ELSEVIER

Information Processing Letters 61 (1997) 129–136

Information Processing Letters

A faster linear systolic algorithm for recovering a longest common subsequence [1]

Thierry Lecroq [a,2], Guillaume Luce [b,3], Jean Frédéric Myoupo [b,*]

[a] *LIR: Laboratoire d'Informatique de Rouen, Faculté des Sciences et des Techniques, 76821 Mont-Saint-Aignan Cedex, France*

[b] *LaRIA: Laboratoire de Recherche en Informatique d'Amiens, Université de Picardie Jules Verne, CURI, 9 rue du Moulin Neuf, 80000 Amiens, France*

Received 1 February 1996; revised 1 December 1996
Communicated by S.G. Akl

Abstract

We present a new linear systolic array architecture of m cells which outputs a longest common subsequence (LCS) of two input strings A and B in time $n + 2m$, where n and m denote the lengths of A and B respectively ($m \leqslant n$). Our approach improves the time of execution required by previous linear systolic arrays for this purpose. Furthermore, a design combining a tree with the linear array provides an LCS and its length in $n + m + \log m$ clock cycles only. © 1997 Elsevier Science B.V.

Keywords: Parallel algorithms; Parallel architectures; Linear systolic arrays; VLSI; Longest common subsequence problem

1. Introduction

In this paper we describe systolic array systems for the longest common subsequence (LCS) problem. Given two input strings, the LCS problem consists in finding a subsequence of both strings which is of maximal possible length, say p. Computing p only solves the problem of determining the length of an LCS (LLCS problem). Several linear systolic arrays have ever been designed for both LLCS and LCS problems but our approach significantly improves the time of execution for both problems.

Given a string A over an alphabet Σ, a *subsequence* of A is any string C that can be obtained from A by deleting zero or some symbols (not necessarily consecutive). The *longest common subsequence* (LCS) problem for the two input strings

$$A = A[1] \ldots A[n] \quad \text{and} \quad B = B[1] \ldots B[m]$$

($m \leqslant n$, without restriction of generality) consists in finding another string $C = C[1] \ldots C[p]$ such that C is a subsequence of both A and B, and is of maximal possible length.

The LLCS and LCS problems frequently arises in a number of areas such as genetic engineering, data compression and syntactic pattern recognition. The asymptotically fastest general solution [9] needs time $O(n^2/\log n)$ and uses the "Four Russians" trick. A lot of algorithms have been developed that, although

* Corresponding author. Email: myoupo@laria.u-picardie.fr.

[1] This work was supported by the Pole Modelisation de la region Picardie.

[2] Email: lecroq@dir.univ-rouen.fr.

[3] Email: luce@laria.u-picardie.fr.

PII S0020-0190(97)00011-2

not improving the general $O(nm)$ time bound of the dynamic programming approach, exhibit a much better performance by specializing on certain classes of pairs of sequences. These algorithms are sensitive to other problem parameters such as the length p of an LCS, the number of matches or the alphabet size (see [3] for a survey). In recent years, exploiting the parallelism of this problem attracts many research interests (among others [1,7] on PRAM models, [2] using the bus-automaton model).

With the recent advances in the very large scale integration (VLSI) techniques, which offer us opportunities to develop parallel computation for both special-purpose and general-purpose devices, systolic array architectures have been successfully designed for efficiently implementing algorithms related to various areas such as digital signal processing, matrix operation, and dynamic programming. A systolic array is a network of processors which rhythmically compute and pass data through the system [5]. The data move through the processors in a highly regular fashion, and simple operations are performed on them. The processors receive their input from their neighbors, operate on it, and pass their result on. One can think of systolic arrays as synchronous parallel devices, and moreover, the control of the operation can be made only by local decisions. We focus here on linear arrays as they are attractive for a simple global clock whose rate is independent of the size of the array.

