Constant Time Algorithm for Template Matching on a Reconfigurable Array of Processors

S. J. HORNG

Department of Electrical Engineering, National Taiwan Institute of Technology, 43, Section 4, Kee-Lung Road, Taipei, Taiwan, ROC

A *reconfigurable array of processors (RAP)* is defined to be an array of processors connected to a reconfigurable bus system whose configuration can be dynamically changed. For an $n \times n$ image and $m \times m$ template, a constant time algorithm for template matching is proposed on a RAP with $m \times m \times n \times n \times n$ processors when the domain of the image and the template is Boolean. Even if the domain is integer and each integer is bounded and represented by a *q-bits* binary sequence, a constant or $O(\log^* m)$ time algorithm is still obtained. The number of processors needed is increased to $2qm^2 \times 2qm^2 \times max\{m, n\} \times n \times n$ and $m^2 \times m^2 \times max\{m^2, n\} \times n \times n$, respectively. An $O(\log m)$ time algorithm also derived for the domain is integer or real but the number of processors is reduced to $m \times m \times n \times n \times n$.

Received August 1991, revised February 1992

1. INTRODUCTION

Most image processing and computer vision problems are intensive in computation and involve parallel processing on the pixels of the image. Several architectures have been proposed for parallel image processing: these include cellular arrays [27], pyramid structures [36], reconfigurable meshes [19], hypercube machines [8–12, 24–26], and the reconfigurable array of processors (RAP). The algorithm proposed in this paper is based on a RAP.

In image processing and computer vision, template matching is a basic operation used for filtering, edge detection, image registration and object detection [32]. Template matching can be described as comparing a template (window) with all possible windows of the image. The result of each window operation is stored in a location corresponding to the top left-hand corner of the window. Let $im(i_1, i_0)$ represent an $n \times n$ image where $0 \leqslant i_1, i_0 < n$. Let $w(i_4, i_3)$ represent an $m \times m$ template where $0 \leqslant i_4, i_3 < m$. Assume the image and the template have the same domain D. Then the result $c(i_1, i_0), 0 \leqslant i_1, i_0 < n$, is given below:

$$c(i_1, i_0) = \sum_{i_4=0}^{m-1} \sum_{i_3=0}^{m-1} im((i_1 + i_4) \bmod n, (i_0 + i_3) \bmod n) \times w(i_4, i_3) \quad (1)$$

From Equation (1), for each $c(i_1, i_0)$ there are m by m product terms. The sum of m by m product terms takes $O(m^2)$ time in a unit processor. Therefore, Equation (1) can be computed in $O(n^2 \times m^2)$ time in a unit processor system.

Many researchers have proposed their algorithms for template matching [8–12, 24–26, 32]. Unlike previous algorithms, however, our algorithm runs in constant time and uses polynomial processors.

Modern VLSI technology makes possible the construction of parallel processing systems that employ thousands of processors. Recent developments in VLSI technology have inspired many researchers to do research on parallel computation. Parallel computation is not only of pure theoretical interest but also of practical value in applications. Owing to the inherent parallelism, the execution time of an algorithm cannot be significantly improved even on the concurrent read concurrent write (CRCW) model by simply increasing the number of processors. Higher time complexity may be required when algorithms are developed on more feasible parallel systems which are constructed by an interconnection network. One very attractive interconnection scheme is the *two-dimensional mesh-connected computer* (*mesh*) because of its simplicity and regularity. However, the communication diameter of 2D $n^{1/2} \times n^{1/2}$ mesh is $\Theta(n^{1/2})$. Hence, the lower bound for those algorithms that require global exchange of data are $\Omega(n^{1/2})$. Recently, many researchers enhanced long distance communication by adding additional communication links to the mesh in order to improve the execution time of an algorithm. These include the pyramid computer [7, 21, 36–37], the mesh-of-trees [14, 23], and meshes with broadcast buses [1, 3–5, 13, 34–35]. Nevertheless, these architectures are static in nature. That is, the architecture cannot be altered during the execution of an algorithm. There are many algorithms that require their supported interconnection scheme to be changed during their execution. Therefore, an array of processors connected to a reconfigurable bus system is more suitable for such algorithms.

A *reconfigurable bus system* is a system whose configuration can be dynamically changed. These include the bus automaton [28, 30], reconfigurable mesh [17], polymorphic-torus network [15] and an array of processors connected to a reconfigurable bus system (RAP) [2, 38–41]. A *bus automaton* consists of a cellular automation augmented with a locally siwtchable global communication network. Problems that have been studied on the *bus automaton* involve pattern recognition [16, 29, 31], parsing of formal language [22] and longest common subsequence [6]. A *reconfigurable mesh* consists of a square array of processors which are connected to a gird-shaped reconfigurable bus system. The processors are situated at the grid intersection points of the bus. Each processor has four locally controllable bus switches to adjust the configuration of the bus system. Miller *et al.* [18–20] have studied several problems such as graph, image and geometry on *reconfigurable mesh.* The *polymorphic-torus network* is similar to *reconfigurable mesh* except that in the *polymorphic-torus network* there are arbitrary crossbars at each processor to control connections between the north, south, east, and west bus ports. Therefore it can embed tree, ring, mesh, pyramid and hypercube efficiently by establishing the programmable local switches [15]. The RAP is similar to a *reconfigurable mesh* except that in the RAP they are multidimensional. Therefore, the *reconfigurable mesh* and *polymorphic-torus network* are functionally equivalent to a 2D RAP. Wang *et al.* [39] have shown that the 2D RAP is at least as powerful as the CRCW shared-memory computer. Efficient algorithms on a RAP have been developed for many applications such as graph [18, 38], image [19], geometry [20], sorting [41] and string processing [6].

In this paper, when the domain of both the image and the template is Boolean, a constant time algorithm for template matching is proposed on a RAP with $m \times m \times n \times n \times n$ processors. However, if the domain of the image and the template are integers and each integer is bounded and represented by a *q-bits* binary sequence, a constant or $O(\log^* m)$ time algorithm is still obtained. But the number of processors is increased to $2qm^2 \times 2qm^2 \times max\{m, n\} \times n \times n$ and $m^2 \times m^2 \times max\{m^2, n\} \times n \times n$, respectively. Furthermore, we also derive an $O(\log m)$ time algorithm for which the domain is integer or real using $m \times m \times n \times n \times n$ processors.

