另一個求最長共同子序列問題的最快心跳式演算法
Another Fastest Systolic Algorithm for the Longest Common Subsequence Problem

林彥君　　　　陳智謙
Yen-Chun Lin　　　　Jyh-Chian Chen

國立台灣科技大學 電子工程技術研究所
Dept. of Electronic Engineering
National Taiwan University of Science and Technology, Taipei 106, Taiwan
yclin@et.ntust.edu.tw　　　george@elc1.lhjc.edu.tw

摘 要

兩字串的最長共同子序列是這兩字串最大長度的共同子序列。在這篇論文裡我們提出一個求最長共同子序列及其長度的心跳式演算法。對長度分別爲 m 和 n 的兩字串，$m \geq n$，只要 $m + 2n - 1$ 步即可求得。當字串循序輸入到含有 n 個處理器的線性心跳式陣列時，該演算法的計算步數爲最小下限。該演算法可修改爲利用廣播的方式，則僅需 $m + n$ 步。

關鍵字：最長共同子序列，多重處理器，平行演算法，心跳式陣列，超大型積體電路

Abstract

A longest common subsequence (LCS) of two strings is a common subsequence of the two strings of maximal length. The LCS problem is to find an LCS of two given strings and the length of the LCS. In this paper, a fast systolic algorithm for the LCS problem is presented. For two strings of length m and n, where m ≥ n, the problem can be solved in m + 2n − 1 time steps. The algorithm achieves the tight lower bound of the computation time when symbols are input sequentially to a linear array of n processors. The systolic algorithm can be modified to take only m + n steps on a multicomputer by using the broadcast operation.

Keywords: longest common subsequence, multicomputer, parallel algorithm, systolic array, VLSI

This research was supported in part by the National Science Council of the R.O.C. under contracts NSC85-2213-E-011-017.

1. Introduction

String C is a subsequence of string A if C is obtained by deleting zero or more symbols from A. For example, *opt* is a subsequence of *computation*. String C is a common subsequence (CS) of strings A and B if C is a subsequence of both A and B. For example, *ac* is a CS of *bcabcb* and *abccb*. A longest CS (LCS) of two strings is a CS of the two strings of maximal length. For example, *abcb* and *bccb* are two LCSs of *bcabcb* and *abccb*. The LCS problem is to find an LCS of two given strings and the length of the LCS (LLCS). Throughout this paper, we assume that the strings in question are $A = A(1)A(2)...A(m)$ and $B = B(1)B(2)...B(n)$, $m \geq n$, and that the LLCS of A and B is p. The LCS problem has been the subject of much research because it can be applied to many areas, such as molecular biology, word processing, pattern recognition, and data compression [2, 5, 12, 15-17].

The time complexity of "equal-unequal" comparison-based approach to solve the LCS problem has been shown to be $\Omega(mn)$ when the number of distinct symbols is not fixed [2]. Many time-optimal sequential algorithms have been devised [3, 5-7, 14, 18]. To obtain even faster solutions for this problem, many parallel algorithms have also been proposed. On the concurrent-read exclusive-write (CREW) parallel random-access machine (PRAM) model, with $mn/\log n$ processing elements (PEs) an efficient algorithm runs in $O(\log m + \log^2 n)$ time [12]. On the same model, the only cost-optimal algorithm uses $mn/(\log^2 n \log\log n)$ PEs and takes $O(\log^2 n \log\log n)$ time when $\log^2 n \log\log n > \log m$; it uses $mn/\log m$ PEs and takes $O(\log m)$ time when $\log^2 n \log\log n \leq \log m$ [12]. However, the CREW PRAM model is not practical for this problem because of the physical limitation in building a machine with more than 100 PEs.

In contrast, the systolic model [8] is more practical for the LCS problem. Systolic algorithms can be implemented on VLSI chips or distributed-memory multicomputers, and these implementations can have much more PEs than PRAMs. In addition, systolic algorithms for the LCS problem can be cost-optimal while using a reasonable number of PEs. A systolic array consists of identical PEs in which data are passed locally and operated on rhythmically. It has the merit of simplicity, regularity, and locality. High performance can be achieved by the concurrent operation of PEs. Robert and Tchuente propose using a 2-D systolic array to solve the LCS problem in $m + 5n - 3$ time steps [15, 16]. Lin uses a linear systolic array of n PEs to solve the problem in $m + 4n - 2$ time steps [10]. Luce and Myoupo solve the problem on a linear array of n PEs in $m + 3n + p - 1$ time steps [13]. Lecroq *et al.* propose using a linear array of n PEs to solve the problem in $m + 2n$ time steps [9]. Lin and Chen devise an even faster linear systolic array of n PEs to solve the LCS problem in $m + 2n - 1$ time steps [11].

