Theoretical Computer Science 136 (1994) 387–417
Elsevier

A quadtree algorithm for template matching on a pyramid computer

H. Senoussi and A. Saoudi[†]

LIPN, Institut Galilée, Université Paris XIII Av. J.B. Clément, 93430 Villetaneuse, France

Communicated by M. Nivat
Received September 1993
Revised January 1994

Abstract

Senoussi, H. and A. Saoudi, A quadtree algorithm for template matching on a pyramid computer, Theoretical Computer Science 136 (1994) 387–417.

We present an algorithm to perform template matching of an $N \times N$ image with an $M \times M$ template on a $(\log N + 1)$-levels pyramid computer. The time complexity of the algorithm is upper bounded by $\alpha \log N + \beta M^2$, where α and β are constants.

0. Introduction

The template matching is a basic operation in image processing and computer vision. It is used in image registration, scene matching, edge and object detection, filtering, finding lines spots and curves, and image location [15, 1]. Its complexity on a single processor is $O(N^2 M^2)$, where $N \times N$ is the size of the image and $M \times M$ the size of the template. Because of this high complexity, many researchers developed parallel algorithms on various architectures. Since mesh-connected computer (MCC) is suited for this operation, it was one of the most used architectures [3, 5, 9, 10, 12]. Hypercube was well studied too [4, 5, 11, 13, 22] for SIMD, and [14] for MIMD. Some researchers designed algorithms for systolic chips [7, 21]. An algorithm for shuffle-exchange is presented in [5]. At the knowledge of the authors, the pyramid computer was used only once, in [2]. All these algorithms assume that both the image

Correspondence to: H. Senoussi, Institut Galilée, LIPN, Université de Paris XIII, Av. J.B. Clément, 93430 Villetaneuse, France. Email: senou@lipn.univ-paris13.fr.
[†] Deceased 11 August 1993.

and the template are represented by matrix. It is well known that matrices are neither the only data structure for representing images nor the most suited to all the situations. Another very useful data structure is the region quadtree [16]. In this article, which is the full version of our paper [19], our motivation is to develop an efficient algorithm using quadtree as data structure. For this, we design an algorithm to perform template matching of an $N \times N$ image with an $M \times M$ template on a $(\log N+1)$-levels pyramid computer. The time complexity of the algorithm is upper bounded by $\alpha \log N + \beta M^2$, where α and β are constants. In another paper [20], we have presented algorithms to perform template matching on a mesh-connected computer when the image and/or the template are represented by quadtrees.

We suppose that N and M are, respectively, equal to 2^n and 2^m.

This paper is organized as follows: The first section contains some basic definitions. The second section deals with quadtrees. In the third section, we introduce a family of blocks that we will use in the design of our algorithm. In the fourth section, we present the algorithm.

Because of space limitations, this paper will give propositions without proving them. The proofs appear in [18].

1. Basic definitions

In this section, we start by defining the operation that we will study. The parallel computer model for which we will develop our parallel algorithm is described in the second subsection. In the third subsection, we introduce a path in two-dimensional arrays.

1.1. Template matching

Template matching of an image $I[0 \ldots N-1, 0 \ldots N-1]$ with a template $T[0 \ldots M-1, 0 \ldots M-1]$ is the problem to compute the matrix $\mathrm{CONV}[0 \ldots N-1, 0 \ldots N-1]$ defined by

$$\mathrm{CONV}[i,j] = \sum_{u=0}^{M-1} \sum_{v=0}^{M-1} T[u,v] * I[(i+u) \bmod N, (j+v) \bmod N] \ .$$

Intuitively, to obtain $\mathrm{CONV}[i,j]$, we put the template over the image such that the north-western point of T coincides with the point of I whose coordinates are (i,j), than the superimposed values are multiplied together and their products are added.

In the following, "mod P" will be understood every time we will deal with coordinates in a $P \times P$ array.

