VLSI DESIGN
1999, Vol. 9, No. 1, pp. 55–67
Reprints available directly from the publisher
Photocopying permitted by license only

Partitionable Bus-based String-matching Algorithm for Run-length Coded Strings with VLDCs

HSIU-NIANG CHEN[a] and KUO-LIANG CHUNG[b,*]

[a] *Department of Information Management, Van-Nung Institute of Technology, 63-1, Shuiwei, Chungli, Taoyuan, Taiwan 320, ROC.;* [b] *Department of Information Management, National Taiwan University of Science and Technology, 43, Section 4, Keelung Road, Taipei, Taiwan 10672, R.O.C.*

(Received 5 May 1997)

String matching (SM) problem is to find the occurrences of a pattern within a text. A vanable length don't care (VLDC) is a special symbol, not belonging to a finite alphabet Σ but in Σ^*. Each VLDC in the pattern can match any substring in the text. Given a run-length coded text of length $2n$ over Σ and a run-length coded pattern of length $2m$ over Σ^*, this paper first presents an $O(1)$ time parallel SM algorithm for run-length coded strings with VLDCs on a reconfigurable mesh (RM) using $O(nm)$ processors. Consider the hardware limitation in VLSI implementation. In order to be suitable for VLSI modular implementation, a partitionable parallel algorithm on the RM with limited processors is further presented. For $N < n$ and $M < m$, the SM for run-length coded strings with VLDCs can be solved in $O(\hat{X}\hat{Y})$ time on the RM using $O(NM)(= O((nm)/(\hat{X}\hat{Y})))$ processors, where $\hat{X} = \lceil (n-1)/(N-1) \rceil$ and $\hat{Y} = \lceil (m-1)/(M-1) \rceil$.

Keywords: String-matching, run-length coding with VLDCs, reconfigurable mesh, parallel algorithms, partitionable algorithm, VLSI design

1. INTRODUCTION

A basic search operation on patterns is the string matching (SM). In many applications, using a special encoding method for representing strings is important and advantageous for saving storage and manipulating them. One well-known method that has been widely used in many fields and has played a valuable historical role in the development of data compression is run-length coding. The basic idea of this method is to replace sequences of identical consecutive symbols with that representative symbol and its multiplicity. For example, the run-length coded representation of the string *aaaabbbaaacccc* is $a^4b^3a^3c^4 = (a,4)\ (b,3)\ (a,3)\ (c,4)$, and the length is reduced from 14 to 8. A variable length don't care (VLDC) is a special symbol, not belonging to Σ but in Σ^* Each VLDC in the pattern can match any substring in the text (possibly zero length). For example, given a text

*Corresponding author. e-mail: klchung@cs.ntit.edu.tw. This research was supported in part by the National Science Council of R.O.C. under contracts NSC85-2121-E011-009, NSC85-2213-M011-002, NSC86-2213-E011-010 and NCHC86-08-015.

string '*cccaaaaabbaaabbbccccddaabb*' and a pattern string '*aabb*∗*cccddaa*', where ∗ is the VLDC, the two matched positions are from 7 to 24 and from 12 to 24. The SM problem for run-length coded strings with VLDCs can be viewed as an extension of the classical SM problem and has many important applications [4] such as editing operations, pattern recognition, file retrieval, DNA matching, *etc.*

Reconfigurable mesh (RM) is a very promising platform to be used in high performance bus-based VLSI architectures due to its simplicity and regularity. Many efficient algorithms and simulations on RMs [5–13] have been developed. Except the results in [8, 9], the number of processors used in most of the published results in the literature is dependent on the problem size. Assume the text (pattern) with length $t(p)$ has been compressed into a run-length coded representation with length $2n(2m)$, $2n < t$ and $2m < p$. Previously, Chen [1] presented an $O(1)$ parallel SM algorithm on a RM with $O(t^2)$ processors. Later, Chung [2] presented an $O(1)$ time parallel SM algorithm on a RM with $O(tp)$ processors. Recently, Chung [3] presented an $O(1)$ time SM algorithm for strings with VLDCs on a RM with $O(mn)$ processors. The number of processors used in the above three parallel SM algorithms on RMs is also dependent on the problem size, and they are not suitable for VLSI modular implementation. The main motivation of this research is to design a partitionable parallel SM algorithm for run-length coded strings with VLDCs such that it is suitable for VLSI modular implementation. To the best of our knowledge, this is the first time such a partitionable parallel SM algorithm on RMs is being proposed in the literature.

This paper first presents an $O(1)$ time parallel SM algorithm for run-length coded strings with VLDCs on the RM using $O(nm)$ processors. Then, a partitionable parallel algorithm on the RM with limited processors is further presented for solving the same problem. For $N < n$ and $M < m$, the SM for run-length coded strings with VLDCs can be solved in $O(\hat{X}\hat{Y})$ time on the RM using $O(NM)$ $(= O((nm)/(\hat{X}\hat{Y})))$ processors, where $\hat{X} = \lceil (n-1)/(N-1) \rceil$ and $\hat{Y} = \lceil (m-1)/(M-1) \rceil$.

The remainder of the paper is arranged as follows. Section 2 introduces our computational model RM. In Section 3, we present the $O(1)$ time parallel SM algorithm for run-length coded strings with VLDCs on the RM with $O(mn)$ processors. In Section 4, the partitionable parallel SM algorithm for solving the same problem is presented. Some concluding remarks are included in Section 5.

2. THE COMPUTATIONAL MODEL: RM

The parallel computational model used is the 2-D RM A reconfigurable bus system is a bus system whose configuration can be dynamically changed. An $m \times n$ RM consists of $m \times n$ (m rows and n columns) identical processors arranged in a 2-D rectangular array with a reconfigurable bus system. The processor located in row i and column j for $1 \le i \le m$ and $1 \le j \le n$ is referred to as PE(i, j). Every processor has four ports denoted by N(north), S(south), E(east), and W(west), respectively. For example, Figure 1 shows a RM of size 3×5, where the RM contains 15 processors and the four black dots on each processor represent four ports.

In each processor, ports can be dynamically connected in pairs to fit computational needs. As shown in Figure 2, each processor connects its W and E ports. Of course, all processors can also connect their own W and S ports; N and S ports; N and E ports, *etc.*

We assume that the RM is operated in a SIMD (single instruction multiple data) model. All

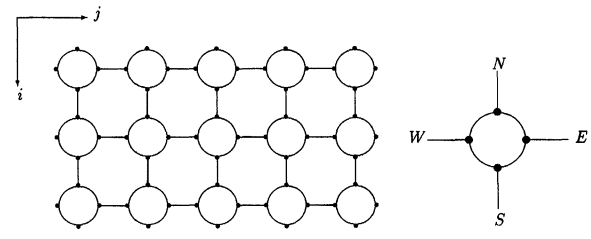


FIGURE 1 A 3×5 RM.