In this paper, another fast systolic algorithm that improves on the previous ones for the LCS problem is presented. Although it also uses a linear array of n PEs and takes $m + 2n - 1$ time steps, it is derived with a relation different from the one used in [11]. The new algorithm performs simpler operations, and thus takes less time for a time step. Possible implementation refinements of our systolic algorithm are also presented. It is noteworthy that a straightforward modification of the algorithm is suited to multicomputer implementation and requires only $m + n$ steps.

Section 2 derives the new systolic algorithm. Section 3 gives an example to help understand the algorithm. Section 4 compares our algorithm with previous ones. Section 5 gives possible implementation refinements. Section 6 concludes this paper.

2. Deriving a new systolic algorithm

In this section, we present a systolic algorithm that computes the LLCS and an LCS simultaneously. For ease of presentation, let $LCS(i, j)$ denote an LCS of $A(1)A(2)...A(i)$ and $B(1)B(2)...B(j)$, and $L(i, j)$ denote the length of $LCS(i, j)$, $1 \le i \le m$, $1 \le j \le n$; in addition, let ε denote the empty string. It is easy to see that the following property exists [5, 9]:

```
For $1 \le i \le m$ and $1 \le j \le n$,
$L(i, 0) = L(0, j) = L(0, 0) = 0$
$LCS(i, 0) = LCS(0, j) = LCS(0, 0) = \varepsilon$
if $A(i) = B(j)$ then
  $L(i, j) = L(i-1, j-1) + 1$
  $LCS(i, j) = LCS(i-1, j-1)B(j)$
else if $L(i, j-1) > L(i-1, j)$ then
  $L(i, j) = L(i, j-1)$
  $LCS(i, j) = LCS(i, j-1)$
else
  $L(i, j) = L(i-1, j)$
  $LCS(i, j) = LCS(i-1, j)$
end if
```

The above property has been used to derive a systolic algorithm for the LCS problem [9]. Based on this property, we give a theorem and obtain a modified property, as follows.

Theorem 1. $LCS(i-1, j-1)$ is equal to the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$.

Proof. Consider the following two cases.

Case 1. If $L(i-1, j-1) < L(i-1, j)$, then $L(i-1, j) = L(i-1, j-1) + 1$. Apparently, it means that $A(i-1) = B(j)$, and $LCS(i-1, j-1)$ is equal to the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$.

Case 2. If $L(i-1, j-1) = L(i-1, j)$, clearly $LCS(i-1, j-1) = LCS(i-1, j)$. Thus, $LCS(i-1, j-1)$ is equal to the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$. Q.E.D.

From Theorem 1, we can replace

$LCS(i, j) = LCS(i-1, j-1)B(j)$

in the above property with

$LCS(i, j) =$ the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$ appended by $B(j)$.

We will also replace the greater-than test in the property, i.e.,

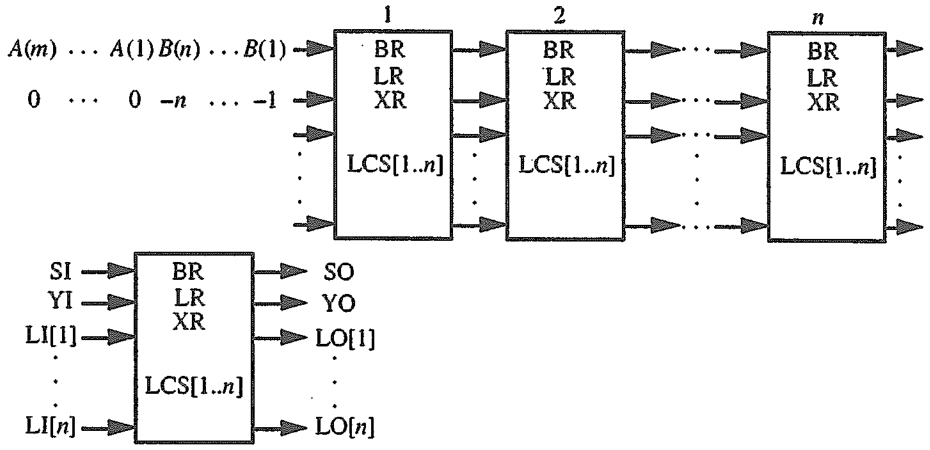
else if $L(i, j-1) > L(i-1, j)$ then,

with the greater-than-or-equal-to test

else if $L(i, j-1) \ge L(i-1, j)$ then.

