EXTRACTING A LONGEST COMMON SUBSEQUENCE BY A SYSTOLIC ALGORITHM ON MULTICOMPUTERS WITH FEWER PROCESSORS

Yen-Chun Lin and *Jih-Wei Yeh*

Department of Electronic Engineering
National Taiwan University of Science and Technology, Taipei 106, Taiwan
E-mail: yclin@computer.org

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. The LCS problem has been the subject of much research because it can be applied to many areas. In this paper, a scalable systolic algorithm is presented. For two given strings of length m and n, where $m \geq n$, the systolic algorithm can solve the LCS problem in $m + 2r - 1$ (respectively $n + 2r - 1$) time steps with $r < n/2$ (respectively $r < m/2$) processors. Experimental results show that the algorithm has satisfactory performance on the IBM SP2 multicomputer.

1. Introduction

If string C is obtained by deleting zero or more symbols from string A, then C is a subsequence of A. For example, *ccb* is a subsequence of *bcabcb*. String C is a common subsequence (CS) of strings A and B if C is a subsequence of both A and B. For example, *ccb* is a CS of *bcabcb* and *abccb*. A longest CS (LCS) is a CS of the two strings with maximal length. Two strings may have more than one LCS. For example, *abcb* and *bccb* are two LCSes of *bcabcb* and *abccb*. The LCS problem is to find an LCS of two given strings and the length of the LCS (LLCS). Unless otherwise stated, we assume that the strings in question are $A = A(1)A(2)\ldots A(m)$ and $B = B(1)B(2)\ldots B(n)$, $m \geq n$, and that the LLCS of A and B is p. The LCS problem can be applied to many areas, such as molecular biology, word processing, pattern recognition, and data compression [6, 7, 17, 24-26].

The sequential time complexity of the LCS problem has been shown to be $\Omega(mn)$ when the number of distinct symbols is not fixed [2]. Sequential algorithms achieving this time bound can be found in [3, 7-9, 21, 27]. Since m and n are usually very large for typical applications, parallel algorithms for solving the LCS problem are desirable. On the concurrent-read exclusive-write (CREW) parallel random access machine (PRAM), with $mn/\log n$ processing elements (PEs) an efficient algorithm runs in $O(\log m + \log^2 n)$ time [17]. On the same model, a cost-optimal algorithm uses $mn/\max\{\log m, \log^2 n \log\log n\}$ PEs and takes $O(\log m + \log^2 n \log\log n)$ time [17]. Because it is infeasible to build a PRAM with more than a few tens of PEs, the PRAM models are not practical for this problem when m and n are large.

In contrast, the systolic model [10] is practical for the LCS problem. A systolic architecture is a regular array of identical PEs. It can be implemented with VLSI chips or a distributed-memory multicomputer that has many more PEs than PRAMs. All PEs operate concurrently to achieve high performance.

Many systolic algorithms for the LCS and related problems have been proposed. Robert and Tchuente propose using a 2-D systolic array to solve the LCS problem in $m + 5n - 3$ time steps [23, 24]. Lin uses a linear systolic array of n PEs to solve the problem in $m + 4n - 2$ time steps [13]. Luce and Myoupo solve the problem on a linear array of n PEs in $m + 3n + p - 1$ time steps [18]. Lecroq *et al.* propose using a linear array of n PEs to solve the problem in $m + 2n$ time steps [12]. Lin and Chen devise two systolic algorithms that use a linear array of n PEs to solve the problem in $m + 2n - 1$ time steps, achieving the exact lower bound on the number of time steps when the $m + n$ symbols are input sequentially [14, 15]. Lin and Yeh introduce a systolic algorithm that uses $\lceil n/2 \rceil$ PEs and takes $m + 2\lceil n/2 \rceil - 1$ time steps [16], in which two symbols of string B are input at a time. It should be noted that no existing automatic approaches can be used to synthesize LCS systolic algorithms [20]. Algorithms for two extensions of the LCS problem can be found in [19, 20].

Given a systolic algorithm requiring a linear array of n PEs, the locally sequential globally parallel, or coalescing, method and the locally parallel globally sequential, or cut-and-pile, technique can be applied to obtain a new systolic algorithm using fewer PEs to solve the same problem [11, 22]. In this paper, we use a new method to obtain an efficient systolic algorithm that solves the LCS problem on a linear array of a fixed number of PEs. If $r < n/2$ (respectively $r < m/2$) PEs are available, the array can receive $\lceil n/r \rceil$ (respectively $\lceil m/r \rceil$) symbols of string B (respectively A) at a time and solve the LCS problem in $m + 2r - 1$ (respectively $n + 2r - 1$) time steps. When implemented on multicomputers, since this algorithm takes fewer time steps than previous systolic algorithms, it can be the fastest, although it involves more computations than the others in a time step.