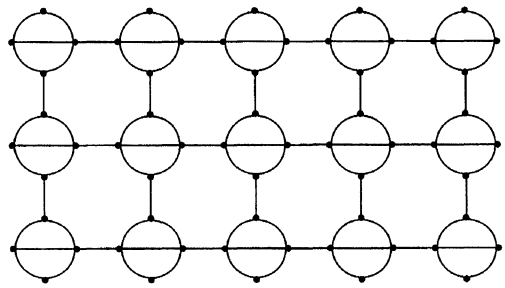


FIGURE 2 The horizontal configuration of the RM.

processors can work synchronously and setting internal connections, performing an arithmetic or boolean operation, broadcasting a value on a bus, or receiving a value from a specific bus only need $O(1)$ time. If there exists a bus between two ports, we assume that it takes $O(1)$ time to broadcast the data from one port to the other. In addition, at any given time only one value can be sent by a bus and processors read the bus if instructed to do so.

3. THE $O(1)$ TIME PARALLEL ALGORITHM

We take the text string '*cccaaaaabbaaabbbccccddaabbccccddaaa*' and pattern string *aabb*∗*cccddaa*' with one VLDC to demonstrate our $O(1)$ time parallel algorithm. The run-length coded text becomes

$$\begin{aligned} T &= c^3a^5b^2a^3b^3c^4d^2a^2b^2c^4d^2a^3 \\ &= (c,3)(a,5)(b,2)(a,3)(b,3)(c,4)(d,2) \\ &\quad (a,2)(b,2)(c,4)(d,2)(a,3) \\ &= (T(1,1),T(2,1))(T(1,2),T(2,2))\cdots \\ &\quad (T(1,12),T(2,12)); \end{aligned}$$

it has twelve entries. The run-length coded pattern becomes

$$\begin{aligned} P &= (a^2b^2 *^1 c^3d^2a^2 \\ &= (a,2)(b,2)(*,1)(c,3)(d,2)(a,2) \\ &= (P(1,1),P(2,1))(P(1,2),P(2,2))\cdots \\ &\quad (P(1,6),P(2,6)); \end{aligned}$$

it has six entries. In this example, the first matched substring in T starts from the location $(2,4)$ (the fourth position in the second entry) and ends at $(8,2)$; the second matched substring in T starts from the location $(4,2)$ and ends at $(8,2)$; the third matched substring starts from the location $(8,1)$ and ends at $(12,2)$.

On the $m \times n$ RM, our $O(1)$ time parallel SM algorithm for run-length coded strings with VLDCs is presented in the following nine steps. Initially, suppose the run-length coded text T has been fed into row 1 of the RM and the run-length coded pattern P has been fed into column 1. That is, processor PE$(1,j)$ stores the data $(T(1,j), T(2,j))$ for $1 \le j \le n$ and processor PE$(i,1)$ stores the data $(P(1,i), P(2,i))$ for $1 \le i \le m$. For simplicity, the initial data allocation on the $m \times n$ RM is illustrated in Figure 3. Here, $n = 2$ and $m = 6$.

Algorithm_1

Step 1 Establish a vertical bus system for each column. This configuration can be built by connecting the N and S ports of each processor. Then, each processor PE$(1,j)$ for $1 \le j \le n$ broadcasts its own data $T(1,j)$ and $T(2,j)$ to the others in the same column *via* the vertical bus system.

Step 2 Establish a horizontal bus system for each row. This configuration can be built by connecting the W and E ports of each processor. Then, each processor PE$(i,1)$ for $1 \le i \le m$ broadcasts its own data

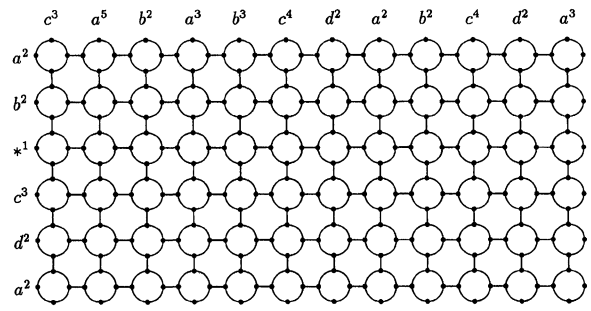


FIGURE 3 The initial configuration of the 6×12 RM.

$P(1, i)$ and $P(2, i)$ to the others in the same row via the horizontal bus system.

Step 3 Each processor first disconnects its vertical and horizontal connections. Then, each processor holding the symbol '$*$' sends a special symbol, say '+', to its south neighbor and north neighbor. Figure 4 only illustrates the special symbols, '$*$' and '+', in each processor.

Step 4 Excepting the processors in the first row, each processor connects its N and E ports.

Step 5 Each processor in the first row, the last row, and the rows holding '+' connects its W and S ports when $P(1, i) = T(1, j)$ and $P(2, i) \leq T(2, j)$. Otherwise do nothing.

Step 6 Each processor PE(i, j), $2 \leq i \leq m-1$, connects its W and S ports when $P(1, i) = T(1, j)$ and $P(2, i) = T(2, j)$. Otherwise do nothing.

Step 7 Each processor PE(i, j) holding '$*$' connects its W and S ports; connects its W and E ports. Figure 5 illustrates the configuration of the RM after performing this step.

Step 8 Each processor PE(m, j), $1 \leq j \leq n$, with the connection linking the W and S ports, first sends the symbol, say '!', from the S port to processor PE$(1, k)$ for $1 \leq k \leq n$. If each processor holding the symbol '$*$' receives the symbol '!' from its S port then it disconnects its W and E ports; disconnects its N and E ports if it has; connects its N and S ports. This can avoid the negative length effect. Figure 6 illustrates the configuration of the RM after performing this step.

Step 9 Along the corresponding stairlike bus system, each processor PE(m, j) $1 \leq j \leq n$, with the connection linking the W and S ports sends the data $(j, P(2, m))$ from the S port to processor PE$(1, k)$ for $1 \leq k \leq n$. Finally, each processor PE$(1, k)$ reports the data $(k, T(2, k) - P(2, 1) + 1)$ as the matched starting location and $(j, P(2, m))$ as the matched ending location if the W port of that processor receives the data. Note that if $P(1, m) =$ '$*$', then only PE(m, n) sends the data $(n, T(2, n))$ to PE$(1, k)$. If $P(1, 1) =$ '$*$', each processor PE$(1, k)$ reports the data $(1, 1)$ as the matched starting location and the received data as the matched ending location if the S port of that processor receives the data. Figure 7 illustrates the configuration of the RM after performing this step.