Although either test is correct, as will be explained, this change contributes to the conception of the new algorithm shown in Fig. 1, which computes $L(m, n) = p$ and $LCS(m, n)$ in $m + 2n - 1$ time steps. The comments in the figure describe the meanings of operations performed by PE j, $1 \le j \le n$. The systolic algorithm uses n PEs, each containing $n + 3$ registers: BR, LR, XR, LCS[1],..., LCS[n]. Each PE has $n + 2$ input ports, named SI, YI, LI[1],..., LI[n], and $n + 2$ output ports, called SO, YO, LO[1],..., LO[n]. For ease of presentation, LCS[1] through LCS[n] will be abbreviated to LCS[1..n]; LI[1] through LI[n] to LI[1..n]; and LO[1] through LO[n] to LO[1..n].

Before explaining how the algorithm works, we first examine the input and output data. The SI input stream contains symbols of B and A. The YI input stream consists of n negative integers -1,..., $-n$ and m 0-valued flags; the negative integer $-i$ indicates that the accompanying symbol in the SI input stream is $B(i)$, while the jth 0-valued flag represents $L(j, 0) = 0$ and implies that the accompanying SI input is $A(j)$. As specified in Fig. 1, the YI input to a PE



```
{Operations performed by PE $j$, $1 \le j \le n$}
1:  if YI < 0 then              {SI belongs to string $B$}
2:      if YI = −1 then                 {SI = $B(j)$}
3:          BR := SI                        {load $B(j)$ into PE $j$}
4:          LR := 0                         {$L(0, j) = 0$}
5:          XR := 0                         {$L(0, j-1) = 0$}
6:      else                            {SI = $B(k)$, where $k > j$}
7:          SO := SI                        {forward $B(k)$}
8:          YO := YI + 1
9:      end if
10: else                        {SI belongs to string $A$, say, $A(i)$}
11:     if SI = BR then                 {$A(i) = B(j)$}
12:         LR := XR + 1                    {$L(i, j) = L(i-1, j-1) + 1$}
13:         LCS[LR] := BR                   {the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$ in
                                             LCS[1..XR] appended by $B(j)$ to make $LCS(i, j)$}
14:     else                            {$A(i) \ne B(j)$}
15:         if YI ≥ LR then                 {$L(i, j-1) \ge L(i-1, j)$}
16:             LR := YI                        {save $L(i, j-1)$ as $L(i, j)$}
17:             LCS[1..LR] := LI[1..LR]         {save $LCS(i, j-1)$ as $LCS(i, j)$}
18:         end if
19:     end if
20:     SO := SI                        {send out $A(i)$}
21:     XR := YI                        {save $L(i, j-1)$}
22:     YO := LR                        {send out $L(i, j)$}
23:     LO[1..LR] := LCS[1..LR]         {send out $LCS(i, j)$}
24: end if
```

Fig. 1. Systolic algorithm for the LCS problem.

affects the operations performed by the PE. The LI[1..n] input ports of the leftmost PE in Fig. 1 are shown simply for consistency among all PEs; no input data are needed for them. The LLCS p can be obtained at the YO port of the rightmost PE at the ($m+2n-1$)th time step, and an LCS can be retrieved simultaneously from the LO[1..n] ports, or more precisely from LO[1..p].

Next, consider the roles played by the registers and the meanings of the I/O data of each PE to understand how the whole array can produce an LCS and the LLCS. Register BR in PE j is used to hold $B(j)$. Table 1 summarizes the meanings of values in registers BR, LR, XR, and LCS[1..LR]; in inputs YI and LI[1..YI]; and in outputs YO and LO[1..YO] of PE j at the time step $A(i)$ is sent to the PE. Note that although registers LCS[1..n] are available for storing $LCS(i, j)$, it is quite probable that only a portion of them are needed; specifically, as register LR keeps $L(i, j)$, only LCS[1..LR] are needed to hold $LCS(i, j)$.

Table 1. Meanings of values in registers and values transferred through I/O ports in the step $A(i)$ is sent to PE j.