Linear systolic arrays for the LLCS and LCS problems are proposed in [6,8,10]. The arrays in [10] and [8] compute the length p of an LCS in $n+2m$ clock cycles and the ones in [6] compute it in $n+2m-1$ clock cycles, where m and n are the lengths of the two strings. But, to recover an LCS, there is no array which achieves this time bound: the algorithms proposed in [6,8] run in time $n+3m+p-1$. In this paper we design a linear systolic array which requires $n+2m$ clock cycles only to recover one LCS. By combining it with a tree which uses a data broadcasting scheme, it can output both the length of an LCS and an LCS itself after $n+m+\log m$ clock cycles. This is the fastest systolic array architecture we know for both LLCS and LCS problems.

The rest of this paper is organised as follows: Section 2 sets the problems. In Section 3, we present the new linear systolic array architecture and algorithm. Section 4 ends the paper.

2. Statement of the problem

Definition 1. Let Σ be any finite set of symbols called an alphabet. A *string* over Σ is any finite sequence of elements from Σ. Σ^* is the set of all strings over Σ, including the empty one ε. Σ^+ denotes $\Sigma^* - \{\varepsilon\}$. For any string A, $|A|$ denotes the length of A (the number of symbols in A). Note $|\varepsilon| = 0$.

Definition 2. A string $A \in \Sigma^+$ is fully specified by writing $A = A[1]\ldots A[n]$, where $A[i] \in \Sigma$ $(1 \leqslant i \leqslant n)$. String $A' = A[i_1]\ldots A[i_k]$ $(1 \leqslant k \leqslant n)$, where $1 \leqslant i_1 < \cdots < i_k \leqslant n$, is called a *subsequence* of A. ε is also a subsequence of A.

Definition 3. Let A and B be two strings over Σ. String C is a *common subsequence* of A and B if it is a subsequence of both A and B. C is a *longest common subsequence* (LCS) of A and B if C is a common subsequence of A and B of maximal possible length.

Throughout this paper, let $LCS(i,j)$ be an LCS of the two strings $A[1:i] = A[1]\ldots A[i]$ and $B[1:j] = B[1]\ldots B[j]$ $(1 \leqslant i \leqslant n,\ 1 \leqslant j \leqslant m$, where $m \leqslant n)$ and $LLCS(i,j) = |LCS(i,j)|$ its length. The *LLCS problem* is to compute the length $p = LLCS(n,m)$ of an LCS of A and B. The *LCS problem* is to find such a string $C = C[1]\ldots C[p] = LCS(n,m)$.

Example 4. The two strings $A = cbacbaaba$ and $B = abcdbb$ (of length $n = 9$ and $m = 6$) admit the two LCSs $acbb$ and $bcbb$ (of length $p = 4$).

A simple dynamic programming scheme which computes p is given below (see [4] for its proof):

Proposition 5. *For* $1 \leqslant i \leqslant n$ *and* $1 \leqslant j \leqslant m$:

$$LLCS(i,0) = LLCS(0,j) = LLCS(0,0) = 0$$
$$LLCS(i,j) = \textit{if } (A[i] = B[j])$$
$$\quad \textit{then } 1 + LLCS(i-1,j-1)$$
$$\quad \textit{else } \max(LLCS(i-1,j), LLCS(i,j-1))$$

The above proposition trivially yields the following one:

Proposition 6. *For* $1 \leqslant i \leqslant n$ *and* $1 \leqslant j \leqslant m$:

$LCS(i,0) = LCS(0,j) = LCS(0,0) = \varepsilon$
$LCS(i,j) = if\ (A[i] = B[j])$
  $then\ LCS(i-1,j-1)A[i]$
  $else\ if\ (LLCS(i-1,j) \geqslant LLCS(i,j-1))$
    $then\ LCS(i-1,j)$
    $else\ LCS(i,j-1)$

Proposition 6, combined with Proposition 5, provides a systolic criterion to construct an LCS in only one pass. We now design the new linear array architecture which enables to directly implement these propositions.