The rest of the paper is organized as follows. The RAP on which the constant time algorithms are based is described in Section 2. Section 3 presents several basic data operations. The constant time algorithm for template matching is proposed in Section 4. Finally, some concluding remarks are included in the last section.

2. RAP AND BASIC NOTATIONS

A *k-dimensional* (*k-D*) RAP of size N contains N processors arranged in a k-D grid. That is, the bus system can be thought of as logically arranged as in a *k-D* array $A(n_{k-1}, n_{k-2}, ..., n_0)$, where n_j, $0 \leqslant j < k$, is the size of the jth dimension and $N = n_{k-1} \times n_{k-2} \times \ldots \times n_0$. Each processor is identified by a unique *k-tuple* index $(i_{k-1}, i_{k-2}, ..., i_0)$, $0 \leqslant j < k$, $0 \leqslant i_j < n_j$. The processor with index $(i_{k-1}, i_{k-2}, ..., i_0)$ is denoted by $P_{i_{k-1}, i_{k-2}, ..., i_0}$. Each processor has $2k$ ports denoted by $-S_j$, $+S_j$, $0 \leqslant j < k$. Processor $P_{i_{k-1}, i_{k-2}, i_j, ..., i_o}$ connects its port $+S_j$ to the i_j-dimension bus and processor $P_{i_{k-1}, i_{k-2}, ..., i_{j+1}, ..., i_0}$ also connects its port $-S_j$ to the i_j-dimension bus, for $0 \leqslant j < k$ and $0 \leqslant i_j < n_j$.

Each processor can perform arithmetic and logic operations. We assume that each arithmetic (logic) operation takes one time unit. Any configuration of the bus system is derivable by properly establishing the local connections among ports within each processor. Each processor can communicate with other processors by broadcasting data on the bus system. We also assume that each broadcasting takes one time unit. If more than one processor attempts to broadcast data on the same bus simultaneously, then a resolution scheme should be applied; otherwise, a collision occurs and the final data received are unexpected. We allow multiple processors to broadcast data on the different buses simultaneously at the same time unit.

To represent the local connection within a processor, we use the notation $\{g_0\}, \{g_1\}, ..., \{g_{t-1}\}$, where g_i, $0 \leqslant i < t$, denotes a group of buses that are connected together [38–41]. That is, each processor has t connections and each connection is described by g_i, $0 \leqslant i < t$.

Let $var(i_{k-1}, i_{k-2}, ..., i_0)$ denote the local variable *var* (memory or register) in a processor with index $i_{k-1}, i_{k-2}, ..., i_0$. For example, $sum(0, 0, 1)$ is a local variable *sum* of processor $P_{0,0,1}$.

A RAP is operated in a single instruction stream, multiple data streams (SIMD) model. An *enable/disable* mask can be used to select a subset of the processors that are to perform an instruction. Only the enabled processors will perform the instruction. The remaining processors will be idle. The set of enabled processors can change from instruction to instruction.

The bus bandwidth between processors is not unlimited. We assume the bus bandwidth is bounded by *q-bits* wide where q is a constant. Therefore, *q-bits* data can be transferred between processors in a unit time. The I/O loading (upload or download) time is fully dependent on how complex the I/O interface between processors and peripherals will be. An application for parity check of a binary sequence was proposed by Ben-Asher *et al.* [2]. They used an *initializing network* to deliver the input binary sequence to the switches of a 2D RAP. Therefore, the complexity of an algorithm is assumed the sum of the computation time of processors and the communication time among processors. This assumption was also used by many researchers [2, 6, 18–20, 38–41].

We show an example for a 2D 4×4 RAP in Figure 1(a). By establishing the local connection $\{-S_0, +S_1\}, \{-S_1, +S_0\}$ for each processor, a dog leg

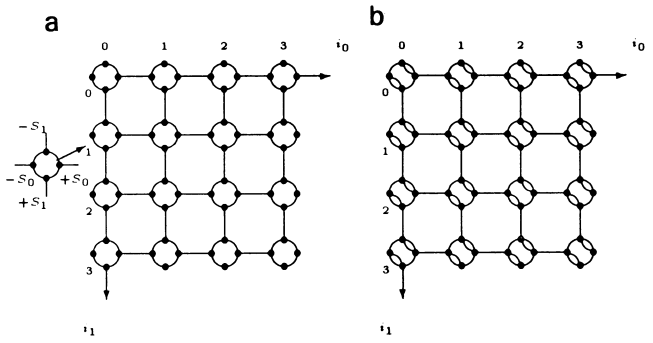


FIGURE 1. A 2-D $n_1 \times n_0$ RAP, where $n_1 = n_0 = 4$. (*a*) A 4×4 RAP. (b) A dog leg configuration.

configuration is shown in Figure 1(b). Figure 2 shows a 3D $4 \times 4 \times 4$ RAP.

3. BASIC DATA OPERATION

Some data operations will be described in this section. These data operations are used for deriving efficient algorithms in Section 4.

3.1. Transpose of a matrix

Let $A = a(i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, be a data matrix. Then the transpose of A, A^t, is defined to be $A^t = ta(i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, where $ta(i_0, i_1) = a(i_1, i_0)$. Consider a 3D $n \times n \times n$ RAP. Conceptually, a 3D *RAP* can be partitioned into n 2D $n \times n$ RAP_{i_2}'s which are denoted by $2D\text{-}RAP_{i_2}$'s, $0 \leqslant i_2 < n$. A constant time transpose algorithm is presented on a 3D $n \times n \times n$ RAP. Initially, the data matrix, A, is stored in the local variable $a(0, i_1, i_0)$ of 2D RAP. Finally, the transpose matrix, A^t, is stored in the local variable $ta(0, i_0, i_1)$. The transpose algorithm (TA) is listed as follows.