Register and I/O values	Before inputs are processed	At the end of the step
BR	$B(j)$	$B(j)$
LR	$L(i-1, j)$	$L(i, j)$
XR	$L(i-1, j-1)$	$L(i, j-1)$
LCS[1..LR]	$LCS(i-1, j)$	$LCS(i, j)$
YI	$L(i, j-1)$	–
LI[1..YI]	$LCS(i, j-1)$	–
YO	–	$L(i, j)$
LO[1..YO]	–	$LCS(i, j)$

The operations of PE j, $1 \le j \le n$, are explained in the following. If the YI input is -1 (line 2), three registers are initialized. Specifically, the SI input, which is symbol $B(j)$, is loaded into the BR register (line 3), and the LR and XR registers are set to 0 to represent $L(0, j) = 0$ and $L(0, j-1) = 0$, respectively (lines 4-5); otherwise, when YI < -1, the SI input, which is $B(k)$, $k > j$, is sent out through the SO port (line 7), and the YI input plus 1 is sent out through the YO port (line 8) so that $B(k)$ can be loaded into the BR register of the kth PE.

If YI ≥ 0 and thus the SI input is a symbol of string A, say, $A(i)$, PE j computes $L(i, j)$ and $LCS(i, j)$. First, consider computing $L(i, j)$. Note that as shown in Table 1, the YI input carries the value $L(i, j-1)$, and registers LR and XR initially contain $L(i-1, j)$ and $L(i-1, j-1)$, respectively. If $A(i) = B(j)$, register LR is loaded with $L(i-1, j-1) + 1$, or $L(i, j)$ (lines 11-12). In case $A(i) \ne B(j)$, if $L(i, j-1) \ge L(i-1, j)$, register LR is also loaded with $L(i, j)$, which is $L(i, j-1)$ from the YI input (lines 15-16); if $L(i, j-1) < L(i-1, j)$, no operations are needed because LR has already contained $L(i-1, j)$, which is also $L(i, j)$.

Next, consider finding $LCS(i, j)$. If $A(i) = B(j)$, because registers LCS[1..XR], or LCS[1..LR−1], contains the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$, LCS[1..LR] contains $LCS(i, j)$ after $B(j)$ is loaded into LCS[LR] (line 13). In case $A(i) \ne B(j)$, if $L(i, j-1) \ge L(i-1, j)$, registers LCS[1..LR] are loaded with $LCS(i, j-1)$ from the LI[1..LR] inputs (line 17); if $L(i, j-1) < L(i-1, j)$, LCS[1..LR] need not be modified because $LCS(i, j)$ equals $LCS(i-1, j)$, which has already resided in registers LCS[1..LR].

Finally, when YI ≥ 0, some more operations are needed to output data and save a temporary value. The symbol $A(i)$ is sent out through port SO (line 20). The YI input, or $L(i, j-1)$, is saved in register XR (line 21), and will become $L(i-1, j-1)$ when the next $A(i)$ is input in the next time step. Further, $L(i, j)$ in register LR and $LCS(i, j)$ in LCS[1..LR] are sent to the right through ports YO and LO[1..LR], respectively (lines 22-23).

As already noted, it is necessary to test $L(i, j-1) \ge L(i-1, j)$ (line 15); testing $L(i, j-1) > L(i-1, j)$ will not make a working algorithm. Actually, either test is good for computing the LLCS. However, because there may be more than one LCS, when $L(i, j-1) = L(i-1, j)$, it is crucial to save $LCS(i, j-1)$ rather than $LCS(i-1, j)$ as $LCS(i, j)$ in LCS[1..LR] (line 17). We now examine the effects of the two saves. First, consider the correct save: in the next time step, LCS[1..XR] keeps $LCS(i-1, j-1)$, which from Theorem 1 is the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$. Therefore, in this time step, if $A(i) = B(j)$, line 13 is appropriate. Next, consider the case $LCS(i-1, j)$ is wrongly saved in LCS[1..LR]. In the next time step, if $A(i) = B(j)$, the fact that LCS[1..XR] contains $LCS(i-2, j)$ will make line 13 pointless.

3. An example

Fig. 2 shows some initial snapshots of our algorithm for an example of $A = bcabcb$ and $B = abccb$. The first snapshot is taken just before the first time step begins. An LCS $abcb$ and $p = 4$ can be obtained at ports LO[1..4] and YO, respectively, of the rightmost PE at time step $m + 2n - 1 = 15$.