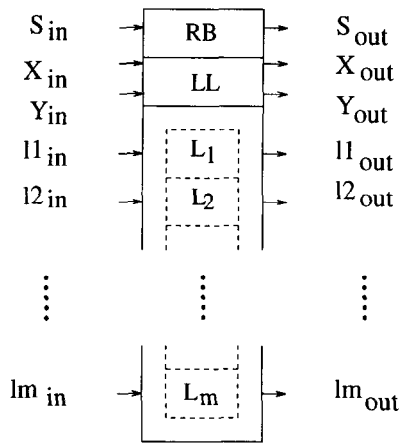

Fig. 1. Processor j ($1 \leqslant j \leqslant m$).

3. A faster linear systolic array architecture

Robert and Tchuente in [10] first investigate linear systolic arrays for the LCS problem. Lin [6] implements Proposition 5 on a one-way linear systolic array of m processors to compute the length p of an LCS in $n + 2m$ clock cycles. In order to recover an LCS, a systolic stack is set on each processor to store matches (i.e. $A[i] = B[j]$) which occurred while computing p. Then they operate a second phase in reverse order to trace back a solution which is output from last processor after $3m$ more clock cycles. Two phases are also required for the algorithms described in [6] and [8] when recovering an LCS is desired (time bound being improved down to $n + 3m + p - 1$ time units). Our approach for recovering an LCS is reminiscent to the former work in [10] but, making use of Proposition 6, we modify the design of each processor by adding an m-registers zone to it, say L, equipped with m I/O channels to carry an LCS along the array. These registers serve as a substitute for the systolic stacks or the CAMs (content-addressable memories) used in [10] and [6] respectively, and are similar to the ones presented in [8]. Introducing m I/O channels makes possible to carry along p its associated LCS. This additional hardware provides an efficient implementation with proposition 6 and suppresses the need of a second phase. So, recovering an LCS is achieved after $n + 2m$ clock cycles.

Hereafter we define the new one-way linear systolic array architecture and present its associate algorithm. Each processor j ($1 \leqslant j \leqslant m$) is equipped by two registers B and LL which respectively store $B[j]$, and $LLCS(i,j)$ after the processing of $A[i]$. Processors are one-way connected by three I/O channels, say S, X and Y, as in [10]. Moreover we equip processor j with m registers, say L_k ($1 \leqslant k \leqslant m$), which constitute a logical register, say $L = L_1 \ldots L_m$, that store $LCS(i,j)$ after the processing of $A[i]$. We also introduce m more I/O channels, say lk ($1 \leqslant k \leqslant m$) in order to transport the content of L from a processor j to processor $j+1$ in a way similar to LL does in [10]. Let l_{in} denote $l1_{\text{in}} \ldots lm_{\text{in}}$ and l_{out} denote $l1_{\text{out}} \ldots lm_{\text{out}}$, where lk_{in} and lk_{out} respectively denote the input and output of lk channel ($1 \leqslant k \leqslant m$).

Fig. 1 shows the design of a processor and Fig. 2 presents the new linear systolic array architecture and its required input data stream. The program of a processor is depicted in Fig. 3 and some pulsations for the strings of Example 4 are shown in Figs. 4–6.

First, a special value, say \$ (\$ $\notin \Sigma$), is fed onto S channel. It is accompanied on l_{in} channels by string $-\ldots-$ which stands for ε. Travelling right through the array, at time j, processor j receives \$ from its input channel S, denoted S_{in}. So, it initalizes L register to ε ($L \leftarrow l_{\text{in}}$, as $LCS(0,j) = \varepsilon$) and it saves \$ in B. It then outputs its input. Then the input items $B[j]$ (j from 1 to m) are inserted through the channel S. The symbol \$ in X tells a cell that the S_{in} symbol is from string B and \$ in Y tells a cell that this symbol has to be stored. Accurately, processor j is the first processor whose register B contains \$ which receives $(B[j], \$, \$)$ as (S,X,Y)-input. It then set LL to 0 (as $LLCS(0,j) = 0$) and B register is assigned $B[j]$,