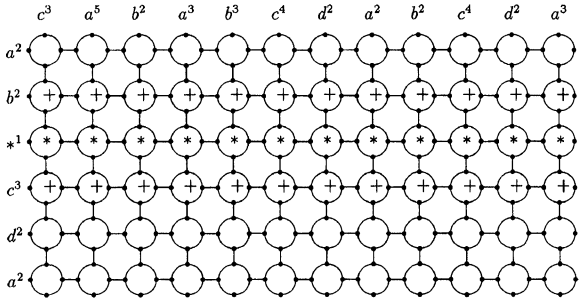


FIGURE 4 After Step 3.

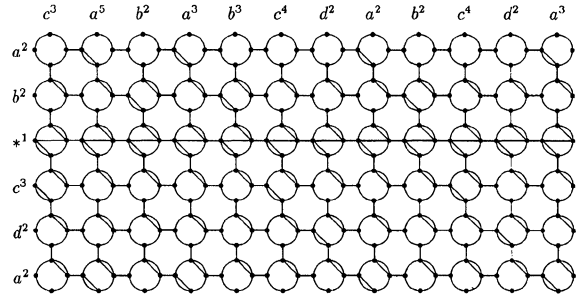


FIGURE 5 After Step 7.

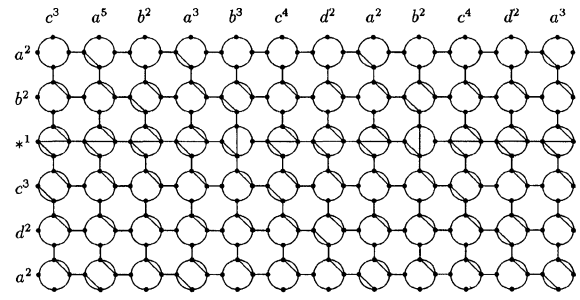


FIGURE 6 After Step 8.

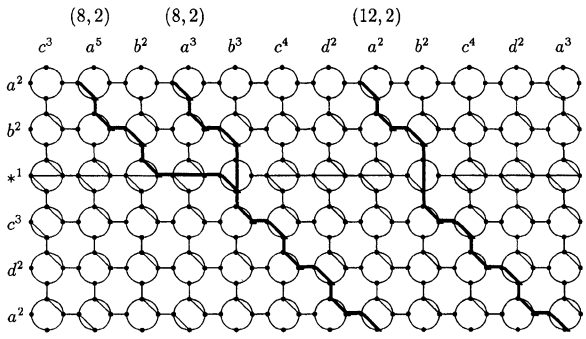


FIGURE 7 After Step 9.

In our example, the processor PE(1, 2) reports the data (2, 4) as the matched starting location and (8, 2) (see Fig. 7) as the matched ending location; PE(1, 4) reports (4, 2) as the matched starting location and (8, 2) (see Fig. 7) as the matched ending location; PE(1, 8) reports (8, 1) as the matched starting location and (12, 2) (see Fig. 7) as the matched ending location. Since each step in Algorithm_1 takes $O(1)$ time, we have the following result.

THEOREM 1 *The SM problem for run-length coded strings with VLDCs can be solved in* $O(1)$ *time on the* $m \times n$ *RM.*

4. THE PARTITIONABLE PARALLEL ALGORITHM

Consider the case when the number of processors available is not enough. Given an RM consisting of $M \times N$ (M rows and N columns) processors, if the run-length coded text (pattern) is of length $2n$ ($2m$), where $N < n$ or $M < m$, this section proposes a partitionable strategy to overcome this hardware limitation. Without loss of generality, we focus on the following two cases: (1)$N < n$; (2)$N < n$ and $M < m$.

A. Case 1: $N < n$

Assume the RM consists of 6×7 processors and the run-length coded strings are the same as in Section 3. That is, $M = 6$, $N = 7$, $m = 6$, and $n = 12$. Initially, suppose processor PE$(1, j)$, $2 \le j \le N-1$, stores the data $(T(1, 2J(N-1) + j), T(2, 2J(N-1) + j))$ and $(T(1, 2J(N-1) 2N-j), T(2, 2J(N-1) + 2N-j))$ for $0 \le J \le \lfloor (n-2)/(2(N-1)) \rfloor$. Specifically, PE(1, 1) stores the data $(T(1, 2J(N-1) + 1), T(2, 2J(N-1) + 1))$; PE$(1, N)$ stores the data $(T(1, 2J(N-1) + N), T(2, 2J(N-1) + N))$. In addition, processor PE$(i, 1)$ stores the data $(P(1, i), P(2, i))$ for $1 \le i \le m$. In our example, the initial data allocation on the $m \times N$ ($= M \times N = 6 \times 7$) RM is illustrated in Figure 8, where in fact PE(1, 7) does hold only one copy of d^2.

LEMMA 2 *Given a run-length coded text of length* $2n$ *where* $n > 1$ *and the number of processors of size* N, *the number of pipes, say* $\hat{X}$, *is equal to* $\lceil (n-1)/(N-1) \rceil$.

Proof Since two pipes have one overlapping entry; three pipes have two overlapping entry; and so on, we then have

$$n + \hat{X} - 1 \le N\hat{X} \Longrightarrow (N-1)\hat{X} \ge n - 1 \Longrightarrow \hat{X} \ge \frac{n-1}{N-1}.$$

Because $\hat{X}$ must be the smallest integer satisfying the above inequality, we select $\hat{X} = \lceil (n-1)/(N-1) \rceil$. Q.E.D.

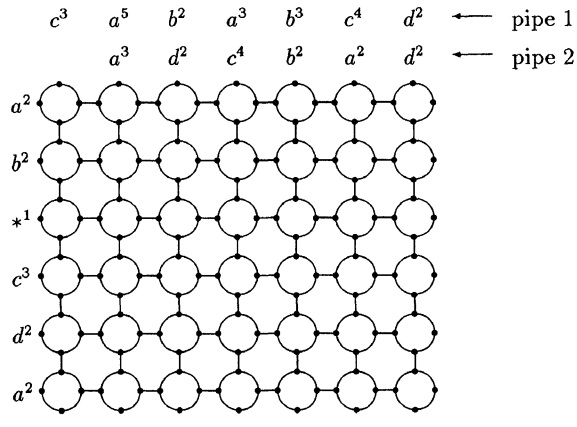


FIGURE 8 The initial ocnfiguration of the 6×7 RM.