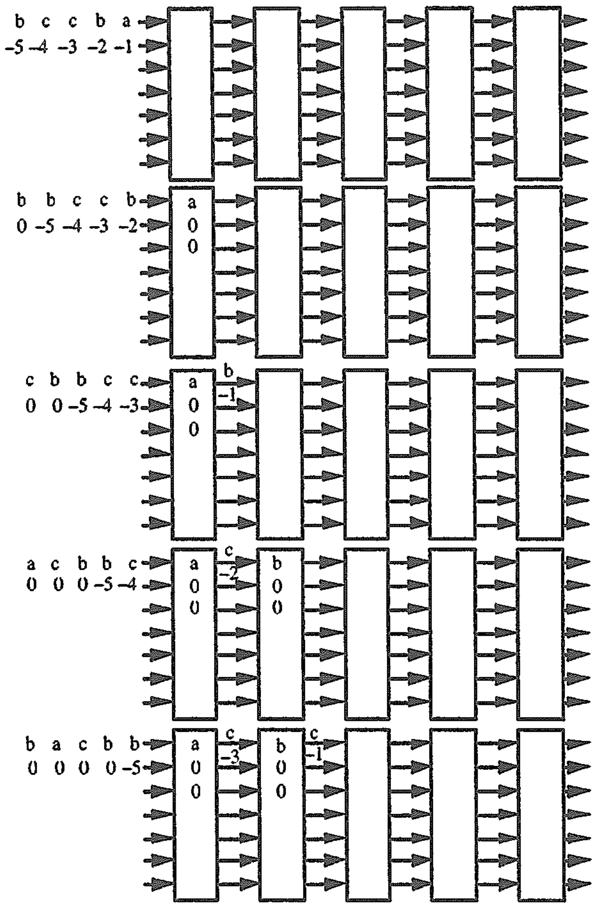


Fig. 2. Initial snapshots for A = *bcabcb* and B = *abccb*.

4. Comparison with previous systolic algorithms

The time complexity of systolic algorithms is measured by the number of time steps required to obtain results. The systolic algorithm presented in this paper takes $m + 2n - 1$ time steps, and is one of the two fastest systolic algorithms for the LCS problem. In fact, as it takes $m + 2n - 1$ time steps for the last one of the $m + n$ sequentially input symbols to be sent to the nth PE, for linear systolic arrays of n PEs that input $m + n$ symbols sequentially as our systolic algorithm does, the tight lower bound of the time complexity is exactly $m + 2n - 1$ time steps. Note that in each time step at most six assignments are executed. The other fastest systolic algorithm, which also takes $m + 2n - 1$ time steps, in a time step performs at most seven assignments and never requires fewer assignments than the new algorithm [11]. The second fastest systolic algorithm takes $m + 2n$ time steps; it requires at most six assignments in each time step [9]. Thus, the new algorithm should be more desirable than the others. Note that these algorithms are all cost-optimal.

Either of the two fastest algorithms needs in each PE one register more than the second fastest algorithm; however, either of the two fastest algorithms uses one fewer input port and one fewer output port for each PE to transfer fewer data. Furthermore, either of the two fastest algorithms requires only two input streams, while the second fastest needs three input streams.

For the two reasons that follow, requiring fewer I/O ports is desirable when systolic algorithms are realized on a chip or with the wafer-scale integration technology. First, links between PEs may occupy very much space and become very expensive [4]. Second, VLSI components have limited number of I/O pins.

When algorithms are implemented on a multicomputer, requiring fewer I/O ports to send fewer data is also advantageous. As PEs have limited number of physical links among them, it is faster to pass fewer data.

5. Refinements of our algorithm

We have assumed that each PE contains registers LCS[1..n], input ports LI[1..n], and output ports LO[1..n]. However, not every PE uses all of the registers and ports. It is easy to see that PE i, $1 \le i \le n$, uses at most i registers, LCS[1..i], for storing an LCS of $A(1)A(2)...A(m)$ and $B(1)B(2)...B(i)$. Thus, PE i needs at most i output ports LO[1..i] to send out the LCS, and uses at most $i - 1$ input ports LI[1..i–1] to receive an LCS of $A(1)A(2)...A(m)$ and $B(1)B(2)...B(i-1)$.