1.2. Pyramid computer

The machine that we will use is defined as follows:

- It has a control unit (CU) and $\log N+1$ levels of processing elements (PEs).
- Each PE has an index (pos_i, pos_j, pos_l) such that $0 \leqslant pos_l \leqslant \log N$ and $0 \leqslant pos_i$, $pos_j \leqslant 2^{\log N - pos_l} - 1$.
- Each processing element $\mathrm{PE}(pos_i, pos_j, pos_l)$ is connected to $\mathrm{PE}(pos_i+1, pos_j, pos_l)$, $\mathrm{PE}(pos_i-1, pos_j, pos_l)$, $\mathrm{PE}(pos_i, pos_j+1, pos_l)$ and $\mathrm{PE}(pos_i, pos_j-1, pos_l)$. In other words, the PEs of a given level are connected via a mesh interconnection network with wrap-around connections.
- Each processing element $\mathrm{PE}(pos_i, pos_j, pos_l)$ such that $pos_l > 0$ is connected to the four processors $\mathrm{PE}(2*pos_i, 2*pos_j, pos_l-1)$, $\mathrm{PE}(2*pos_i, 2*pos_j+1, pos_l-1)$, $\mathrm{PE}(2*pos_i+1, 2*pos_j, pos_l-1)$, and $\mathrm{PE}(2*pos_i+1, 2*pos_j+1, pos_l-1)$ called its sons.
- Each processor consists of an ALU with registers and a local memory.
- The PEs memory is capable of holding data only. Hence PEs need be able to perform only basic operations. Fetching and decoding operations are performed by the CU.
- The PEs are synchronized to perform the same function at the same time.
- The control unit broadcasts instructions and enable masks. The enable mask is used to select a subset of the PEs to execute the instruction.

Fig. 1 shows the PEs of a 4-level pyramid computer (Each ○ represents a PE, the wrap-around connections are not represented).

In this paper, we will use the following notations:

- Parentheses ( ) are used to index the PEs (e.g. VAL (pos_i, pos_j, pos_l) is the variable VAL of PE (pos_i, pos_j, pos_l)).
- Brackets [ ] are used to index an array (e.g. $\mathrm{CONV}[i,j]$ is the element of the ith row and the jth column of the matrix CONV.)
- The symbol := denotes the intraprocessor assignment.
- The symbol $\rightarrow$ denotes interprocessor assignment.

The main two interprocessor operations that we will use are the following.

```
procedure SHIFT(VAL, $d_i, d_j$)
  begin
    $\mathrm{VAL}(pos_i, pos_j, pos_l) \rightarrow \mathrm{VAL}(pos_i+d_i, pos_j+d_j, pos_l)$.
  end.
```

```
procedure SEND(VAL)
  begin
    $\mathrm{VAL}(pos_i, pos_j, pos_l) \rightarrow \mathrm{VAL}(2*pos_i, 2*pos_j, pos_l-1)$.
    $\mathrm{VAL}(pos_i, pos_j, pos_l) \rightarrow \mathrm{VAL}(2*pos_i, 2*pos_j+1, pos_l-1)$.
    $\mathrm{VAL}(pos_i, pos_j, pos_l) \rightarrow \mathrm{VAL}(2*pos_i+1, 2*pos_j, pos_l-1)$.
    $\mathrm{VAL}(pos_i, pos_j, pos_l) \rightarrow \mathrm{VAL}(2*pos_i+1, 2*pos_j+1, pos_l-1)$.
  end.
```