In Section 2, we present new properties concerning the LCS problem. Our new systolic algorithm is derived from these properties. In Section 3, we introduce the new systolic algorithm for solving the LCS problem and give an example to illustrate the algorithm. In Section 4, we present experimental results. Section 5 concludes this paper.

2. Some properties

Let $LCS(i, j)$ denote an LCS of $A(1)A(2)\ldots A(i)$ and $B(1)B(2)\ldots B(j)$, and $L(i, j)$ denote the length of $LCS(i, j)$, $1 \le i \le m$, $1 \le j \le n$. When $A(1)A(2)\ldots A(i)$ and $B(1)B(2)\ldots B(j)$ have only one LCS, $LCS(i, j)$ represents it. However, when there are more than one LCS, $LCS(i, j)$ represents an appropriate LCS of $A(1)A(2)\ldots A(i)$ and $B(1)B(2)\ldots B(j)$ to make the context meaningful. In addition, let $L(i, 0) = L(0, j) = L(0, 0) = 0$ and $LCS(i, 0) = LCS(0, j) = LCS(0, 0) = \varepsilon$, where ε denotes the empty string. By combining two properties presented in [7, 12], we have the following property:

Property 1.

```
For $1 \le i \le m$ and $1 \le j \le n$,
  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 $L(i, j-1) \ge L(i-1, j)$ then
        $L(i, j) = L(i, j-1)$
        $LCS(i, j) = LCS(i, j-1)$
  else   $\{L(i, j-1) < L(i-1, j)\}$
        $L(i, j) = L(i-1, j)$
        $LCS(i, j) = LCS(i-1, j)$
  end if
```

Moreover, it has been shown in [15] that

$LCS(i-1, j-1)$
$= \text{the first } L(i-1, j-1) \text{ symbols of } LCS(i-1, j)$.

Consequently, the equation

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

in Property 1 can be replaced by

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

Thus, Property 1 can be modified to become the following property.

Property 2.

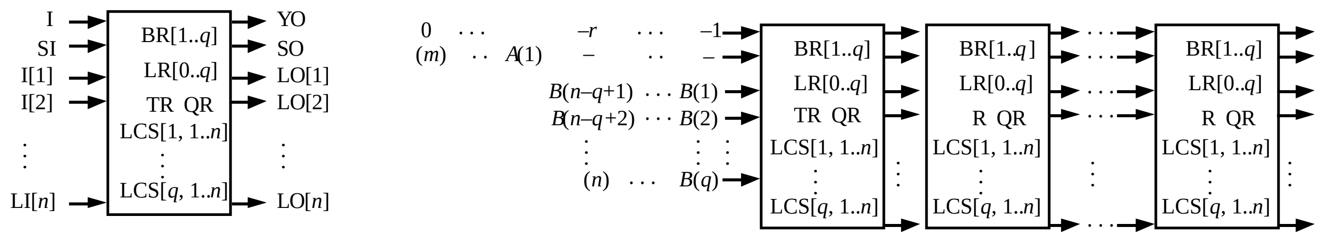
```
For $1 \le i \le m$ and $1 \le j \le n$,
  if $A(i) = B(j)$ then
        $L(i, j) = L(i-1, j-1) + 1$
        $LCS(i, j)$
              = the first $L(i-1, j-1)$ symbols of
                  $LCS(i-1, j)$ appended by $B(j)$
  else if $L(i, j-1) \ge L(i-1, j)$ then
        $L(i, j) = L(i, j-1)$
        $LCS(i, j) = LCS(i, j-1)$
  else   $\{L(i, j-1) < L(i-1, j)\}$
        $L(i, j) = L(i-1, j)$
        $LCS(i, j) = LCS(i-1, j)$
  end if
```

Further, by replacing the integer j in Property 2 with $j + k$, where $1 \le k < n$, we can obtain Property 3.