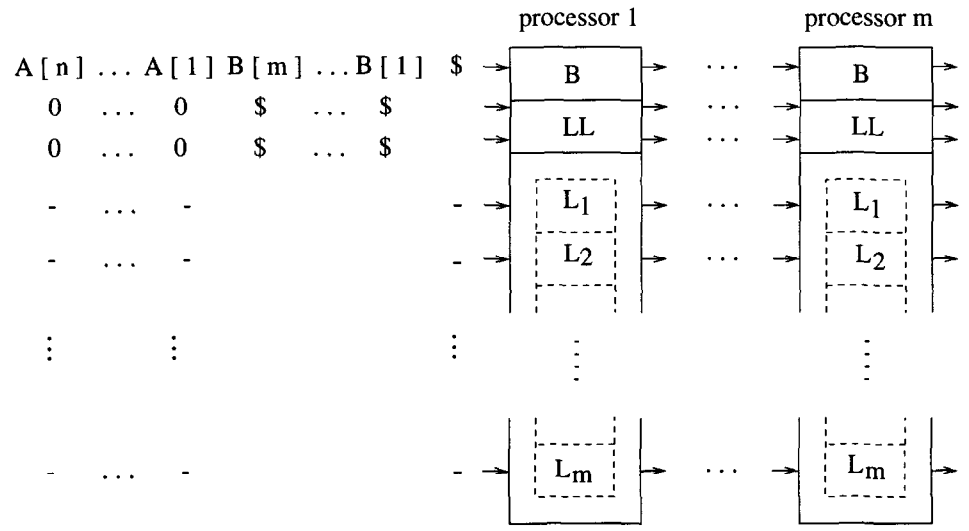

Fig. 2. The linear array and its input data stream.

```
if (S_in = $) then
  B ← $
  L ← l_in
  l_out ← L
else
  if (X_in = $) then
    X_out ← $
    if (Y_in = $) and(B = $) then
      B ← S_in
      LL ← 0
      Y_out ← 0
    else
      Y_out ← Y_in
    endif
  else
    X_out ← LL
    if (Y_in > LL) then
      LL ← Y_in
      L ← l_in
    else
      if (S_in = B) and (LL < 1 + X_in) then
        LL ← 1 + X_in
        L ← l_in.S_in
      endif
    endif
    Y_out ← LL
    l_out ← L
  endif
endif
S_out ← S_in
```

Fig. 3. The program of a processor.

clearly at time $2j$. Processor j then outputs 0 from Y. Thus the accompanying $B[j]$ will not further be considered. This principle is the one devised in [10].

The input items $A[i]$ (i from 1 to n) are sequentially inserted. As $A[i]$ travels from the left to the right, according to both Propositions 5 and 6, processor j processes $A[i]$ at time $m+i+j$ and computes both $LLCS(i,j)$ and $LCS(i,j)$ as a function of $A[i]$, $B[j]$, $LLCS(i-1,j-1)$, $LLCS(i,j-1)$, $LLCS(i-1,j)$, $LCS(i,j-1)$, $LCS(i-1,j)$ and $LCS(i-1,j-1)$ (indeed $LCS(i,j-1)$), fulfilling the following:

Before processor j processes $A[i]$:
$B = B[j]$
$LL = LLCS(i-1,j)$
$L = L_1 L_2 \ldots L_m = LCS(i-1,j)$
$S_{in} = A[i]$
$X_{in} = LLCS(i-1,j-1)$
$Y_{in} = LLCS(i,j-1)$
$l_{in} = l1_{in} l2_{in} \ldots lm_{in} = LCS(i,j-1)$

After processor j had processed $A[i]$:
$B = B[j]$
$LL = LLCS(i,j)$
$L = L_1 L_2 \ldots L_m = LCS(i,j)$
$S_{out} = A[i]$
$X_{out} = LLCS(i-1,j)$
$Y_{out} = LLCS(i,j)$
$l_{out} = l1_{out} l2_{out} \ldots lm_{out} = LCS(i,j)$