Procedure $TA(a, ta)$:

Step 1: // Step 1–3: Copy the i_1th row of $a(i_1, i_0)$ to the diagonal of $2D\text{-}RAP_{i_1}$. //

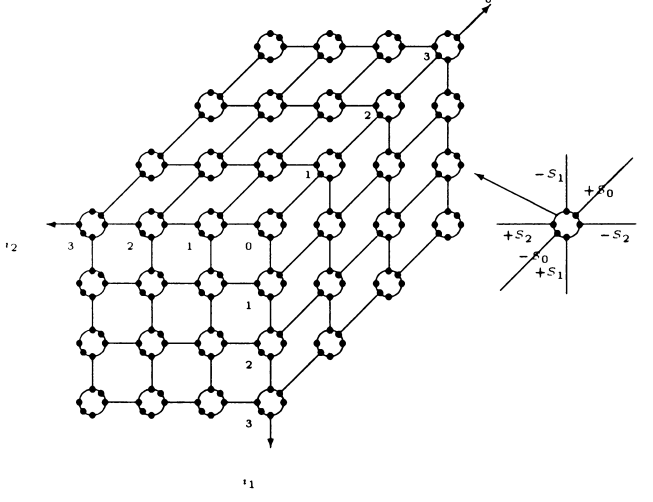


FIGURE 2. A $4 \times 4 \times 4$ RAP.

Set $ta(0, i_1, i_0)$ to $a(0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$.

Step 2:

Copy $ta(0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, to $ta(i_1, i_1, i_0)$, through the i_2-dimension bus (all processors establish the local connection $\{-S_2, +S_2\}$) on which it is connected.

Step 3:

Copy $ta(i_1, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, to $ta(i_1, i_0, i_0)$, through the i_1-dimension bus (all processors establish the local connection $\{-S_1, +S_1\}$) on which it is connected.

Step. 4: // Step 4–5: Copy the diagonal of $2D\text{-}RAP_{i_1}$ to the i_1th column of $ta(i_0, i_1)$. //

Copy $ta(i_1, i_0, i_0)$, $0 \leqslant i_1, i_0 < n$, to $ta(i_1, i_0, i_1)$, through the i_0-dimension bus (all processors establish the local connection $\{-S_0, +S_0\}$) on which it is connected.

Step 5:

Copy $ta(i_1, i_0, i_1)$, $0 \leqslant i_1, i_0 < n$, to $ta(0, i_0, i_1)$, through the i_2-dimension bus on which it is connected.

End TA.

LEMMA 1. *The TA procedure can be executed in* $O(1)$ *time on a 3D* $n \times n \times n$ *RAP. Initially, the data matrix,* A, *is stored in the local variable* $a(0, i_1, i_0)$ *of* $2D\text{-}RAP_0$ *for all* $0 \leqslant i_1, i_0 < n$.

Proof. Clearly, the algorithm is correct. The time complexity is analyzed as follows. Steps 1, 2, 3, 4 and 5 take $O(1)$ time, respectively. Hence, the time complexity is $O(1)$. QED

3.2. Rotation

Let $A = a(i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, be a data matrix. Assume the data matrix, A, is stored in the local variable $a(0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, of $2D\text{-}RAP_0$. Define $RUA(a, i)$, $0 \leqslant i < n$, as a circular shift up operation in which the data of i_1 row of A in processor P_{0,i_1,i_0}, $0 \leqslant i_1, i_0 < n$, is moved to the $(i_1 - i) \bmod n$ row of A in processor $P_{0,(i_1-i) \bmod n, i_0}$. The rotate up algorithm (RUA) is listed as follows.

Procedure $RUA(a, i)$:

Step 1:

Copy $a(0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, to $a(i_1, i_1, i_0)$ through the i_2-dimension bus.

Step 2:

Copy $a(i_1, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, to $a(i_1, (i_1 - i) \bmod n, i_0)$ through the i_1-dimension bus.

Step 3:

Copy $a(i_1, (i_1 - i) \bmod n, i_0)$, $0 \leqslant i_1, i_0 < n$, to $a(0, (i_1 - i) \bmod n, i_0)$ through the i_2-dimension bus.

End RUA.

Define $RLA(a, j)$, $0 \leqslant j < n$, as a circular shift left operation in which the data of i_0 column of A in processor P_{0,i_1,i_0}, $0 \leqslant i_1, i_0 < n$, is moved to the

$(i_0 - j)\ mod\ n$ column of A in processor $P_{0,i_1,(i_0-j)\ mod\ n}$. The rotate left algorithm (RLA) is listed as follows.

Procedure $RLA(a, j)$:

Step 1:
  Call $TA(a, b)$.
Step 2:
  Call $RUA(b, j)$.
Step 3:
  Call $TA(b, a)$.

End RLA.

Finally, define $RULA(a, i, j)$, $0 \leqslant i, j < n$, as a circular shift up and left operation in which the data of i_1 row and i_0 column of A in processor P_{0,i_1,i_0}, $0 \leqslant i_1, i_0 < n$, is moved to the $(i_1 - i)\ mod\ n$ row and $(i_0 - j)\ mod\ n$ column of A in processor $P_{0,(i_1-i)\ mod\ n,(i_0-j)\ mod\ n}$. The rotate up and left algorithm (RULA) is listed as follows.

Procedure $RULA(a, i, j)$:

Step 1:
  Call $RUA(a, i)$.
Step 2:
  Call $RLA(a, j)$.

End RULA.