Property 3.

```
For $1 \le i \le m$, $1 \le j \le n - k$, and $1 \le k < n$,
  if $A(i) = B(j + k)$ then
        $L(i, j + k) = L(i-1, j + k - 1) + 1$
        $LCS(i, j + k)$
              = the first $L(i-1, j + k - 1)$
                  symbols of $LCS(i-1, j + k)$
                  appended by $B(j + k)$
  else if $L(i, j + k - 1) \ge L(i-1, j + k)$ then
        $L(i, j + k) = L(i, j + k - 1)$
        $LCS(i, j + k) = LCS(i, j + k - 1)$
  else   $\{L(i, j + k - 1) < L(i-1, j + k)\}$
        $L(i, j + k) = L(i-1, j + k)$
        $LCS(i, j + k) = LCS(i-1, j + k)$
  end if
```

Property 2 and Property 3 can be combined to obtain Property 4.



```
{Operations performed by PE $y$, $1 \le y \le r$.  Let $q = \lceil n/r \rceil$ and $j = (y-1)q + 1$}
 1. if YI < 0 then                          {LI[1..q] belong to string $B$}
 2.     if YI = –1 then                     {LI[1..q] = $B(j) \ldots B(j+q-1)$}
 3.         BR[1..q] := LI[1..q]            {load $B(j) \ldots B(j+q-1)$ to BR[1..q]}
 4.         LR[0..q] := 0                   {$L(0, j-1) = \ldots = L(0, j+q-1) = 0$}
 5.     else                                {LI[1..q] = $B((h-1)q+1), \ldots, B(hq)$, where $h > y$}
 6.         LO[1..q] := LI[1..q]            {forward $B((h-1)q+1), \ldots, B(hq)$}
 7.         YI := YI + 1
 8.     end if
 9. else                                    {SI belongs to string $A$, say, $A(i)$}
10.     TR := LR[1]                         {save $L(i-1, j)$}
11.     if SI = BR[1] then                  {$A(i) = B(j)$}
12.         LR[1] := LR[0] + 1              {$L(i, j) = L(i-1, j-1) + 1$}
13.         LCS[1, LR[1]] := BR[1]          {the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$
                                              in LCS[1, 1..LR[0] appended by $B(j)$ make $LCS(i, j)$}
14.     else                                {$A(i) \ne B(j)$}
15.         if YI ≥ LR[1] then              {$L(i, j-1) \ge L(i-1, j)$}
16.             LR[1] := YI                 {$L(i, j) = L(i, j-1)$}
17.             LCS[1, 1..LR[1]] := LI[1..YI]   {$LCS(i, j) = LCS(i, j-1)$}
18.         end if                          {if $L(i, j-1) < L(i-1, j)$, no operations are needed}
19.     end if
20.     for k = 1 to q – 1                  {compare $A(i)$ with $B(j+k)$, $1 \le k < q$}
21.         QR := LR[k+1]                   {save $L(i-1, j+k)$}
22.         if SI = BR[k+1] then            {$A(i) = B(j+k)$}
23.             LR[k+1] := TR + 1           {$L(i, j+k) = L(i-1, j+k-1) + 1$}
24.             LCS[k+1, TR+1] := BR[k+1]   {the first $L(i-1, j+k-1)$ symbols of
                                              $LCS(i-1, j+k)$ in LCS[k+1, 1..TR]
                                              appended by $B(j+k)$ make $LCS(i, j+k)$}
25.         else if LR[k] ≥ LR[k+1]         {$L(i, j+k-1) \ge L(i-1, j+k)$}
26.             LR[k+1] := LR[k]            {$L(i, j+k) = L(i, j+k-1)$}
27.             LCS[k+1, 1..LR[k+1]] := LCS[k, 1..LR[k+1]] {$LCS(i, j+k) = LCS(i, j+k-1)$}
28.         end if                          {if $L(i, j+k-1) < L(i-1, j+k)$,
                                              no operations are needed}
29.         TR := QR                        {save $L(i-1, j+k)$, to be $L(i-1, j+k-1)$ in
                                              the next iteration}
30.     end for
31.     SO := SI                            {send out $A(i)$}
32.     YO := LR[q]                         {send out $L(i, j+q-1)$}
33.     LO[1..YO] := LCS[q, 1..LR[q]]       {send out $LCS(i, j+q-1)$}
34.     LR[0] := YI                         {save $L(i, j-1)$}
35. end if
```

Fig. 1. Systolic algorithm for the LCS problem.

by the LI[1..q] inputs are loaded into the BR[1..q] registers (line 3), and LR[0..q] registers are set to 0 to represent $L(0, j-1) = \ldots = L(0, j+q-1) = 0$ (line 4). Otherwise, when YI < –1, the LI[1..q] inputs, which carry $B((h-1)q+1), \ldots, B(hq)$, where $h > y$, are sent out through the LO[1..q] ports (line 6), and the YI input plus 1 is sent out through the YO port (line 7) so that registers BR[1..q] and LR[0..q] in PE $y+1$ through PE r can be initialized.