By Lemma 2, we first partition the run-length coded text into $\hat{X}$ pipes and each pipe has one overlapping entry shared with the last pipe. In Figure 8, the last entry in the first pipe is shared with the last entry in the second pipe. In general, the last (first) entry in the odd (even) pipe is shared with the last (first) entry in the next pipe. Besides this entry-sharing feature, the text is arranged into a snakelike row-major order. For clarity, let $N = 5$, the text = $a^1b^1c^1d^1e^1f^1g^1h^1i^1j^1k^1l^1m^1$ can be arranged into the following snakelike row-major:

$$\begin{array}{lllll} a^1 & b^1 & c^1 & d^1 & e^1 \leftarrow \text{pipe 1} \\ i^1 & h^1 & g^1 & f^1 & e^1 \leftarrow \text{pipe 2} \\ i^1 & j^1 & k^1 & l^1 & m^1 \leftarrow \text{pipe 3} \end{array}$$

Our parallel algorithm process these $\hat{X}$ pipes from the $\hat{X}$th pipe to the first pipe successively and processing each pipe is similar to the parallel algorithm described in Section 3. Our partitionable parallel algorithm for this case, *i.e.*, $N < n$, is described below.

Algorithm_2

Step 1 Each processor PE$(1, j)$ for $1 \le j \le N$ broadcasts all of its own data to the other processors in the same column *via* the vertical bus system. It takes $O(\hat{X})$ time. Here, we assume that each time it takes $O(1)$ time to broadcast a data.

Step 2 and Step 3 These two steps are the same as Step 2 and Step 3 described in Algorithm_1, respectively.

For $X = \hat{X}$ **to 1** /* we process these $\hat{X}$ pipes from the last one to the first one */
begin

Step 4 Each processor first disconnects its all connections. Then, excepting the processors in the first row, each processor connects its N and E (W) ports for odd (even) X.

Step 5 Each processor in the first row, the last row, and the rows holding '+' connects its W (E) and S ports for odd (even) X when $P(1,i) = T(1, 2 \lfloor (X)/(2) \rfloor (N-1) + j)$ $(T(1, X(N-1) - j + 2))$ and $P(2,i) \le T(2, 2 \lfloor (X)/(2) \rfloor (N-1) + j)$ $(T(2, X(N-1) - j + 2))$. Otherwise do nothing.

Step 6 Each processor PE(i, j), $2 \le i \le m-1$, connects its W (E) and S ports for odd (even) X when $P(1,i) = T(1, 2 \lfloor (X)/(2) \rfloor (N-1) + j)$ $(T(1, X(N-1) - j + 2))$ and $P(2,i) = T(2, 2 \lfloor (X)/(2) \rfloor (N-1) + j)$ $(T(2, X(N-1) - j + 2))$. Otherwise do nothing.

Step 7 Each processor PE(i, j) holding '$*$' connects its W (E) and S ports for odd (even) X; connects its W and E ports.

Step 8 For pipe X, *i.e.*, the Xth pipe, if X is odd (even), processor PE(i, N) (PE$(i, 1)$), $2 \le i \le m$, holding the data sent from pipe $X+1$ and processor PE(m, j), $1 \le j \le N$, with the connection linking the W (E) and S ports first send the symbol '!' from the S port to processor PE$(1, k)$ for $1 \le k \le N$ or PE$(i, 1)$ (PE(i, N)). If each processor holding the symbol '$*$' receives the symbol '!' from its S port then it disconnects its W and E ports; disconnects its N and E (W) ports if it has; connects its N and S ports.

Step 9 Along the corresponding stairlike bus systems, processor PE(i, N) (PE$(i, 1)$), $2 \le i \le m$, holding the data sent from pipe $X+1$ and processor PE(m, j), $1 \le j \le N$, with the connection linking the W (E) and S ports, send the received data or the data $(2 \lfloor (X)/(2) \rfloor (N-1) + j,\ P(2, m))$ $((X(N-1) - j + 2, P(2, m)))$ from the S port to processor PE$(1, k)$ for $1 \le k \le N$ or PE$(i, 1)$ (PE(i, N)). Finally, PE$(i, 1)$ (PE(i, N)) keeps the received data, which will be used by pipe $X-1$, and PE$(1, k)$ reports the data $(2 \lfloor (X)/(2) \rfloor (N-1) + k,$ $T(2, 2 \lfloor (X)/(2) \rfloor (N-1) + k) - P(2, 1) + 1)$ $((X(N-1) - k + 2, T(2, X(N-1) - k + 2)$

$-P(2,1)+1)$) as the matched starting location and the received data as the matched ending location if the W (E) port of that processor receives the data.

end

For pipe 2 in our example, processor PE(3, 7) keeps the data (12, 2) and PE(5, 7) keeps the data (8, 2), which will be used by pipe 1; PE(1, 6) reports (8, 1) as the matched starting locations and (12, 2) as the matched ending location (see Figs. 9(a) and (b)). For pipe 1, PE(1, 2) reports (2, 4) as the matched starting locations and (8, 2) as the matched ending location; PE(1, 4) reports (4, 2) as the matched starting locations and (8, 2) as the matched ending location (see Figs. 9(c) and (d)). Under the same cost $O(nm)$ as in Theorem 1, we have the following result.

THEOREM 3 *For $N < n$, the SM problem for run-length coded strings with VLDCs can be solved in $O(\hat{X})$ time on the RM using $O(Nm)(= O((nm)/(\hat{X})))$ processors, where $\hat{X} = \lceil (n-1)/(N-1) \rceil$.*

B. Case 2: $N < n$ and $M < m$

Assume the RM consists of 5×5 processors and the run-length coded text and pattern are the same as in Section 3. That is, $M = 5$, $N = 5$, $m = 6$, and $n = 12$. Initially, suppose processor PE$(1, j)$, $2 \le j \le N-1$, stores the data $(T(1, 2J(N-1)+j), T(2, 2J(N-1)+j))$ and $(T(1, 2J(N-1)+2N-j), T(2, 2J(N-1)+2N-j))$ for $0 \le J \le \lfloor (n-2)/(2(N-1)) \rfloor$. Specifically, PE(1, 1) stores the data $(T(1, 2J(N-1)+1), T(2, 2J(N-1)+1))$; PE$(1, N)$ stores the data $(T(1, 2J(N-1)+N), T(2, 2J(N-1)+N))$. In addition, processor PE$(i, 1)$, $2 \le i \le M-1$, stores the data $(P(1, 2I(M-1)+i), P(2, 2I(M-1)+i))$ and $(P(1, 2I(M-1)+2M-i), P(2, 2I(M-1)+2M-i))$ for $0 \le I \le \lfloor (m-2)/(2(M-1)) \rfloor$. Specifically, PE(1, 1) stores the data $(P(1, 2I(M-1)+1), P(2, 2I(M-1)+1))$; PE$(M, 1)$ stores the data $(P(1, 2I(M-1)+M), P(2, 2I(M-1)+M))$. In our example for this case, the initial data allocation