LEMMA 2. *The RUA, RLA, and RULA procedures can be executed in O(1) time on a 3D $n \times n \times n$ RAP, respectively. Initially, the data matrix, A, is stored in the local variable $a(0, i_1, i_0)$ of 2D-RAP_0 for all $0 \leqslant i_1, i_0 < n$.*

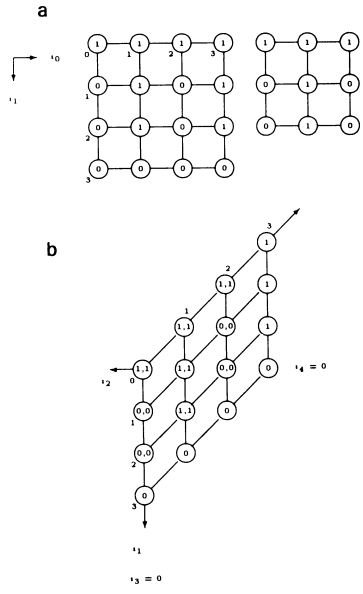
Proof. We prove that the *RLA* procedure is correct. The correctness of other procedures clearly follows. Let $a'(i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, be the matrix after matrix $a(i_1, i_0)$ is rotated left j positions, where $a'(i_1, i_0) = a(i_1, (i_0 + j)\ mod\ n)$. After Step 1 of *RLA* procedure, $b(i_1, i_0) = a(i_0, i_1)$, $0 \leqslant i_1, i_0 < n$, by Lemma 1. Let $f(i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, be the matrix after matrix $b(i_1, i_0)$ is rotated up j positions by Step 2 of *RLA* procedure, where $f(i_1, i_0) = b((i_1 + j)\ mod\ n, i_0)$. Let $d(i_1, i_0)$ be the transpose matrix of f. Then after Step 3 of the *RLA* procedure,

$$\begin{aligned} d(i_1, i_0) &= f(i_0, i_1), \quad 0 \leqslant i_1, \quad i_0 < n \\ &= b((i_0 + j)\ mod\ n, i_1) \\ &= a(i_1, (i_0 + j)\ mod\ n) \\ &= a'(i_1, i_0) \end{aligned}$$

The $O(1)$ time complexity can be easily verified for each procedure.

3.3. Logical operations

LEMMA 3. (*Miller et al.* [19]): *Given a linear RAP of size n (or an $n \times n$ RAP), in which each processor stores a Boolean data, the logical* OR (*or* AND) *of these data can be computed in O(1) time. Let the logical* OR (or AND) *algorithm be denoted as ORA(data) (or ANDA(data)).*



3.4. Linear matrix

Let $A = a(i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, be a 2D data matrix. The linear matrix of A, A', is defined to be $A' = b(r)$, $0 \leqslant r < n^2$, where $r = i_1 + i_0 \times n$. Assume the data matrix, A, is initially stored in the local variable $a(0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, of 2D-RAP_0. Finally, the linear matrix, A', is stored in the local variable $b(0, i_1 + i_0 \times n, 0)$. The linear operation algorithm (LOA) is listed as follows.

Procedure $LOA(a, b)$:

Step 1: // Copy the i_0th column of $a(i_1, i_0)$ to the diagonal of 2D-RAP_{i_0}. //
  1. Copy $a(0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, to $a(i_0, i_1, i_0)$ through the i_2-dimension bus.
  2. Copy $a(i_0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, to $a(i_0, i_1, i_1)$ through the i_0-dimension bus.

Step 2: // Shift down the i_0th column of $a(i_1, i_0)$ at the diagonal of 2D-RAP_{i_0} by $i_0 \times n$ positions and put the result to b. //
  1. Copy $a(i_0, i_1, i_1)$, $0 \leqslant i_1, i_0 < n$, to $a(i_0, i_1 + i_0 \times n, i_1)$ through the i_1-dimension bus.
  2. Copy $a(i_0, i_1 + i_0 \times n, i_1)$, $0 \leqslant i_1, i_0 < n$, to $a(i_0, i_1 + i_0 \times n, 0)$ through the i_0-dimension bus.
  3. Copy $a(i_0, i_1 + i_0 \times n, 0)$, $0 \leqslant i_1, i_0 < n$, to $b(0, i_1 + i_0 \times n, 0)$ through the i_2-dimension bus.

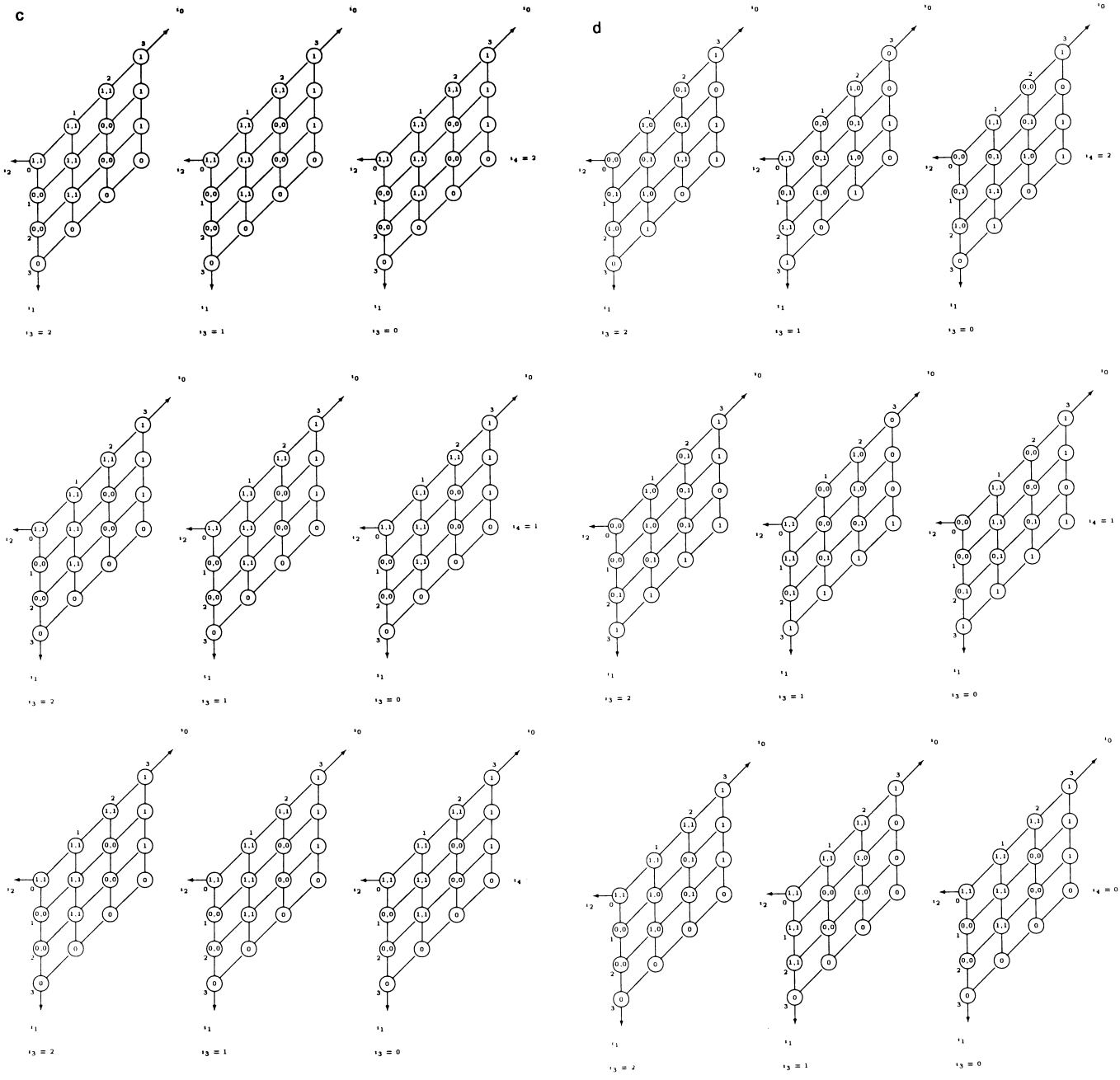
End LOA.