If YI ≥ 0 and thus the SI input is a symbol of A, say $A(i)$, PE y computes $L(i, j), \ldots, L(i, j+q-1)$, and $LCS(i, j), \ldots, LCS(i, j+q-1)$. First, consider finding $L(i, j)$ and $LCS(i, j)$. Note that as shown in Table 1, before the inputs are processed, registers LR[0] and LR[1] hold the values $L(i-1, j-1)$ and $L(i-1, j)$, respectively. If $A(i) = B(j)$ (line 11), register LR[1] is loaded with $L(i-1, j-1) + 1$, or $L(i, j)$ (line 12). Because registers LCS[1, 1..LR[0]] contains the first $L(i-1, j-1)$ symbols of $LCS(i-1, j)$, LCS[1, 1..LR[1]] holds $LCS(i, j)$ after loading $B(j)$ to LCS[1, LR[1]] (line 13). In case $A(i) \neq B(j)$, if $L(i, j-1) \geq L(i-1, j)$ (line 15), register LR[1] is also loaded with $L(i, j)$, which is $L(i, j-1)$ from the YI input (line 16), and LCS[1, 1..LR[1]] is loaded with $LCS(i, j-1)$ from LI[1..YI] (line 17); if $L(i, j-1) < L(i-1, j)$, no operations are needed since LR[1] has already obtained $L(i-1, j) = L(i, j)$, and LCS[1, 1..LR[1]] has contained $LCS(i-1, j) = LCS(i, j)$.

Next, consider finding $L(i, j+1)$ through $L(i, j+q-1)$ and $LCS(i, j+1)$ through $LCS(i, j+q-1)$. From the comments in lines 10 and 29 we can see that at the beginning of each iteration of the for-loop between lines 20 and 30, register TR holds the value of $L(i-1, j+k-1)$, and TR will be updated to hold $L(i-1, j+k)$ at the end of each iteration as a result of executing lines 12 and 29. If $A(i) = B(j+k)$, register LR[k+1] is loaded with $L(i-1, j+k-1) + 1$, or $L(i, j+k)$ (line 23). Because registers LCS[k+1, 1..TR] contains $LCS(i-1, j+k-1)$, LCS[k+1, 1..LR[k+1]] contains $LCS(i, j+k)$ after loading $B(j+k)$ to LCS[k+1, LR[k+1]] (line 24). In case $A(i) \neq B(j+k)$, if $L(i, j+k-1) \geq L(i-1, j+k)$ (line 25), register LR[k+1] is loaded with $L(i, j+k)$, which is $L(i, j+k-1)$ from LR[k] (line 26), and LCS[k+1, 1..LR[k+1]] is loaded with $LCS(i, j+k-1)$ from LCS[k, 1..LR[k+1]] (line 27); If $L(i, j+k-1) < L(i-1, j+k)$, no operations are needed because LR[k+1] has already held $L(i-1, j+k) = L(i, j+k)$, and LCS[k+1, 1..LR[k+1]] has contained $LCS(i-1, j+k) = LCS(i, j+k)$.

Finally, some more operations are needed to output data and save a temporary value. The symbol $A(i)$ and $L(i, j+q-1)$ in register LR[q] are sent out through ports SO and YO, respectively (lines 31-32). The $LCS(i, j+q-1)$ in registers LCS[q, 1..LR[q]] are sent out through ports LO[1..YO] (lines 33). The YI input, which is $L(i, j-1)$, is saved in register LR[0] (line 34).

The correctness of the presented algorithm is made clear by considering the correspondence between Property 4 and the core operations of the algorithm. Each equation in Property 4 has a corresponding operation in lines 11-28 of Fig. 1 except for the following two cases:

```
if $A(i) \neq B(j)$ and $L(i, j-1) < L(i-1, j)$ then
        $L(i, j) = L(i-1, j)$
        $LCS(i, j) = LCS(i-1, j)$
end if
```

and

```
if $A(i) \neq B(j+k)$ and
            $L(i, j+k-1) < L(i-1, j+k)$ then
        $L(i, j+k) = L(i-1, j+k)$
        $LCS(i, j+k) = LCS(i-1, j+k)$
end if
```

As already mentioned, in these two cases, no operations are need.