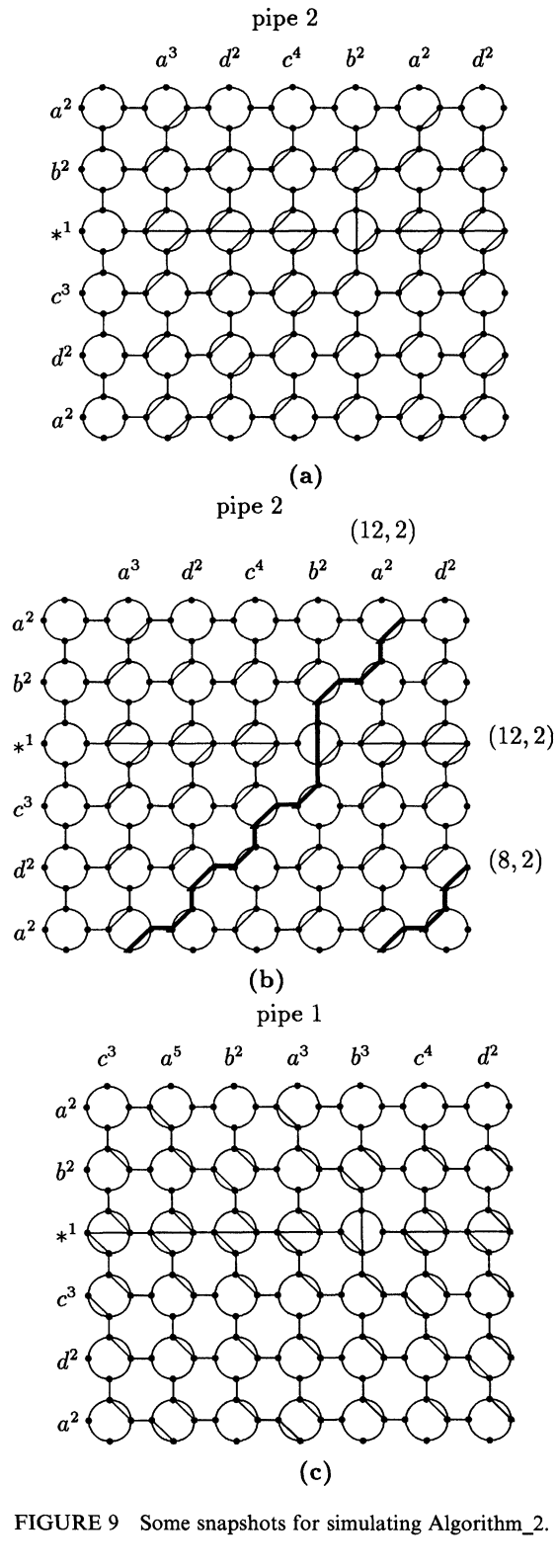


FIGURE 9 Some snapshots for simulating Algorithm_2.

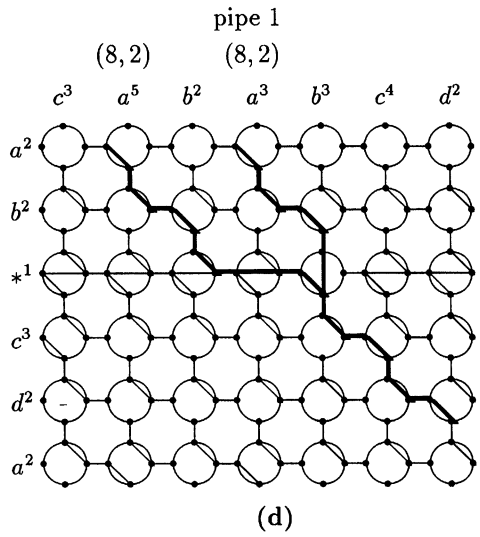


FIGURE 9 (Continued).

on the $M \times N$ $(= 5 \times 5)$ RM is illustrated in Figure 10.

By Lemma 2, it follows that the number of pipes for text and pattern are $\hat{X} = \lceil (n-1)/(N-1) \rceil$ for $n > 1$ and $\hat{Y} = \lceil (m-1)/(M-1) \rceil$ for $m > 1$, respectively. Therefore, we partition the run-length coded text (pattern) into $\hat{X}(\hat{Y})$ pipes and each pipe has one overlapping entry shared with the last pipe. For the run-length coded text in Figure 10,

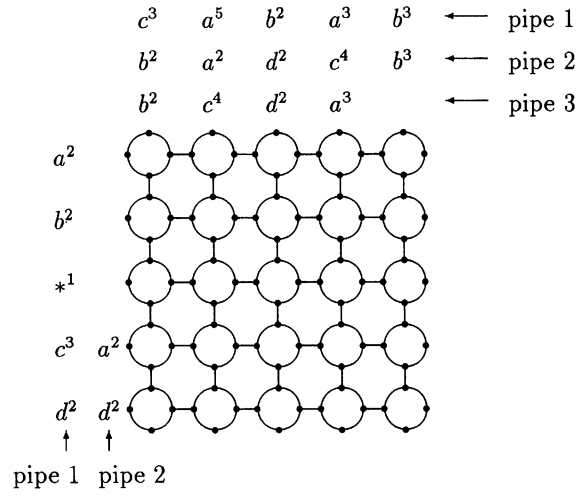


FIGURE 10 The initial configuration of the 5×5 RM.

the last entry $(= b^3)$ in the first pipe is shared with the last entry in the second pipe; the first entry $(= b^2)$ in the second pipe is shared with the first entry in the third pipe. In addition, for the run-length coded pattern in Figure 10, the last entry $(= d^2)$ in the first pipe is shared with the last entry in the second pipe. That is, the text is arranged into a snakelike row-major order and the pattern is arranged into a snakelike column-major order. Our algorithm process these $\hat{X}\hat{Y}$ pipes from pipe $\hat{Y}$ to pipe 1 for each fixed pipe X, $\hat{X} \geq X \geq 1$, successively; it is similar to Algorithm −2, but we change the roles of text and pattern each other. Our partitionable parallel algorithm for Case 2 is described below.