When implemented on a multicomputer, such as the IBM SP2 [1], the systolic algorithm can be modified to take advantage of the scatter operation or the broadcast operation. A scatter operation can send $B(j)$ to PE j, $1 \le j \le n$, in the first step. Alternatively, a broadcast operation can send the string B to every PE in the first step. In the same first step, $B(j)$ is retrieved and saved in register BR of PE j, and registers LR and XR are initialized to 0 to represent $L(0, j) = L(0, j-1) = 0$. After the first initialization step, symbols of A can be sent to the array step by step. At the $(i+1)$th step, the ith YI input flag accompanying $A(i)$ is 0, representing $L(i, 0) = 0$ for $1 \le i \le m$. Beginning at the second step, the PE operations performed in each step are exactly the same as those specified by lines 11-23 in Fig. 1. Although the first step should take longer than the other steps, the algorithm now takes only $m + n$ steps. Note that the scatter operation sends to each PE only $1/n$ of the amount of data sent by the broadcast operation, thus one may expect the scatter operation to be faster than the broadcast operation; however, the scatter operation is experimentally slower than the broadcast operation

[19].

6. Conclusion

We have presented a systolic algorithm for solving the LCS problem. The algorithm takes $m + 2n - 1$ time steps, which is the tight lower bound for systolic algorithms that use a linear array of n PEs to process $m + n$ sequentially input items. The algorithm is thus not only cost-optimal but also the fastest when symbols are input sequentially to a linear systolic array of n PEs. Moreover, compared with the other fastest systolic algorithm, it may take less time but never takes more time in a time step. It is suited to both VLSI and multicomputer implementations. In fact, for implementation on multicomputers, our algorithm can be modified to take advantage of the broadcast operation and takes only $m + n$ steps.

References

[1] T. Agerwala, *et al.*, SP2 system architecture, *IBM Syst. J.* **34** (2) (1995) 152-184.

[2] A. Aho, D. Hirschberg and J. Ullman, Bounds on the complexity of the longest common subsequence problem, *J. ACM* **23** (1) (1976) 1-12.

[3] A. Apostolico, S. Browne and C. Guerra, Fast linear-space computations of longest common subsequences, *Theoretical Comput. Science* **92** (1992) 3-17.

[4] Y. Feldman and E. Shapiro, Spatial machines: A more realistic approach a parallel computation, *CACM* **35** (10) (1992) 61-73.

[5] D.S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *CACM* **18** (6) (1975) 341-343.

[6] D.S. Hirschberg, Algorithms for the longest common subsequence problem, *J. ACM* **24** (4) (1977) 664-675.

[7] S.K. Kumar and C.P. Rangan, A linear-space algorithm for the LCS problem, *Acta Inform.* **24** (1987) 353-362.

[8] H.T. Kung, Why systolic architectures?, *IEEE Computer* **15** (1) (1982) 37-46.

[9] T. Lecroq, G. Luce and J.F. Myoupo, A faster linear systolic algorithm for recovering a longest common subsequence, *Inform. Process. Lett.* **61** (3) (1997) 129-136.

[10] Y.C. Lin, New systolic arrays for the longest common subsequence problem, *Parallel Comput.* **20** (9) (1994) 1323-1334.

[11] Y.C. Lin and J.C. Chen, An even faster systolic algorithm for the longest common subsequence problem, in: *Proc. 1997 Workshop on Distributed System Technologies and Applications* (1997) 603-610.

[12] M. Lu and H. Lin, Parallel algorithms for the longest common subsequence problem, *IEEE Trans. Parallel Distributed Syst.* **5** (8) (1994) 835-848.

[13] G. Luce and J.F. Myoupo, An efficient linear systolic algorithm for recovering longest common subsequences, in: *Proc. IEEE Int. Conf. on Algorithms and Architectures for Parallel Processing* (1995) 20-29.

[14] N. Nakatsu, Y. Kambayashi and S. Yajima, A longest common subsequence algorithm suitable for similar text strings, *Acta Inform.* **18** (1982) 171-179.

[15] P. Quinton and Y. Robert, *Systolic Algorithms & Architectures.* (Prentice Hall International, Hertfordshire, UK, 1991).

[16] Y. Robert and M. Tchuente, A systolic array for the longest common subsequence problem, *Inform. Process. Lett.* **21** (4) (1985) 191-198.

[17] D. Sankoff and J.B. Kruskal, *Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison.* (Addison-Wesley, Reading, MA, 1983).

[18] R.A. Wagner and M.J. Fischer, The string to string correction problem, *J. ACM* **21** (1974) 168-173.

[19] Z. Xu and K. Hwang, Modeling communication overhead: MPI and MPL performance on the IBM SP2, *IEEE Parallel & Distributed Technology* 4 (1) (1996) 9-23.