Fig. 2 shows initial snapshots for finding an LCS and p on two PEs for the case of $A = acbdcbe$ and $B = abceba$. The first snapshot is taken just before the first time step begins. An LCS $abcb$ and $p = 4$ can be obtained simultaneously at ports LO[1..p] and YO of the rightmost PE at time step $m + 2r - 1 = 10$.

If string A is input before string B, that is, the roles of A and B are interchanged, a dual algorithm can be easily obtained. If each PE has $mq + 2q + 3$ registers, where $q = \lceil m/r \rceil$, as a result of reading q symbols of A at a time before B, the dual algorithm takes only $n + 2r - 1$ time steps.

We now name the algorithm presented in [16] Algorithm 1, and the one specified by Fig. 1 Algorithm 2. Like Algorithm 1, Algorithm 2 can be modified by using the broadcast operation, as follows: In the first step, symbols of string B are broadcast to every PE. Registers BR[1..q] and LR[0..q] are initialized in the same step. After that, a symbol of string A can be sent to the array at each following step, and each PE performs the operations specified by lines 10-34 of Fig. 1. Hence, the broadcast variant of Algorithm 2 takes $m + r$ steps.

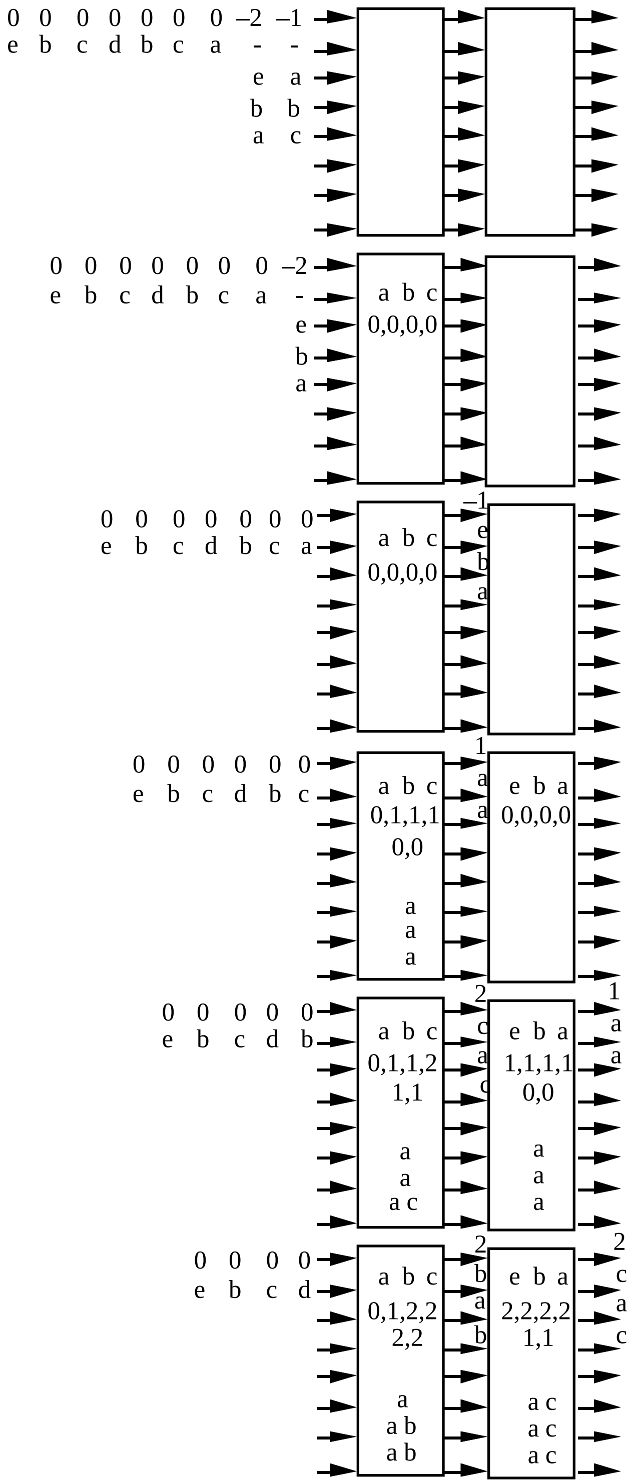


Fig. 2. Initial snapshots for finding an LCS and the LLCS for *A*= *acbdcbe* and *B* = *abceba*.