Algorithm_3

Step 1 Each processor PE$(1, j)$ for $1 \leq j \leq N$ broadcasts all of its own data to the others in the same column *via* the vertical bus system. It takes $O(\hat{X})$ time.

Step 2 Each processor PE$(i, 1)$ for $1 \leq i \leq M$ broadcasts all of its own data to the others in the same row *via* the horizontal bus system. It takes $O(\hat{Y})$ time.

Step 3 Each processor first disconnects its all connections. Then, in each pipe Y for $1 \leq Y \leq \hat{Y}$, the processor holding the symbol '$*$' sends a special symbol '+' to its south neighbor and north neighbor, and the others try to receive the symbol '+', from the N and S ports. This step takes $O(\hat{Y})$ time.

For $X = \hat{X}$ **to 1** /* we process these $\hat{X}$ pipes from the last one to the first one */
For $Y = \hat{Y}$ **to 1** /* we process these $\hat{Y}$ pipes from the last one to the first one */
begin

Step 4 Each processor first disconnects its all connections. Then, excepting the processors in the first (Mth) row for odd (even) Y, each processor connects its N (S) and E

ports when X is odd; connects its N (S) and W ports when X is even.

Step 5 Each processor holding $P(1,1)$, $P(1,m)$, and '+' connects its W and S (N) ports for odd X and odd (even) Y when $T(1, 2\lfloor(X)/(2)\rfloor(N-1)+j) = P(1, 2\lfloor(Y)/(2)\rfloor (M-1)+i)$ $(P(1, Y(M-1)-i+2))$ and $T(2, 2\lfloor(X)/(2)\rfloor (N-1)+j) \geq (P(2, 2\lfloor(Y)/(2)\rfloor (M-1)+i)$ $(P(2, Y(M-1)-i+2))$; each processor holding $P(1,1)$, $P(1,m)$, and '+' connects its E and S (N) for even X and odd (even) Y when $T(1, X(N-1)-j+2) = P(1, 2\lfloor(Y)/(2)\rfloor (M-1)+i)$ $(P(1, Y(M-1)-i+2))$ and $T(2, X(N-2)-i+2)$ $(P(2, 2\lfloor(Y)/(2)\rfloor (M-1)+i)$ $(P(2, Y(M-1)-i+2))$. Otherwise do nothing.

Step 6 Each processor PE(i,j), $1 \leq i \leq M$ and $1 \leq j \leq N$, connects its W and S (N) for odd X and odd (even) Y when $T(1, 2\lfloor(X)/(2)\rfloor (N-1)+j) = P(1, 2\lfloor(Y)/(2)\rfloor (M-1)+i)$ $(P(1, Y(M-1)-i+2))$ and $T(2, 2\lfloor(X)/(2)\rfloor (N-1)+j) = P(2, 2\lfloor(Y)/(2)\rfloor (M-1)+i)$ $(P(2, Y(M-1)-i+2))$; each processor PE(i,j), $1 \leq i \leq M$ and $1 \leq j \leq N$, connects its E and S (N) for even X and odd (even) Y when $T(1, X(N-1)-j+2) = P(1, 2\lfloor(Y)/(2)\rfloor (M-1)+i)$ $(P(1, Y(M-1)-i+2))$ and $T(2, X(N-1)-j+2) \geq (P(2, 2\lfloor(Y)/(2)\rfloor (M-1)+i)$ $(P(2, Y(M-1)-i+2))$. Otherwise do nothing.

Step 7 Each processor PE(i,j) holding '$*$' connects its W and E ports; connects its W and S (N) for odd X and odd (even) Y; connects its E and S (N) for even X and odd (even) Y.

Step 8

Case X is odd With the connection linking the W and S (N) for odd (even) Y, each processor PE(i,j) holding $(P(1,m), P(2,m))$, processor PE(M,j) (PE$(1,j)$) holding the data sent from pipe $Y+1$ and processor PE(i,N) holding the data sent from pipe $X+1$, where $1 \leq j \leq N$ and $1 \leq i \leq M$, first send the symbol '!' from the S (N) port to the processor PE$(1, k)$ (PE(M, k)) for $1 \leq k \leq N$ or PE$(i, 1)$. If each processor holding the symbol '$*$' receives the symbol '!' from its S (N) port then it disconnects its W and E ports; disconnects its N (S) and E ports if it has; connects its N and S ports.

Case X is even With the connection linking the E and S (N) for odd (even) Y, each processor PE(i,j) holding $(P(1,m), P(2,m))$, processor PE(M,j) (PE$(1,j)$) holding the data sent from pipe $Y+1$ and processor PE$(i,1)$ holding the data sent from pipe $X+1$, where $1 \leq j \leq N$ and $1 \leq i \leq M$, first senf the symbol '!' from the S (N) port to the processor PE$(1, k)$ (PE(M, k)) for $1 \leq k \leq N$ or PE(i,N). If each processor holding the symbol '$*$' receives the symbol '!' from its S (N) port then it disconnects its W and E ports; disconnects its N (S) and W ports if it has; connects its N and S ports.

Step 9

Case X is odd Along the corresponding stair-like bus systems, with the connection linking the W and S (N) for odd (even) Y, each processor PE(i,j) holding $(P(1, m), P(2, m))$, processor PE(M, j) (PE$(1, j)$) holding the data sent from pipe $Y+1$, and processor PE(i,N) holding the data sent from pipe $X+1$, where $1 \leq j \leq N$ and $1 \leq i \leq M$, send the

data (2 $\lfloor(X)/(2)\rfloor$ $(N-1)+j$, $P(2, m)$) if they hold ($P(1, m)$, $P(2, m)$) or the received data from the S (N) port to the processor PE(1, k) (PE(M, k)) for $1 \le k \le N$ or PE(i, 1). Finally, PE(1, k) reports the data (2 $\lfloor(X)/(2)\rfloor$ $(N-1)+k$, $T(2,$ 2 $\lfloor(X)/(2)\rfloor$ $(N-1)+k)-P(2, 1)$ + 1) as the matched starting location and the received data as the matched ending location if the W port of that processor receives the data and it holds ($P(1, 1)$, $P(2, 1)$); otherwise, PE(1, k) (PE(M k)) and PE(i, 1) keep the received data if the W port of those processors receive the data.