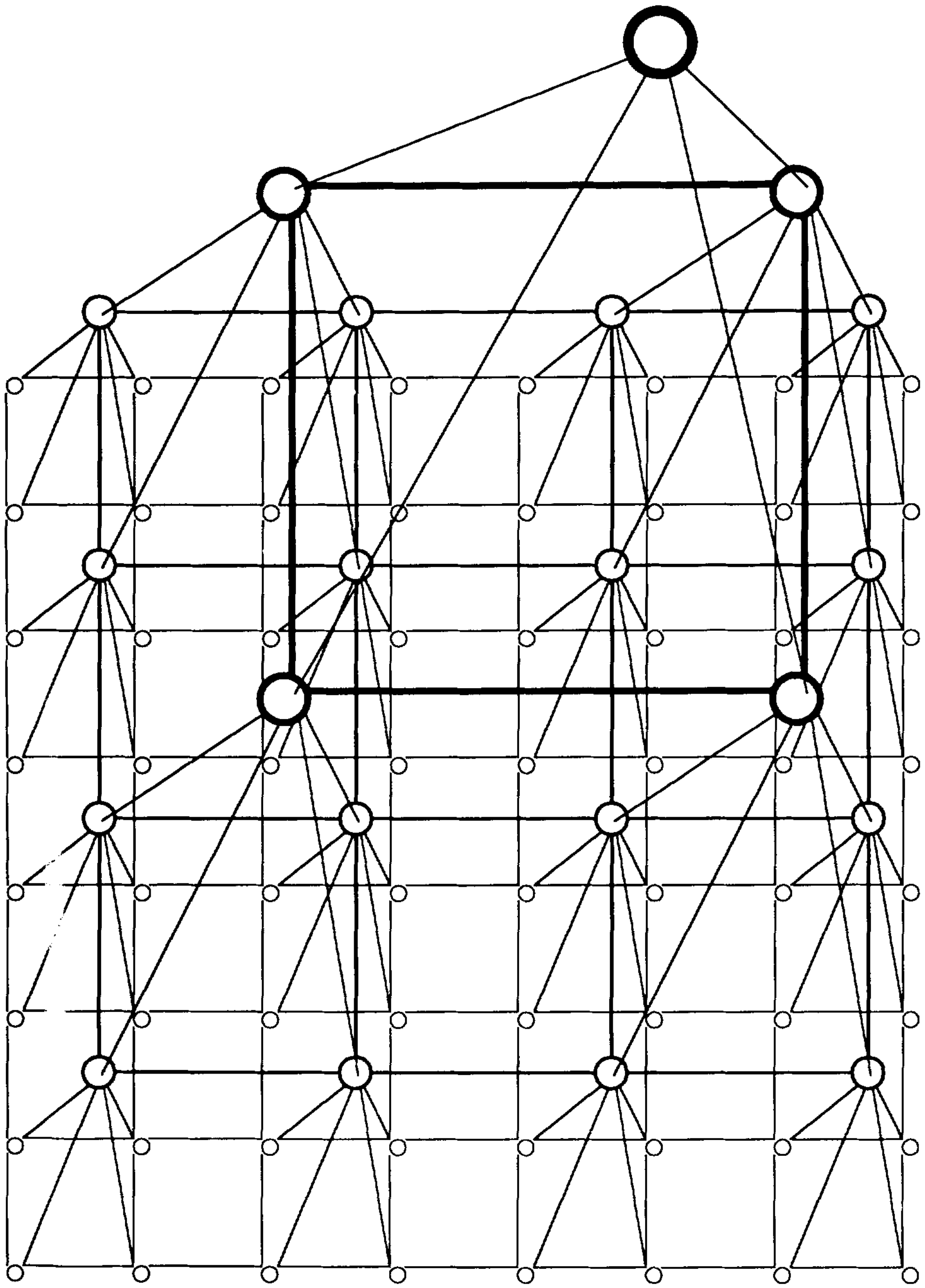


Fig. 1. A 4-level pyramid computer.

1.3. A path in a two dimensional array

Let A be an array of size $2^l \times 2^l$. As we use them here, the coordinates (abs, ord) of an element of A (denoted $A[abs, ord]$) are such that $0 \leqslant abs, ord \leqslant 2^l - 1$. Such an array can be subdivided into four quadrants which are denoted as NW, NE, SW and SE.