4. Experimental results

We have implemented the broadcast variants of Algorithm 1 and Algorithm 2 with Message-Passing Interface (MPI) [5] and C on the IBM SP2 [1]. In this section, we present experimental results of algorithms after briefly introducing the SP2 and MPI.

With a scalable architecture, the SP2 allows us to build a system that ranges in size from 4 to hundreds of processor nodes, which can achieve performance in TeraFLOPs.

The SP2 we used on the National Center for High-performance Computing (NCHC), Taiwan, had 110 processor nodes: 40 POWER2 (P2) nodes with clock rate 66.7 MHz, 40 POWER2 Superchip (P2SC) nodes with clock rate 120 MHz, and 30 P2SC nodes with clock rate 160 MHz. The peak performance for each node is 266 MFLOPs for P2 or 480 MFLOPs for P2SC. In addition, each processor has its own I/O ports, 128 MB or 256 MB main memory and 2 GB to 9 GB hard disk space; it can be regarded as an independent workstation.

A high-performance switch is available for low latency and high bandwidth communication. It provides the interconnection network for all the nodes to be regarded as fully connected. The switch is a multistage network; by adding switches, the system can continue to provide the same level of bandwidth to each processor node while we expand the number of processors.

MPI is a portable message-passing standard for easing the development of parallel applications. The standard defines the syntax and semantics of library routines useful in writing message-passing programs in Fortran 77 or C. In a word, MPI is used to communicate messages among a set of processors. In addition, MPI is easily compatible with distributed-memory multicomputers, shared-memory multiprocessors, networks of workstations and the combination of these elements.

In order to evaluate the performance of a program, we recorded the starting time and the ending time in each processor. Then, we took the earliest starting time and the latest ending time in all processors to compute the total execution time of the program. Although there are two kinds of processor nodes in the NCHC's SP2, we choose only P2 nodes to conduct the experiment. Because there are some factors that may affect the execution time, such as the workload and the competition of the network with other users, we took the least execution time of 10 runs as the result.

Two important performance measures for evaluating parallel algorithm are speedup and efficiency [4]. Absolute speedup is the ratio between the time needed for the fastest serial algorithm running on one processor and the time T_p needed for a parallel algorithm running on p processors. Relative speedup, S_r, is defined as T_1/T_p and relative efficiency is defined as S_r/p, where T_1 is the execution time for the parallel algorithm running on one processor. In this section, relative speedup and relative efficiency are used to evaluate algorithm performance.

Table 2 shows the comparison between Algorithm 1 and Algorithm 2. Clearly, the performance of Algorithm 2 is better than Algorithm 1. Tables 3 and 4 show the experimental results of Algorithm 2. These tables reveal that Algorithm 2 is efficient.

Table 2. Comparison between Algorithm 1 and Algorithm 2 for m = 10000.

	Algorithm 1		Algorithm 2	
	$n = 8$	$n = 16$	$n = 8$	$n = 16$
Execution time (μs)	43.7	46.2	34.8	37.6

Table 3. Experimental results of Algorithm 2 implemented on the SP2 for m = 2000 and n = 2000.

Performance measure	Number of PEs			
	1	2	4	8
Execution time (μs)	1573	855.7	475.3	243.1
Relative speedup	–	1.84	3.31	6.47
Relative efficiency	–	0.92	0.83	0.81

Table 4. Experimental results of Algorithm 2 implemented on the SP2 for m = 10000 and n = 2000.

Performance measure	Number of PEs				
	1	2	4	8	16
Execution time (μs)	8324	4701	2366	1203	605
Relative speedup	–	1.77	3.52	6.92	13.8
Relative efficiency	–	0.89	0.88	0.87	0.85

5. Conclusion

We have presented a systolic algorithm that uses a linear array of $r < n/2$ PEs to solve the LCS problem. It takes the fewest time steps of all systolic algorithms for the LCS problem. Compared with other systolic algorithms, the algorithm also requires fewer processors. Experimental results confirm that multicomputer implementation of the presented algorithm can be efficient.

Acknowledgments

This research was supported in part by the National Science Council of the R.O.C. under contract NCS87-2213-E-011-005. The authors are grateful to the National Center for High-performance Computing of the R.O.C. for sharing the SP2.

