTGATTCTTCTGCCTTAGCCTCCCGAATAGCTGGAACTACAGGCACCTGCCACC

ATGCCC-AGCTAATTTTT-GTATTTTTAGTAGAGATGGGTTTTCACCATGTTGGCCAGGTTGGTCTC
ATGCCCCAGCTAATTTTTTTGTATTCTCAGTAGAGATGGGGTTT-ACCATGTTGGCCAGGCTGCTC--
```

Figure 3: Local Alignment (s: C_1 $(10,136,751 - 10,137,213)$ $|s| + gap = 463 + 6 = 469$, t: ALU $|t| + gap = 434 + 35 = 469$, score: 196)

References

[1] J. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology* (PWS Publishing Company ITP, Boston, 1997).

[2] B. Bergeron, *Bioinformatics Computing* (Pearson Education, Inc., Prentice Hall PTR, Indiana, 2003).

[3] R. Bellman, *Dynamic Programming* (University Press, New York, 1957).

[4] E. W. Myers and W. Miller, *Optimal alignments in linear space*, CABIOS **4** (1988) 11-17.

[5] X. Huang, R. C. Hardison and W. Miller, *A space-efficient algorithm for local similarities*, CABIOS **6** (1990) 373-381.

[6] 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** (1970) 443-453.

[7] W. R. Pearson and D. J. Lipman, *Improved tools for biological sequence comparison*, Proc. Natl. Acad. Sci. **85** (1988) 2444-2448.

[8] S. F. Altschul, W. Gish, W. Miller, E. W. Myers and D. J. Lipman, *Basic local alignment search tool*, J. Mol. Biol. **215** (1990) 403-410.

[9] S. Rajko and S. Aluru, *Space and Time Optimal Parallel Sequence Alignments*, ICPP (2003) 39-47.

[10] C. A. R. Hoare, *Quicksort*, Computer Journal **5** (1962) 10-15.

[11] J. Neumann, *First Draft of a Report on the EDVAC* (1945).

[12] D. Hirschberg, *A linear space algorithm for computing maximal common subsequences*, ACM **18** (1975) 341-343.

[13] P. Heidelberger, A. Norton and J. T. Robinson, *Parallel Quicksort Using Fetch-And-Add*, IEEE Transactions on Computers **39** (1990) 133-138.

[14] Nat. Center for Biotechnology Information ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/

[15] DBGET http://www.genome.jp/dbget-bin/www_bget?gb*HSARWG1D1

[16] K. H. Fasman, *Finding genes in DNA with a Hidden Markov Model*, JCB **4** (1997) 127-141.