Definition 1. For each (abs, ord) such that $0 \leqslant abs, ord \leqslant 2^l - 1$, for each k such that $0 \leqslant k \leqslant l$, $\mathrm{SA}_k(abs, ord)$ is the $2^k \times 2^k$ subarray of A containing (abs, ord), whose north-western element is of coordinates $((abs/2^k) * 2^k, (ord/2^k) * 2^k)$.

A path in A is a list of points $(P_k)_{k=0,\ldots,Nb-1}$ of A, where Nb is the number of points visited. If we denote by (α_k, β_k) the coordinates of P_k, then the length of the path will be equal to

$$\sum_{k=1}^{Nb-1} (|\alpha_k - \alpha_{k-1}| + |\beta_k - \beta_{k-1}|).$$

In the following, we will use a path verifying the following conditions:

(1) It starts at the point of coordinates $(0, 0)$.

(2) It is hamiltonian.

(3) After an element (abs, ord), it visits the points belonging to the subarray $SA_1(abs, ord)$, than the other points of the subarray $SA_2(abs, ord)$, etc.

Such a path is defined as follows using the Freeman notations (Fig. 2): (# is the operator of concatenation): Construct $P(1, \mathrm{NW}, \mathrm{CW})$ or $P(1, \mathrm{NW}, \mathrm{CCW})$ by the following rules:

$k > 1$

$$P(k, \mathrm{NW}, \mathrm{CW}) = P(k-1, \mathrm{NW}, \mathrm{CCW}) \# 0 \# P(k-1, \mathrm{NW}, \mathrm{CW})$$
$$\# 3 \# P(k-1, \mathrm{NW}, \mathrm{CW}) \# 2 \# P(k-1, \mathrm{SE}, \mathrm{CCW}).$$

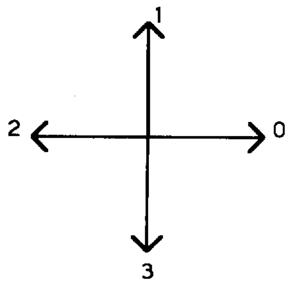


Fig. 2. The Freeman notation.

$$P(k,\mathrm{NW},\mathrm{CCW})=P(k-1,\mathrm{NW},\mathrm{CW})\#3\#P(k-1,\mathrm{NW},\mathrm{CCW})$$
$$\#0\#P(k-1,\mathrm{NW},\mathrm{CCW})\#1\#P(k-1,\mathrm{SE},\mathrm{CW}).$$
$$P(k,\mathrm{NE},\mathrm{CW})=P(k-1,\mathrm{NE},\mathrm{CCW})\#3\#P(k-1,\mathrm{NE},\mathrm{CW})$$
$$\#2\#P(k-1,\mathrm{NE},\mathrm{CW})\#1\#P(k-1,\mathrm{SW},\mathrm{CCW}).$$
$$P(k,\mathrm{NE},\mathrm{CCW})=P(k-1,\mathrm{NE},\mathrm{CW})\#2\#P(k-1,\mathrm{NE},\mathrm{CCW})$$
$$\#3\#P(k-1,\mathrm{NE},\mathrm{CCW})\#0\#P(k-1,\mathrm{SW},\mathrm{CW}).$$
$$P(k,\mathrm{SW},\mathrm{CW})=P(k-1,\mathrm{SW},\mathrm{CCW})\#1\#P(k-1,\mathrm{SW},\mathrm{CW})$$
$$\#0\#P(k-1,\mathrm{SW},\mathrm{CW})\#3\#P(k-1,\mathrm{NE},\mathrm{CCW}).$$
$$P(k,\mathrm{SW},\mathrm{CCW})=P(k-1,\mathrm{SW},\mathrm{CW})\#0\#P(k-1,\mathrm{SW},\mathrm{CCW})$$
$$\#1\#P(k-1,\mathrm{SW},\mathrm{CCW})\#2\#P(k-1,\mathrm{NE},\mathrm{CW}).$$
$$P(k,\mathrm{SE},\mathrm{CW})=P(k-1,\mathrm{SE},\mathrm{CCW})\#2\#P(k-1,\mathrm{SE},\mathrm{CW})$$
$$\#1\#P(k-1,\mathrm{SE},\mathrm{CW})\#0\#P(k-1,\mathrm{NW},\mathrm{CCW}).$$
$$P(k,\mathrm{SE},\mathrm{CCW})=P(k-1,\mathrm{SE},\mathrm{CW})\#1\#P(k-1,\mathrm{SE},\mathrm{CCW})$$
$$\#2\#P(k-1,\mathrm{SE},\mathrm{CCW})\#3\#P(k-1,\mathrm{NW},\mathrm{CW}).$$