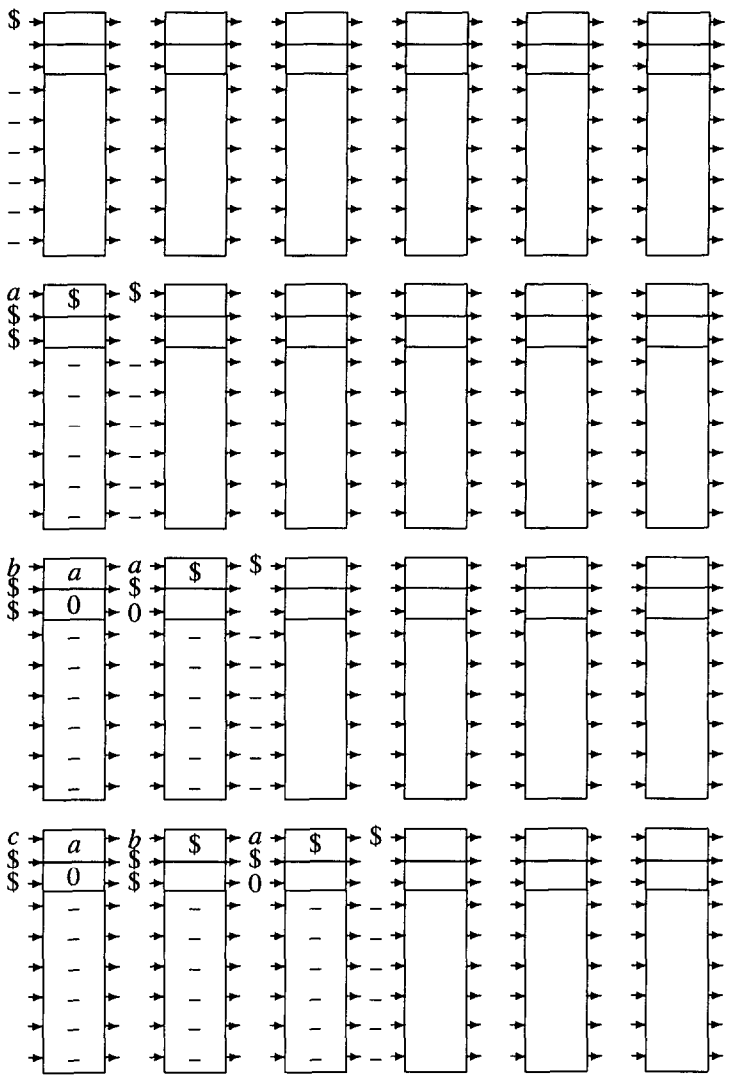

Fig. 4. Some initialization pulses for $A = cbacbaaba$ and $B = abcdbb$.

Accurately, at time $m + i + j$, $A[i]$ enters processor j (either $A[i-1]$ has been processed one cycle before or $LL = 0$ and $L = \varepsilon$). LL, which stands for $LLCS(i-1, j)$, is output on X channel, say X_{out}. Then, if $Y_{\text{in}} > LL$ (i.e. $LLCS(i, j-1) > LLCS(i-1, j)$), LL is assigned $LLCS(i, j-1)$ and L is assigned l_{in} (i.e. $LCS(i, j) = LCS(i, j-1)$) via the lk input channels linked with the L_k registers ($1 \leqslant k \leqslant m$). Else if a match occurs with $LL < 1 + X_{\text{in}}$ (i.e. $LLCS(i-1, j) < 1 + LLCS(i-1, j-1)$) then LL is assigned $LL + 1$ and L is first assigned l_{in}, also L_{LL} is assigned S_{in} (i.e. we obtain $LCS(i, j) = LCS(i-1, j-1)A[i]$). At the end of this program, LL is sent onto Y_{out} and lk_{out} channels output L to the next processor. As $A[i]$ is initially input with value 0 onto X and Y channels ($LLCS(i-1, 0) = LLCS(i, 0) = 0$), and with input value ε onto lk channels ($LCS(i-1, 0) = LCS(i, 0) = \varepsilon$), it is output from processor j with $Y_{\text{out}} = LLCS(i, j)$ and $l_{\text{out}} = L = LCS(i, j)$, according to both Propositions 5 and 6. Note that m registers and m I/O channels are obviously sufficient to carry an LCS of maximal possible length m.