LEMMA 4. *The LOA procedure can be executed in O(1) time on a 3D $n \times n^2 \times n$ RAP. Initially, the data matrix, A, is stored in the local variable $a(0, i_1, i_0)$ of 2D-RAP_0 for all $0 \leqslant i_1, i_0 < n$.*

3.5. Summation

LEMMA 5. (*Wang et al.* [40] *and Ben-Asher et al.* [2]): *Let nis, the summation of i integers, be defined as*

$$nis = \sum_{i_0=0}^{i-1} a_{i_0}$$



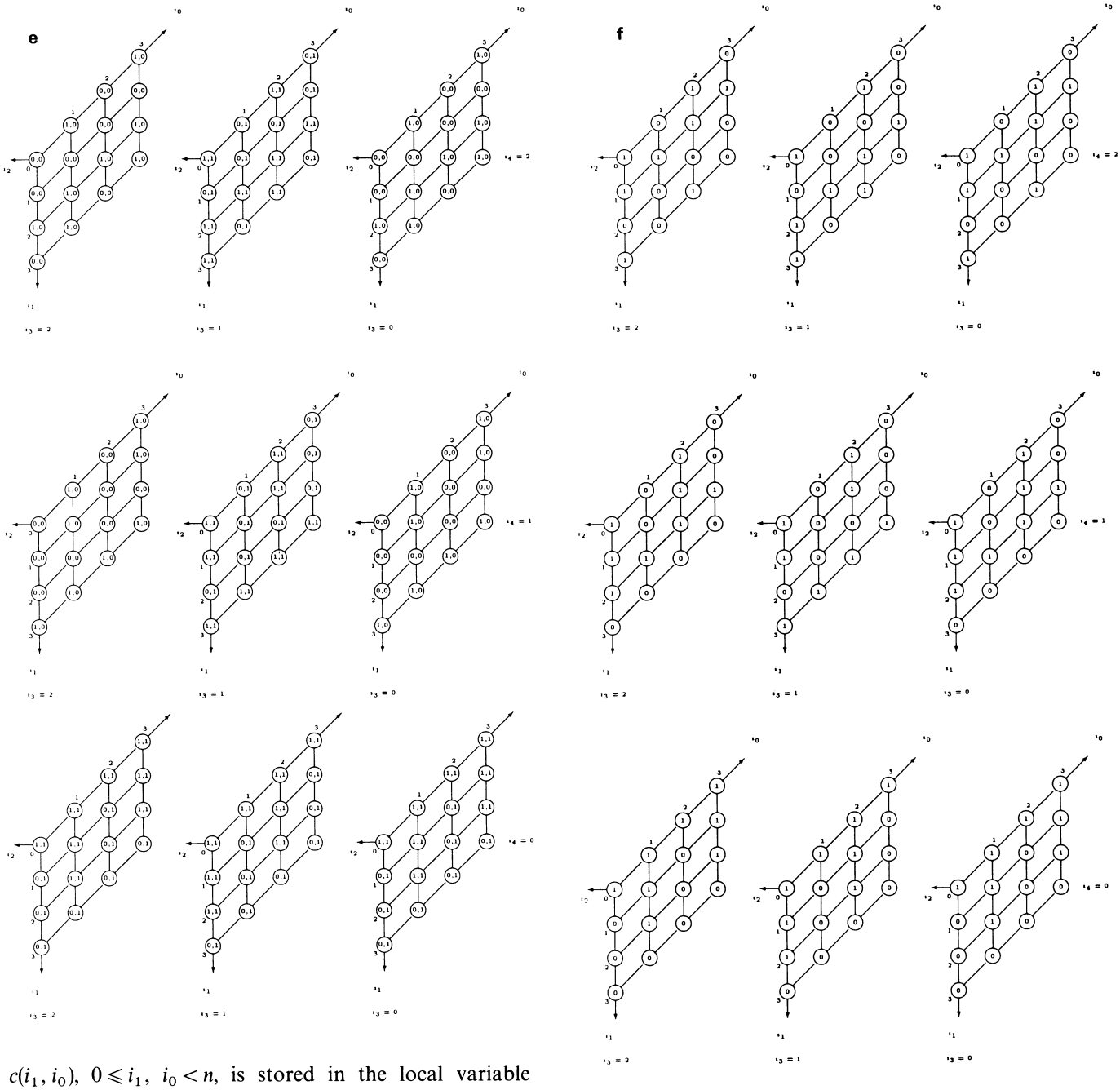
where a_{i_0} and nis are bounded and represented by a q-bits binary sequence and q is a constant. That is, $a_{i_0} = b_{i_0,q-1}, b_{i_0,q-2}, \ldots, b_{i_0,0}$ and $nis = s_{m-1}, s_{m-2}, \ldots, s_0$, where $nis = (a_{i-1} + a_{i-2} \ldots + a_0) \bmod 2^q$. Initially, $b_{i_0,r}$, $0 \leqslant i_0 < i$, $0 \leqslant r < q$, is stored in processor $P_{(2r+1)i+i_0,0}$ by Wang et al. [40] or a_{i_0}, $0 \leqslant i_0 < i$ is stored in processor $P_{0,i_0,0}$ by Ben-Asher et al. [2], respectively. The summation of i integers can be computed in $O(1)$ time (Wang et al. [40]) and $O(\log^* i)$ time (Ben-Asher et al. [2]) on a 2D $2qi \times 2qi$ RAP and 3D $i \times i \times i$ RAP, respectively. $\log^*$ *i denotes log ... * times ... log i.*