and

$$P(1,\mathrm{NW},\mathrm{CW})=032.$$
$$P(1,\mathrm{NW},\mathrm{CCW})=301.$$
$$P(1,\mathrm{NE},\mathrm{CW})=321.$$
$$P(1,\mathrm{NE},\mathrm{CCW})=230.$$
$$P(1,\mathrm{SW},\mathrm{CW})=103.$$
$$P(1,\mathrm{SW},\mathrm{CCW})=012.$$
$$P(1,\mathrm{SE},\mathrm{CW})=210.$$
$$P(1,\mathrm{SE},\mathrm{CCW})=123.$$

$P(k,Q,\mathrm{CW})$ (resp. $P(k,Q,\mathrm{CCW})$) is the path which visits the four quadrants of a $2^k \times 2^k$ array clockwise (resp. counterclockwise) starting by the quadrant Q.

Example. Let us take $l=3$ and find the two paths verifying the conditions (1)–(3).

$P(3,\mathrm{NW},\mathrm{CW})$

$=P(2,\mathrm{NW},\mathrm{CCW})\#0\#P(2,\mathrm{NW},\mathrm{CW})\#3\#P(2,\mathrm{NW},\mathrm{CW})\#2\#P(2,\mathrm{SE},\mathrm{CCW}).$

$=P(1,\mathrm{NW},\mathrm{CW})\#3\#P(1,\mathrm{NW},\mathrm{CCW})\#0\#P(1,\mathrm{NW},\mathrm{CCW})\#1\#210\#0\#$

$P(1,\mathrm{NW},\mathrm{CCW})\#0\#P(1,\mathrm{NW},\mathrm{CW})\#3\#P(1,\mathrm{NW},\mathrm{CW})\#2\#P(1,\mathrm{SE},\mathrm{CCW})\#3$

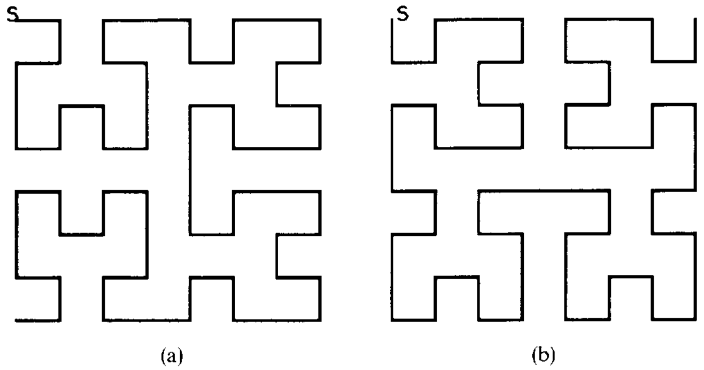


Fig. 3. The two paths $P(3,\mathrm{NW},*)$.