References

[1] T. Agerwala, J.L. Martin, J.H. Mirza, D.C. Sadler, D.M. Dias, and M. Snir, "SP2 system architecture," *IBM Syst. J.*, vol. 34, pp. 152-184, 1995.

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

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

[4] I. Foster, *Designing and Building Parallel Programs*. Reading, MA: Addison-Wesley, 1995.

[5] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, MA: MIT Press, 1994.

[6] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge, England: Cambridge University Press, 1997.

[7] D.S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *CACM*, vol. 18, pp. 341-343, June 1975.

[8] D.S. Hirschberg, "Algorithms for the longest common subsequence problem," *J. ACM*, vol. 24, pp. 664-675, Oct. 1977.

[9] S.K. Kumar and C.P. Rangan, "A linear-space algorithm for the LCS problem," *Acta Inform.*, vol. 24, pp. 353-362, 1987.

[10] H.T. Kung, "Why systolic architectures?," *IEEE Computer*, vol. 15, pp. 37-46, Jan. 1982.

[11] S.Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[12] T. Lecroq, G. Luce, and J.F. Myoupo, "A faster linear systolic algorithm for recovering a longest common subsequence," *Inform. Process. Lett.*, vol. 61, pp. 129-136, Feb. 14 1997.

[13] Y.-C. Lin, "New systolic arrays for the longest common subsequence problem," *Parallel Comput.*, vol. 20, pp. 1323-1334, Sep. 1994.

[14] Y.-C. Lin and J.-C. Chen, "An efficient systolic algorithm for the longest common subsequence problem," *J. Supercomput.*, vol. 12, pp. 373-385, Oct. 1998.

[15] Y.-C. Lin and J.-C. Chen, "Another efficient systolic algorithm for the longest common subsequence problem," *J. Chinese Institute Engineers*, vol. 23, pp. 607-613, Sep. 2000.

[16] Y.-C. Lin and J.-W. Yeh, "Deriving a systolic algorithm for the LCS problem," in *Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, 1998, pp. 1890-1897.

[17] M. Lu and H. Lin, "Parallel algorithms for the longest common subsequence problem," *IEEE Trans. Parallel Distributed Syst.*, vol. 5, pp. 835-848, Aug. 1994.

[18] 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, pp. 20-29.

[19] G. Luce and J.F. Myoupo, "Application-specific array processors for the longest common subsequence problem of three sequences," *Parall. Algo. Applic.*, vol. 13, pp. 27-52, 1998.

[20] G. Luce and J.F. Myoupo, "Systolic-based parallel architecture for the longest common subsequences problem," *Integration*, vol. 25, pp. 53-70, Sep. 1998.

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

[22] J.J. Navarro, J.M. Llaberia, and M. Valero, "Partitioning: an essential step in mapping algorithms into systolic array processors," *IEEE Computer*, vol. 20, pp. 78-88, July 1987.

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

[24] Y. Robert and M. Tchuente, "A systolic array for the longest common subsequence problem," *Inform. Process. Lett.*, vol. 21, pp. 191-198, Oct. 7, 1985.

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

[26] J. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology*. Boston, MA: PWS, 1997.

[27] R.A. Wagner and M.J. Fischer, "The string to string correction problem," *J. ACM*, vol. 21, pp. 168-173, Jan. 1974.

Property 4.

```
For $1 \le i \le m$, $1 \le j \le n - k$, and $1 \le k < n$,
  if $A(i) = B(j)$ then                                {Property 2}
      $L(i, j) = L(i - 1, j - 1) + 1$
      $LCS(i, j)$
          = the first $L(i - 1, j - 1)$ symbols of
              $LCS(i - 1, j)$ appended by $B(j)$
  else if $L(i, j - 1) \ge L(i - 1, j)$ then
      $L(i, j) = L(i, j - 1)$
      $LCS(i, j) = LCS(i, j - 1)$
  else  {$L(i, j - 1) < L(i - 1, j)$}
      $L(i, j) = L(i - 1, j)$
      $LCS(i, j) = LCS(i - 1, j)$
  end if
  if $A(i) = B(j + k)$ then                            {Property 3}
      $L(i, j + k) = L(i - 1, j + k - 1) + 1$
      $LCS(i, j + k)$
          = the first $L(i - 1, j + k - 1)$
              symbols of $LCS(i - 1, j + k)$
              appended by $B(j + k)$
  else if $L(i, j + k - 1) \ge L(i - 1, j + k)$ then
      $L(i, j + k) = L(i, j + k - 1)$
      $LCS(i, j + k) = LCS(i, j + k - 1)$
  else  {$L(i, j + k - 1) < L(i - 1, j + k)$}
      $L(i, j + k) = L(i - 1, j + k)$
      $LCS(i, j + k) = LCS(i - 1, j + k)$
  end if
```