If A is a 2D integer matrix and matrix A is stored in a 2D RAP, we can first transfer matrix A into a linear matrix B by Lemma 4 then sum each element of matrix B by Lemma 5. Each integer and the sum are bounded as those assumed by Lemma 5. This leads to the following lemma.

LEMMA 6. *The summation of $i \times i$ integers can be computed in $O(1)$ time and $O(\log^* i)$ time on a 3D $i \times 2qi^2 \times 2qi^2$ RAP and 3D $i^2 \times i^2 \times i^2$ RAP, respectively.*

4. TEMPLATE MATCHING WITH n^3m^2 PROCESSOR ELEMENTS

Assume the domain D of the image and the template is Boolean. The product and sum operators are logical EQ and AND, respectively. Initially, the image and template are stored in the local variables $im(0, 0, 0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$ and $w(0, 0, 0, i_4, i_3)$, $0 \leqslant i_4, i_3 < m$ of 2D-$RAP_{0,0,0,i_1,i_0}$, respectively. Finally, the result,



$c(i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, is stored in the local variable $c(0, 0, 0, i_1, i_0)$ of $2D\text{-}RAP_{0,0,0}$.

There are $m \times m$ product terms for computing each element of Equation (1). Let the $m \times m$ product terms of $c(i_1, i_0)$, $im((i_1 + i_4) \bmod n, (i_0 + i_3) \bmod n)$ EQ $w(i_4, i_3)$, be computed in processor $P_{i_4,i_3,0,i_1,i_0}$ of $2D\text{-}RAP_{i_4,i_3,0}$ for all $0 < i_4, i_3 < m$ and $0 \leqslant i_1, i_0 < n$. In order to implement this idea, first copy the image and template from the $2D\text{-}RAP_{0,0,0}$ to all $2D\text{-}RAP_{i_4,i_3,0}$, $0 \leqslant i_4, i_3 < m$. Then each $2D\text{-}RAP_{i_4,i_3,0}$, $0 \leqslant i_4, i_3 < m$, aligns the image and the template data based on its index. That is, perform $RULA(im, i_4, i_3)$ and $RULA(w, i_4, i_3)$ in parallel, respectively. Then copy $w(i_4, i_3, 0, 0, 0)$ to $w(i_4, i_3, 0, i_1, i_0)$ for all $0 \leqslant i_4, i_3 < m$ and $0 \leqslant i_i, i_0 < n$.

Therefore, each processor has one image element and one template element. All processors compute the product terms simultaneously. Finally, the $n \times n$ sum of products are moved to $2D\text{-}RAP_{0,0,0}$. The template matching algorithm (TMA) is listed as follows.

Procedure $TMA(im, w, c)$:

Step 1: // Copy the image and template of $2D\text{-}RAP_{0,0,0}$ to all $2D\text{-}RAP_{i_4,i_3,0}$, $0 \leqslant i_4, i_3 < m$. //

1. Copy $im(0, 0, 0, i_1, i_0)$ to $im(0, i_3, 0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, $1 \leqslant i_3 < m$, through the i_3-dimension bus (all processors establish the local connection $\{-S_3, +S_3\}$).
2. Copy $im(0, i_3, 0, i_1, i_0)$ to $im(i_4, i_3, 0, i_1, i_0)$, $0 \leqslant i_1, i_0 < n$, $1 \leqslant i_4 < m$, $0 \leqslant i_3 < m$, through the i_4-dimension bus (all processors establish the local connection $\{-S_4, +S_4\}$).

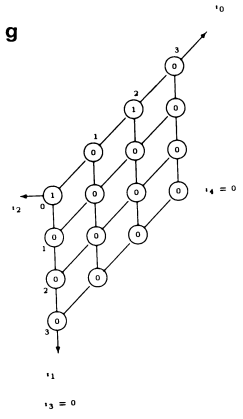


FIGURE 3. (a) A 4 × 4 image and 3 × 3 template. (b) Initially, the image and template are stored in the local variables $im(0, 0, 0, i_1, i_0)$ and $w(0, 0, 0, i_4, i_3)$, respectively, of $2D\text{-}RAP_{0,0,0}$ for all $0 \leqslant i_4, i_3 < m$, $0 \leqslant i_1, i_0 < n$. (c) After Step 1. (d) After Step 2.2. (e) After Step 2.3. (f) After Step 3. (g) After Step 4.

3. Copy $w(0, 0, 0, i_1, i_0)$ to $w(0, i_3, 0, i_1, i_0)$, $0 \leqslant i_1, i_0 < m$, $1 \leqslant i_3 < m$, through the i_3-dimension bus.
4. Copy $w(0, i_3, 0, i_1, i_0)$ to $w(i_4, i_3, 0, i_1, i_0)$, $0 \leqslant i_3, i_1, i_0 < m$, $1 \leqslant i_4 < m$, through the i_4-dimension bus.