#*P*(1, NW, CCW)#0#*P*(1, NW, CW)#3#*P*(1, NW, CW)#2#*P*(1, SE, CCW)#2
#210#1#*P*(1, SE, CCW)#2#*P*(1, SE, CCW)#3#*P*(1, NW, CW).
=032#3#301#0#301#1#210#0#301#0#032#3#032#2#123#3#301
#0#032#3#032#2#123#2#210#1#123#2#123#3#032.
=032330103011210030100323032212333010032303221232210112321233032.

This path is shown by the Fig. 3(a).

P(3, NW, CCW)
=*P*(2, NW, CW)#3#*P*(2, NW, CCW)#0#*P*(2, NW, CCW)#1#*P*(2, SE, CW).
=*P*(1, NW, CCW)#0#*P*(1, NW, CW)#3#*P*(1, NW, CW)#2#*P*(1, SE, CCW)#3
#*P*(1, NW, CW)#3#*P*(1, NW, CCW)#0#*P*(1, NW, CCW)#1#*P*(1, SE, CW)#0
#*P*(1, NW, CW)#3#*P*(1, NW, CCW)#0#*P*(1, NW, CCW)#1#*P*(1, SE, CW)#1
#*P*(1, SE, CCW)#2#*P*(1, SE, CW)#1#*P*(1, SE, CW)#0#*P*(1, NW, CCW).
=301#0#032#3#032#2#123#3#032#3#301#0#301#1#210#0#032#3
#301#0#301#1#210#1#123#2#210#1#210#0#301.
=301003230322123303233010301121000323301030112101123221012100301.

This path is shown by the Fig. 3(b). *S* is the starting point of the path.
In the following we will use the path $P(*,\mathrm{NW},\mathrm{CW})$ and call it Π_4.

2. Quadtrees

Given a $2^k \times 2^k$ binary image, its quadtree is the tree of degree four which is defined as follows: The root of the tree is associated with the entire image. For each node of the tree, if the region of the image associated with it has only one value (Black or White) than this node is a leaf, else the region is subdivided into four equal-sized quadrants (labeled in order NW, NE, SW and SE) and the node has four sons, each of them being associated with one of the four quadrants.

In the following, we will refer to items in the quadtree (e.g. nodes) and mean their counterparts in the picture (blocks), or vice versa.

The level of a quadtree's node is defined as in [17] by saying that the root has level k and other nodes have a level that is one lower than their father's level.

There are two kinds of quadtree representations: pointer-based and pointer-less. The pointer-less representations are termed *linear quadtree* [17].[1]

The quadtree representation that we use in this work is a variation of the linear quadtree and is defined as follows:

($*$) Since we deal only with binary images, the quadtree can be described by its black leaves only.

($*$) If we know the level lev of a black leaf and the coordinates (a, b) of one of its points, the other points of this leaf are those whose coordinates (abs, ord) verify

$$(a/2^{lev}) * 2^{lev} \leqslant abs \leqslant ((a/2^{lev})+1) * 2^{lev} - 1,$$

$$(b/2^{lev}) * 2^{lev} \leqslant ord \leqslant ((b/2^{lev})+1) * 2^{lev} - 1.$$

The quadtree is then described by a set QD( ) of triples (a, b, lev). Each triple corresponds to a black leaf. The point of coordinates (a, b) is a point of the block called its *representant* and lev is the level of the leaf in the quadtree. In the following, we will use two types of representant:

- The NW-representant of a leaf is its Northwestern point. If the NW-representant is used for all the leaves, we will talk about the *NW-representation* of the quadtree.
- The SE-representant of a leaf is its Southeastern point. If the SE-representant is used for all the leaves, we will talk about the *SE-representation* of the quadtree.

We can associate with a quadtree the following parameters:

- The number of its black leaves, that is the cardinality of the set QD( ). This number will be denoted NL( ).
- The number of its black pixels which is obtained by summing for all the elements of QD( ) the values 2^{2lev}. This number will be denoted S( ).
- The highest (lowest) value of lev in the set $\{(*, *, lev)\}$, that is the highest (lowest) level of the quadtree containing a black leaf. This parameter will be denoted LEV_MAX( ) (LEV_MIN( )).