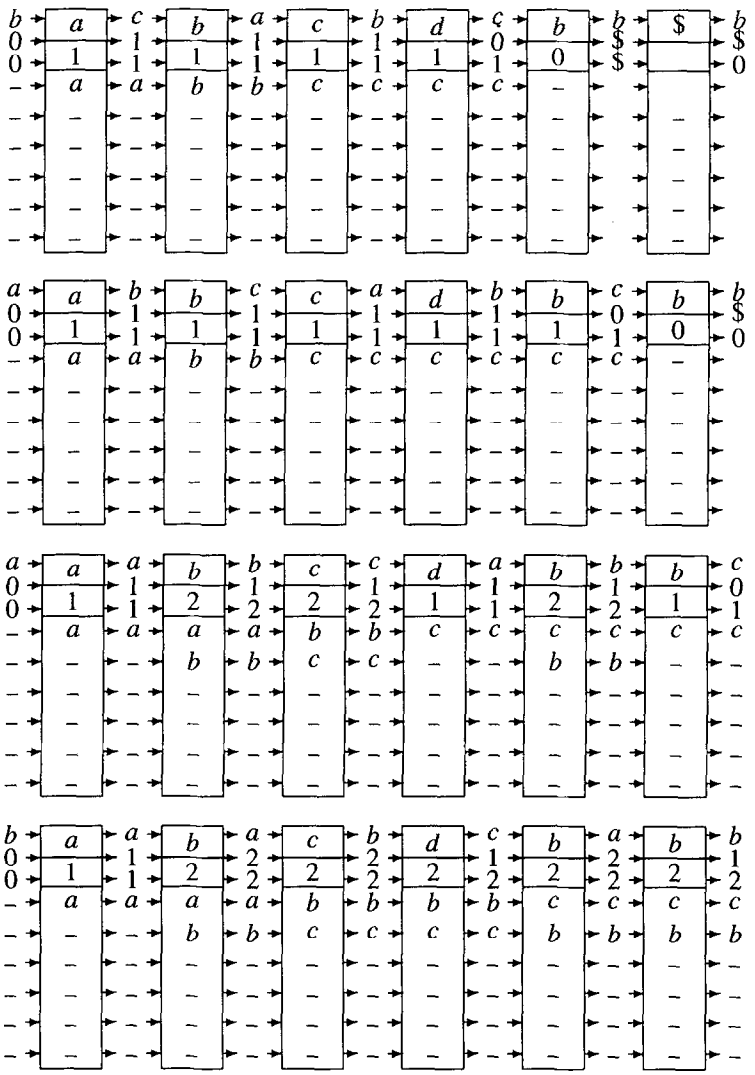

Fig. 5. Some consecutive pulses for $A = cbacbaaba$ and $B = abcdbb$.

As $A[n]$ is being processed by processor m at time $n + 2m$, $LLCS(n,m)$ and $LCS(n,m)$ are respectively output from Y_{out} and l_{out} (indeed $l1_{\text{out}} \ldots lp_{\text{out}}$) channels of processor m, solving both the LLCS and LCS problems on a one-way linear systolic array of m processors in time $n + 2m$.

A design combining $m - 1$ processors configured as a binary-tree synchronized with the above m-cells linear systolic array reduces the time required to initialize the array to logarithmic delay at the expense of simple additional hardware. A similar approach, applied to dictionary coding, is described in [11]. The tree broadcasts the shorter input string B to the linear array $(B[1], \ldots, B[m]$ successively feed the root of the tree, each symbol being simply propagated toward its two sons on each clock cycle), so $LLCS(i,j)$ (and $LCS(i,j)$ if desired) is computed at time $\log m + i + j$ instead of time $m + i + j$. Whence a total time of $n + m + \log m$ is required which is faster than the designs proposed in [6,8,10] for both problems.