3. The algorithm

Property 4 can be mapped to a systolic algorithm specified by Fig. 1. Fig. 1 shows a linear array of r PEs, to which $q = \lceil n/r \rceil \ge 2$ symbols of B can be input at a time. The algorithm computes $L(m, n) = p$ and $LCS(m, n)$ in $m + 2r - 1$ time steps. Each of the r PEs contains $nq + 2q + 3$ registers: BR[1..q], LR[0..q], TR, QR, and LCS[1, 1..n],..., LCS[q, 1..n]. Each PE except for the first one has $n + 2$ input ports, named YI, SI, LI[1..n], and $n + 2$ output ports, called YO, SO, LO[1..n]. Note that in Fig. 1 it is assumed that $q = n/r$. If n mod $r = k \ne 0$, then $q = \lceil n/r \rceil$, each of the symbols $B(n - q + k + 1)$,..., $B(n)$ in the LI input streams should become a special symbol “-”, and the symbol $B(n - q + k)$ should be replaced by $B(n)$. For ease of presentation, we assume n mod $r = 0$, unless otherwise stated.

Before explaining how the algorithm works, we first examine the input and output data. The YI input stream consists of r negative integers –1,..., $-r$ and m 0-valued flags; a negative integer $-i$ indicates that the accompanying symbols in the LI[1..q] input streams are $B((i - 1)q + 1)$,..., $B(iq)$. The jth 0-valued flag represents $L(j, 0) = 0$. As specified in Fig. 1, the value of YI input to a PE affects the operations performed by the PE. The SI input stream contains r symbols of "-" followed by symbols of A. The LLCS p can be obtained at the YO port of the rightmost PE at the $(m + 2r - 1)$th time step, and an LCS can be retrieved simultaneously from the LO[1..n] ports, or more precisely LO[1..p]. The comments in Fig. 1 describe the meanings of operations performed by PE y, $1 \le y \le r$.

It is helpful to understand the roles played by the registers and the meanings of the I/O data of PE y, $1 \le y \le r$, to understand how the whole array can produce an LCS and the LLCS. Table 1 summarizes the meanings of values in registers BR[1..q], LR[0..q], and LCS[1, 1..n] through LCS[q, 1..n]; values transferred through input ports YI and LI[1..YI] to PE y; and values transferred through output ports YO and LO[1..YO] of PE y. For ease of presentation, let $j = (y - 1)q + 1$. Note that although registers LCS[k, 1..n] are assumed to be available for storing $LCS(i, j + k - 1)$, for $1 \le k \le q$, it is quite probable that only portions of the registers are needed; specifically, since register LR[k] keeps $L(i, j + k - 1)$, only LCS[k, 1..LR[k]] are needed to hold $LCS(i, j + k - 1)$. Similarly, only ports LI[1..YI] of LI[1..n] and LO[1..YO] of LO[1..n] are needed.

Table 1. Meanings of values in registers and values transferred through I/O ports at the time.step in which $A(i)$ is sent to PE y. Note: $j = (y - 1)q + 1$ and $1 \le k \le q$.

Register or I/O port	Before inputs are processed	At the end of the time step
BR[k]	$B(j + k - 1)$	$B(j + k - 1)$
LR[0]	$L(i - 1, j - 1)$	$L(i, j - 1)$
LR[k]	$L(i - 1, j + k - 1)$	$L(i, j + k - 1)$
LCS[k, 1..n]	$LCS(i - 1, j + k - 1)$	$LCS(i, j + k - 1)$
YI	$L(i, j - 1)$	–
LI[1..YI]	$LCS(i, j - 1)$	–
YO	–	$L(i, j + q - 1)$
LO[1..YO]	–	$LCS(i, j + q - 1)$

The operations of PE y are explained in the following. If the YI input is negative (line 1), registers BR[1..q] and LR[0..q] are initialized. Specifically, if YI = –1 (line 2), symbols $B(j)$,..., $B(j + q - 1)$ carried