Step 2: // Align image and window. //

1. For each $3D\text{-}RAP_{i_4,i_3}$, $0 \leqslant i_4, i_3 < m$, call $RULA(im, i_4, i_3)$.
2. For each $3D\text{-}RAP_{i_4,i_3}$, $0 \leqslant i_4, i_3 < m$, call $RULA(w, i_4, i_3)$.
3. Establish the local connection $\{-S_1, +S_1, -S_0, +S_0\}$ for processor $P_{i_4,i_3,0,i_1,i_0}$, $0 \leqslant i_4, i_3 < m$, $0 \leqslant i_1, i_0 < n$. Copy $w(i_4, i_3, 0, 0, 0)$ to $w(i_4, i_3, 0, i_1, i_0)$, $0 \leqslant i_4, i_3 < m$, $0 \leqslant i_1, i_0 < n$, through the established bus.

Step 3: // Compute the product terms. //

Set $c(i_4, i_3, 0, i_1, i_0)$ to $im(i_4, i_3, 0, i_1, i_0)$ EQ $w(i_4, i_3, 0, i_1, i_0)$, $0 \leqslant i_4, i_3 < m$, $0 \leqslant i_1, i_0 < n$.

Step 4: // Accumulate the sum of products. //

Set $c(0, 0, 0, i_i, i_0)$ to $ANDA(c(i_4, i_3, 0, i_1, i_0))$, $0 \leqslant i_4, i_3 < m$, $0 \leqslant i_1, i_0 < n$.

End TMA.

We show a snapshot of template matching in Figure 3, where $n = 4$ and $m = 3$. We use (im, w) to represent the image and the template respectively in Figure 3(b–e).

THEOREM 1. *The TMA procedure can be computed in $O(1)$ time on a 5D $m \times m \times n \times n \times n$ RAP. Initially, the image and the template are stored in the local variables $im(0, 0, 0, i_1, i_0)$ and $w(0, 0, 0, i_4, i_3)$, respectively, of $2D\text{-}RAP_{0,0,0}$ for all $0 \leqslant i_4, i_3 < m$, $0 \leqslant i_1, i_0 < n$.*

Proof. The correctness of this algorithm directly follows from Equation (1). The time complexity is analyzed as follows. Steps 1 and 3 take $O(1)$ time, respectively. Step 2 takes $O(1)$ time, as shown by Lemma 2. Step 4 takes $O(1)$ time, as shown by Lemma 3. Hence, the time complexity is $O(1)$. QED

Note that when the domain D of the image and the template is quantized into *q-bits* binary sequence, an $O(1)$ or $O(\log^* m)$ time algorithm for template matching can be obtained as the sum of products (Step 4 of TMA) can be computed in constant time as shown by Lemma 6. However, the number of processors is increased to $2qm^2 \times 2qm^2 \times max\{m, n\} \times n \times n$ and $m^2 \times m^2 \times max\{m^2, n\} \times n \times n$, respectively. The sum of products (Step 4 of TMA) can be obtained in $O(\log m)$ time by bus splitting technique [19] for the domain D is integer or real and the time complexity is increased to $O(\log m)$ time but the number of processors is reduced to $m \times m \times n \times n \times n$.

5. CONCLUSIONS

In this paper, we propose a constant time algorithm for template matching using a RAP. Template matching is a computation intensive task. It also requires interprocessor communication. Although the communication is localized to windows of size $m \times m$, it may lead to non-optional performance on the parallel processing system if not carefully implemented [8–10, 12, 24–25]. The architecture of RAP is not only simple and regular but also reconfigurable. Therefore, the communication overhead can be reduced to $O(1)$. Hence, many problems such as image, graph, sorting, geometry and string processing can be solved in constant time by establishing the suitable configuration of the bus system [2, 6, 18–20, 38–41].

Note that the complexity of a switch in a RAP is fully dependent on how many dimensions of a RAP and the bandwidth between processors will be. It is still practical as there are two VLSI implementations that have demonstrated the feasibility and benefits of a 2D RAP, one is the YUPPIE (Yorktown Ultra-Parallel Polymorphic Image Engine) chip [15] and the other is the GCN (Gated-Connection Network) chip [33]. Like the CRCW shared-memory model, there is a gap between the theoretical RAP model and the physical implementations. We agree the connection delay will depend on the problem size so that the constant time for broadcasting delay is not true. However, although it is not true, the broadcasting delay is very small. For example, in a 10^6 processors YUPPIE, only 16 machine cycles are enough for broadcasting. Using the *pre-charged* circuits, the GCN has further reduced the delay. With the advance of VLSI technology, we believe that the gap will be narrowed down and a practical RAP machine will be built in the near future.

REFERENCES

[1] A. Aggarwal, Optimal bounds for finding maximum on array of processors with k global buses. *IEEE Transactions on Computers*, **C-35**, pp. 62–64 (1986).
[2] Y. Ben-Asher, D. Peleg, R. Ramaswami and A. Schuster, The power of reconfiguration. *Journal of Parallel and Distributed Computing*, **13**, pp. 139–153 (1991).
[3] S. H. Bokhari, Finding maximum on an array processor with a global bus. *IEEE Transactions on Computers*, **C-33**, pp. 133–139 (1984).