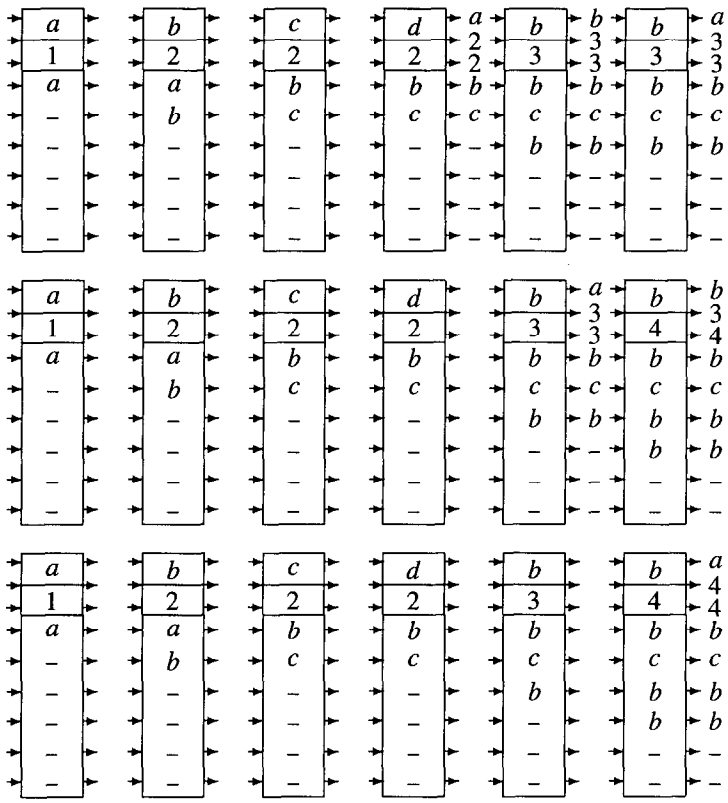

Fig. 6. Final snapshots for $A = cbacbaaba$ and $B = abcdbb$.

4. Conclusion

In this paper, we have presented a new linear systolic array architecture capable of computing both an LCS of two strings and its length in time $n + 2m$. It is a variant of the ones in [6,8]. However, one handicap of our algorithm is that it is non modularly extensible in the sens that not only the time of execution and the number of processors used depend on the size of the problem: m channels are necessary to carry an LCS which is stored in some processors when computing p. Nevertheless, our design yields a faster algorithm and needs only one phase instead of two as in [6,8].

Acknowledgements

We thank the anonymous referees whose comments and suggestions have significantly improved the presentation of this paper.

References

[1] A. Apostolico, M. Attalah, L. Larmore and S. Mcfaddin, Efficient parallel algorithms for string editing and related problems, *SIAM J. Comput.* 19 (1990) 968–988.

[2] D.M. Champion and J. Rothstein, Immediate parallel solution of the longest common subsequence problem, in: *Proc. ICPP* (1987) 70–77.

[3] V. Dančík and M.S. Paterson, Longest common subsequences, in: *Proc. Ann. Symp. Theor. Aspects Comput. Sci.* (1994) 127–142.

[4] D.S. Hirschberg, A linear space algorithm for computing maximal common subsequences, *Comm. ACM* 18 (6) (1975) 341–343.

[5] H.T. Kung, Why systolic architectures?, *IEEE Comput.* 15 (1) (1980) 37–46.

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

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

[8] G. Luce and J.F. Myoupo, An efficient linear systolic array for recovering longest common subsequences, in: *Proc. IEEE Internat. Conf. Algo. Archi. Parall. Process.*, Brisbane, Australia (1995) 20–29.

[9] W.J. Masek and M.S. Paterson, A faster algorithm for computing string edit distances, *J. Comput. System Sci.* 20 (1980) 18–31.

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

[11] R.J. Zito-Wolf, A broadcast/reduce architecture for high-speed data compression, in: *Proc. IEEE Symp. Parall. Distrib. Process.*, 1990.