Case X is even Along the corresponding stair-like bus systems, with the connection linking the E and S (N) for odd (even) Y, each processor PE(i,j) holding ($P(1, m)$, $P(2, m)$), processor PE(M, j) (PE(1, j)) holding the data sent from pipe $Y+1$, and processor PE(i, 1) holding the data sent from pipe $X+1$, where $1 \le j \le N$ and $1 \le i \le M$, send the data ($X(N-1)-j+2$, $P(2, m)$) if they hold ($P(1, m)$, $P(2, m)$) or the received data from the S (N) port to the processor PE(1, k) (PE(M, k)) for $1 \le k \le N$ or PE(i, N). Finally, PE(1, k) reports the data ($X(N-1)-k+2$, $T(2, X(N-1)$ $-k+2)-P(2, 1)+1)$) as the matched starting location and the received data as the matched ending location if the E port of that processor receives the data and it holds ($P(1, 1)$, $P(2, 1)$); otherwise, PE(1, k) (PE(M, k)) and PE(i, N) keep the received data if the E port of those processors receive the data.

end

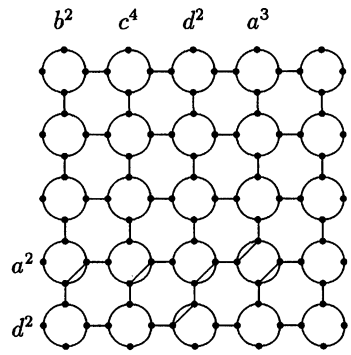
It is observed that from Step 4 to Step 9, the major difference when compared with Algorithm_1 is to replace E (W) port by W (E) port when it runs the even pipe of the run-length coded text; replace S (N) port by N (S) port when it runs the even pipe of the run-length coded pattern.

For $X = 3$ and $Y = 2$ in our example, the processor PE(5, 3) keeps the data (12, 2) coming from PE(4, 4) (see Figs. 11(a) and (b)). For $X = 3$ and $Y = 1$, PE(2, 1) and PE(3, 1) keep the data (12, 2) coming from PE(5, 3) (see Figs. 11(c) and (d)). For $X = 2$ and $Y = 2$, PE(5, 3) keeps the data (8, 2) coming from PE(4, 2) (see Figs. 11(e) and (f). For $X = 2$ and $Y = 1$, PE(1, 2) reports the data (8, 1) as the matched starting location and (12, 2) coming from PE(2, 1) as the matched ending location; PE(2, 5) and PE(3, 5) keep the data (8, 2) coming from PE(5, 3) (see Figs. 11(g) and (h)). For $X = 1$ and $Y = 2$, no processor keeps the data (see Figs. 11(i) and (j)). For $X = 1$ and $Y = 1$, PE(1, 2) reports the data (2, 4) as the matched starting location and (8, 2) as the matched ending location; PE(1, 4) reports the data (4, 2) as the matched starting location and (8, 2) as the matched ending location (see Figs. 11(k) and (l)). Under the same cost $O(nm)$ as in Theorem 1, we have the following result.

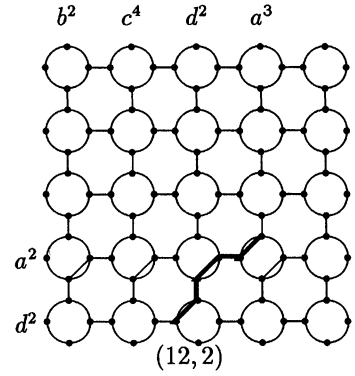
THEOREM 4 *For $N < n$ and $M < m$, the SM problem for run-length coded strings with VLDCs can be solved in $O(\hat{X}\hat{Y})$ time on the RM using $O(NM) = (= O((nm)/(\hat{X}\hat{Y})))$ processors, where $\hat{X} = \lceil (n-1)/(N-1) \rceil$ and $\hat{Y} = \lceil (m-1/(M-1) \rceil$.*

5. CONCLUSIONS

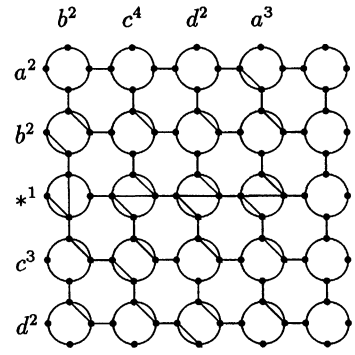
The significance of string-matching for run-length coded strings with VLDCs is due to its popular use



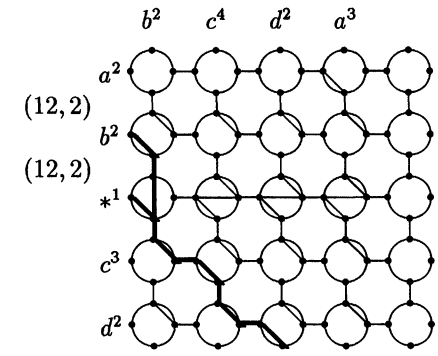
(a) X=3 and Y=2



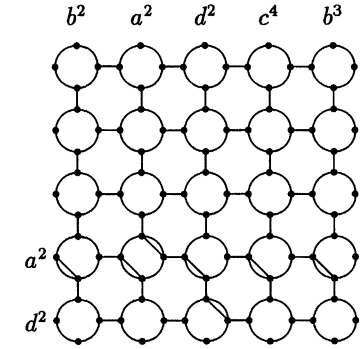
(b) X=3 and Y=2



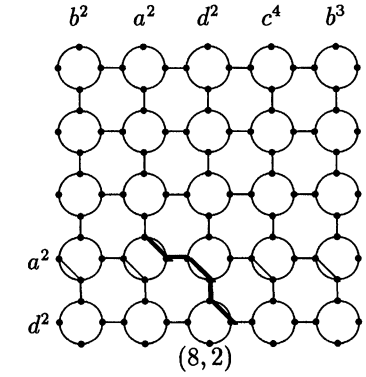
(c) X=3 and Y=1



(d) X=3 and Y=1



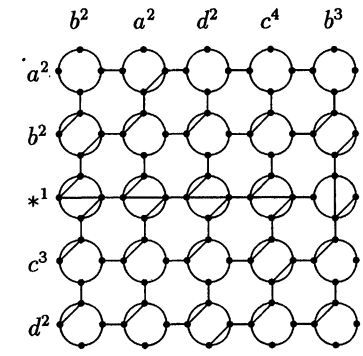
(e) X=2 and Y=2



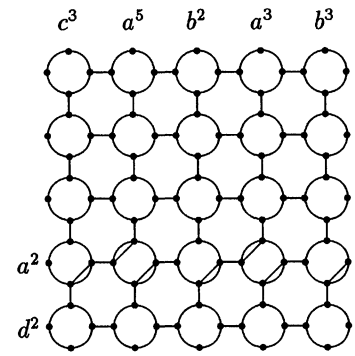
(f) X=2 and Y=2

FIGURE 11 Some snapshots for simulating Algorithm_3.