[4] D. Carlson, *The Mesh with a Global Mesh: A Flexible High Speed Organization for Parallel Computation.* Technical Report, Electrical and Computer Engineering Department, University of Massachusetts (1985).
[5] D. A. Carlson, Performing tree and prefix computations on modified mesh-connected parallel computers. In: *Proceedings of the International Conference on Parallel Processing*, pp. 715–718. IEEE, NY (1985).
[6] D. M. Champion and J. Rothstein, Immediate parallel solution of the longest common subsequence problem. In: *Proceedings of the International Conference on Parallel Processing*, pp. 70–77. IEEE, MD (1984).
[7] C. R. Dyer, A VLSI pyramid machine for hierarchical parallel image processing. In: *Proceedings of the International Conference on Parallel Processing*, pp. 381–386. IEEE, St. Charles (1981).
[8] Z. Fang, X. Li and L. M. Ni, Parallel algorithms for image template matching on hypercube SIMD computers. In: *Proceedings of the IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pp. 33–40. IEEE, NY (1985).
[9] Z. Fang and L. M. Ni, Parallel algorithms for 2D convolutions. In: *Proceedings of the International Conference on Parallel Processing*, pp. 262–269. IEEE, NY (1986).
[10] Z. Fang and L. M. Ni, On the communication complexity of generalized 2-D convolution on array processors. *IEEE Transactions on Computers*, **38**, pp. 183–193 (1989).
[11] S. J. Horng, W. T. Chen and M. Y. Fang, Optimal speed-up algorithms for template matching on SIMD hypercube multiprocessors with restricted local memory. *Information Processing Letters*, **38**, pp. 29–37 (1991).
[12] V. K. P. Kumar and V. Krishnan, Efficient image template matching on hypercube SIMD arrays. In: *Proceedings of the International Conference on Parallel Processing*, pp. 765–771. IEEE, NY (1987).
[13] V. K. P. Kumar and C. S. Raghavendra, Array processor with multiple broadcasting. *Journal of Parallel Distributed Computing*, **4**, pp. 173–190 (1987).
[14] F. T. Leighton, *Parallel Computations using Mesh of Trees.* Technical Report, MIT, Cambridge, MA (1982).
[15] H. Li and M. Maresca, Polymorphic-torus network. *IEEE Transactions on Computers*, **38**, pp. 1345–1351 (1989).
[16] J. R. Melby, *Recognition of Straight Lines by Bus Automata using Parallel Processing.* PhD Dissertation, Ohio State University (1980).
[17] R. Miller, V. K. P. Kumar, D. Reisis and Q. F. Stout, Meshes with reconfigurable buses. In: *Proceedings of the Fifth MIT Conference on Advanced Research in VLSI*, pp. 163–178. MIT, Cambridge MA (1988).
[18] R. Miller, V. K. P. Kumar, D. Reisis and Q. F. Stout, Data movement operations and applications on reconfigurable VLSI arrays. *Proceedings of the International Conference on Parallel Processing*, pp. 205–208. IEEE, NY (1988).
[19] R. Miller, V. K. P. Kumar, D. Reisis and Q. F. Stout, Image computations on reconfigurable VLSI arrays. In: *Proceedings of the IEEE Conputer Society Conference on Computer Vision and Pattern Recognition*, pp. 925–930. IEEE, Washington D.C. (1988).
[20] R. Miller and Q. F. Stout, Efficient parallel convex hull algorithms. *IEEE Transactions on Computers*, **C-37**, pp. 1605–1618 (1988).
[21] R. Miller and Q. F. Stout, Data movement techniques for the pyramid computer. *SIAM Journal on Computing*, **16**, pp. 36–80 (1987).
[22] J. M. Moshell and J. Rothstein, Bus automata and immediate languages. *Information Control*, **40**, pp. 88–121 (1979).
[23] D. Nath, S. N. Maheshwari and P. C. P. Bhatt, Efficient VLSI networks for parallel processing based on orthogonal trees. *IEEE Transactions on Computers*, **C-32**, pp. 569–581 (1983).
[24] X. Qu and X. Li, Parallel template matching algorithms. In: *Proceedings of the International Conference on Parallel Processing*, pp. 223–225. IEEE, NY (1988).
[25] S. Ranka and S. Sahni, Image template matching on SIMD hypercube computers. In: *Proceedings of the International Conference on Parallel Processing*, pp. 84–91. IEEE, NY (1988).
[26] S. Ranka and S. Sahni, Image template matching on MIMD hypercube computers. In: *Proceedings of the International Conference on Parallel Processing*, pp. 92–99. IEEE, NY (1988).
[27] A. Rosenfeld, Parallel image processing using cellular arrays. *Computer*, **16**, pp. 14–20 (1983).
[28] J. Rothstein, On the ultimate limitations of parallel processing. In: *Proceedings of the International Conference on Parallel Processing*, pp. 206–212. IEEE, St. Charles (1976).
[29] J. Rothstein, Toward pattern-recognizing visual prostheses. In: *Proceedings of the IFAC Symposium on Control Aspects Prosthetics Orthotics*, pp. 87–89, Pergamon Press/International Federation of Automatic Control, Oxford (1982).
[30] J. Rothstein, Bus automata, brains, and mental models. *IEEE Transactions on Systems Man, Cybernetics*, **18**, pp. 522–531 (1988).
[31] J. Rothstein and A. Davis, Parallel recognition of parabolic and conic patterns by bus automata. In: *Proceedings of the International Conference on Parallel Processing*, pp. 288–297. IEEE, St. Charles (1979).
[32] L. J. Seigel, H. J. Seigel and A. E. Feather, Parallel processing approaches to image correlation. *IEEE Transactions on Computers*, **C-31**, pp. 208–217 (1982).
[33] D. B. Shu and J. G. Nash, The gated interconnection network for dynamic programming. In: *Concurrent Computations*, S. K. Tewsburg *et al.* (ed.), Plenum, New York (1988).
[34] Q. F. Stout, Mesh connected computers with broadcasting. *IEEE Transactions on Computers*, **C-32**, pp. 826–830 (1983).
[35] Q. F. Stout, Meshes with multiple buses. In: *Proceedings of the 27th IEEE Symposium on the Foundations of Computer Science*, pp. 264–273. IEEE, NY (1986).
[36] S. L. Tanimoto, A pyramidal approach to parallel processing. In: *Proceedings of the 1983 International Symposium on Computer Architecture*, pp. 372–378. Publisher, Location (1983).
[37] L. Uhr, *Algorithm-Structured Computer Arrays and Networks.* Academic Press, New York (1984).
[38] B. F. Wang and G. H. Chen, Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems. *IEEE Transactions on Parallel and Distributed Systems*, **1**, pp. 500–507 (1990).
[39] B. F. Wang and G. H. Chen, Two-dimensional processor array with a reconfigurable bus system is at least as powerful as CRCW model. *Information Processing Letters*, **36**, pp. 31–36 (1990).
[40] B. F. Wang, G. H. Chen and H. Li, Configurational computation: a new computation method on processor arrays with reconfigurable bus system. In: *Proceedings of the International Conference on Parallel Processing*, pp. III-42–III-49. CRC, Boca Raton (1991).
[41] B. F. Wang, G. H. Chen and F. C. Lin, Constant time sorting on a processor array with a reconfigurable bus system. *Information Processing Letters*, **34**, pp. 187–192 (1990).