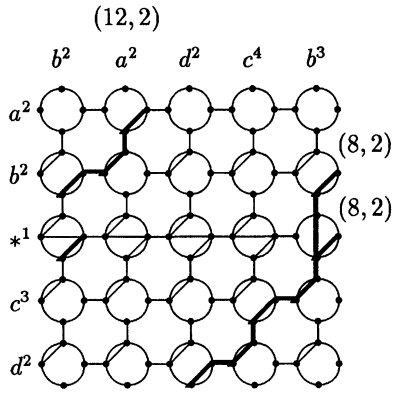
in pattern recognition, edit operations, and so on. Given a run-length coded text (pattern) of length $2n$ ($2m$), the main contributions of this paper are twofold: first we have presented an $O(1)$ time parallel SM algorithm for run-length coded strings with VLDCs on the RM with $O(nm)$ processors,



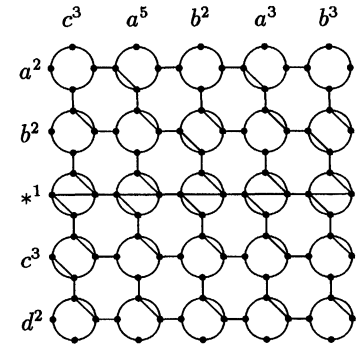
(g) X=2 and Y=1



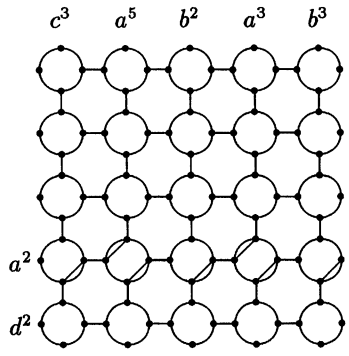
(j) X=1 and Y=2



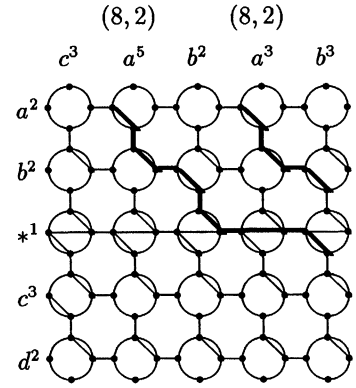
(h) X=2 and Y=1



(k) X=1 and Y=1



(i) X=1 and Y=2



(l) X=1 and Y=1

FIGURE 11 (Continued).

and this result generalizes the results in [2, 3]; second, the most important is that we have presented a partitionable parallel SM algorithm for solving the same problem, and this partitionable parallel algorithm on RMs is very suitable for VLSI modular implementation.

Acknowledgements

The authors are indebted to the anonumous referees and Prof. S. Olariu for reviewing and processing this paper.

References

[1] Chen, G. H. (1992). "An $O(1)$ time algorithm for string matching", *Intern. J. Computer Math.*, **42**, 185–191.
[2] Chung, K. L., "An improved $O(1)$ time algorithm for string matching", Research Report, *Dept. of Information Mgmt.*, National Taiwan Institute of Technology, Dec. 1993.
[3] Chung, K. L. (1996). "$O(1)$ time parallel string-matching algorithm with VLDCs", *Pattern Recognition Letters*, **17**, 475–479.
[4] Crochemore, M. and Rytter, W. (1994). *Text Algorithms*, Oxford University Press, New York.
[5] Li, H. and Stout, Q. F. (Eds.), (1991). *Reconfigurable Massively Parallel Computers*, Prentice-Hall, Englewood Cliffs, N.J.
[6] Lin, R. and Olariu, S. (1995). Reconfigurable buses with shift switching-concepts and applications", *IEEE Trans. Parallel Distrib. Syst.*, **6**, 93–102.
[7] Miller, R., Prasanna-Kumar, V. K., Reisis, D. I. and Stout, Q. F. (1993). "Parallel computations on reconfigurable meshes", *IEEE Trans. on Comput.*, **42**, 678–692.
[8] Olariu, S., Schwing, J. L. and Zhang, J. (1991). "Fundamental algorithms on reconfigurable meshes", *Proc. 29th Annual Allerton Conference on Communications, Control, and Comput.*, pp. 811–820.
[9] Olariu, S., Schwing, J. L. and Zhang, J. (1994). "Fundamental data movement on reconfigurable meshes", *Internat. J. High Speed Comput.*, **6**, 311–323.
[10] Olariu, S. and Schwing, J. L. (1996). "A novel deterministic sampling scheme with application to broadcast-efficient sorting on the reconfigurable mesh", *J. Parallel and Distributed Comput.*, **32**, 215–222.
[11] Rothstein, J. (1976). "On the ultimate limitations of parallel processing", *Proc. Int. Conf on Parallel Processing*, pp. 206–212.
[12] Rothstein, J. (1988). "Bus automata, brains, and mental models", *IEEE Trans. Syst., Man and Cybern.*, **SMC-18**, pp. 522–531.
[13] Wang, B. F. and Chen, G. H. (1990). "Constant time algorithms for the transitive closure and some related graph problems on processor arrays with a reconfigurable bus system", *IEEE Trans. Parallel Distrib. Syst.*, **1**, 500–507.
[14] Zhang, K., Wang, T. L. and Shasha, D. (1992). "Fast serial and parallel algorithms for approximate tree matching with VLDCs", *Combinatorial Pattern Matching Conf.*, pp. 148–158.

Author's Biographies

Hsiu-Niang Chen received the B.S. degree in Business Mathematics in 100 from Soochow University and the M.S. degree in Information Management in 1993 from National Taiwan Institute of Technology. Since 1993, she has been a lecturer in the Department of Information Management of Van-Nung Institute of Technology. Her current research interests include parallel computing, string matching, and image processing.

Kuo-Liang Chung received the B.S., M.S., and Ph.D. degrees in Computer Science and Information Engineering from National Taiwan University of R.O.C. Since 1995, he has been a professor in the Department of Information Management of the National Taiwan Institute of Technology. His current research interests include image processing, computer graphics, data compression, and high speed computing. Dr. Chung has authored more than 65 papers in refereed international well-known journals. He is a member of ACM, IAPR, IEEE, and SIAM.