In the following, we will need to order the black leaves of a quadtree. We will say that QD( ) is ordered by Π_4 when an element (a_1, b_1, lev_1) is used before another element (a_2, b_2, lev_2) if (Π_4 visits (a_1, b_1) before (a_2, b_2)).

Now, we define two transforms on the quadtree of a binary image. The first one is the symmetry and the second is the *projection* onto a level lev.

[1] As Samet note, this is more general than the original definition of Gargantini [6], who termed linear quadtree a sorted list of the black nodes.

2.1. *Symmetrical of a quadtree*

Let us consider the quadtree of a binary image $T(0...2^m-1, 0...2^m-1)$, whose SE-representation is given by the set $\mathrm{QD}(T) = \{(\alpha_i, \beta_i, lev_i)\ 0 \leqslant i \leqslant \mathrm{NL}(T)-1\}$, $\mathrm{NL}(T)$ being the number of black leaves of the quadtree, and let us define the "function" Sym_m as follows.

For (α, β, lev) such that $0 \leqslant \alpha, \beta \leqslant 2^m - 1$ and $0 \leqslant lev \leqslant m$

$$\mathrm{Sym}_m(\alpha, \beta, lev) = (2^m - 1 - \alpha, 2^m - 1 - \beta, lev).$$

Let us call $\mathrm{QS}(T)$ the set $\mathrm{Sym}_m(\mathrm{QD}(T))$. It is easy to see that there exists a $2^m \times 2^m$ binary image T' whose quadtree's NW-representation is given by the set $\mathrm{QD}(T') = \mathrm{QS}(T)$. The quadtree of T' will be called the *symmetrical* of the quadtree of T.

Example. Let us consider the image of Fig. 4(a). Its quadtree is shown by Fig. 5(a) and Fig. 6(a) shows the blocks corresponding to the leaves of this quadtree. The SE-representation of this quadtree is given by the set $\mathrm{QD}(T) = \{(1,1,1), (3,1,0), (0,3,0), (1,3,0), (7,3,2), (1,5,0), (4,5,0), (3,6,0), (1,7,1), (3,7,0), (7,7,1)\}$.

The NW-representation of the symmetrical of this quadtree is given by the set: $\mathrm{QS}(T) = \{(6,6,1), (4,6,0), (7,4,0), (6,4,0), (0,4,2), (6,2,0), (3,2,0), (4,1,0), (6,0,1), (4,0,0), (0,0,1)\}$.

This quadtree is shown by Fig. 5(b), the image corresponding to it by Fig. 4(b) and the blocks corresponding to its leaf nodes by Fig. 6(b).

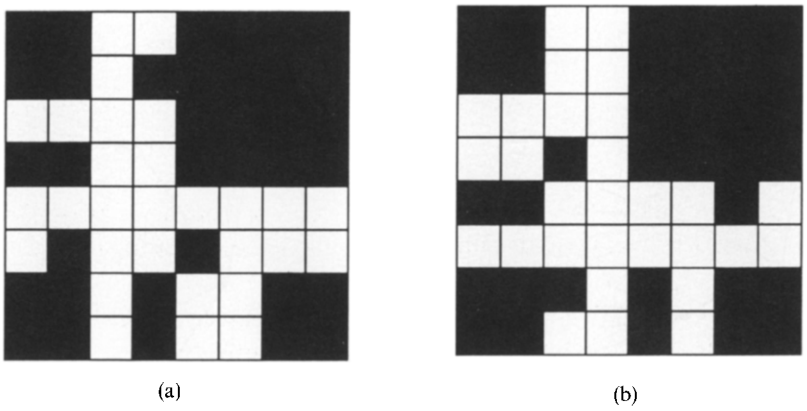


Fig. 4. (a) Image T; (b) image T' corresponding to the symmetrical of the quadtree of T.

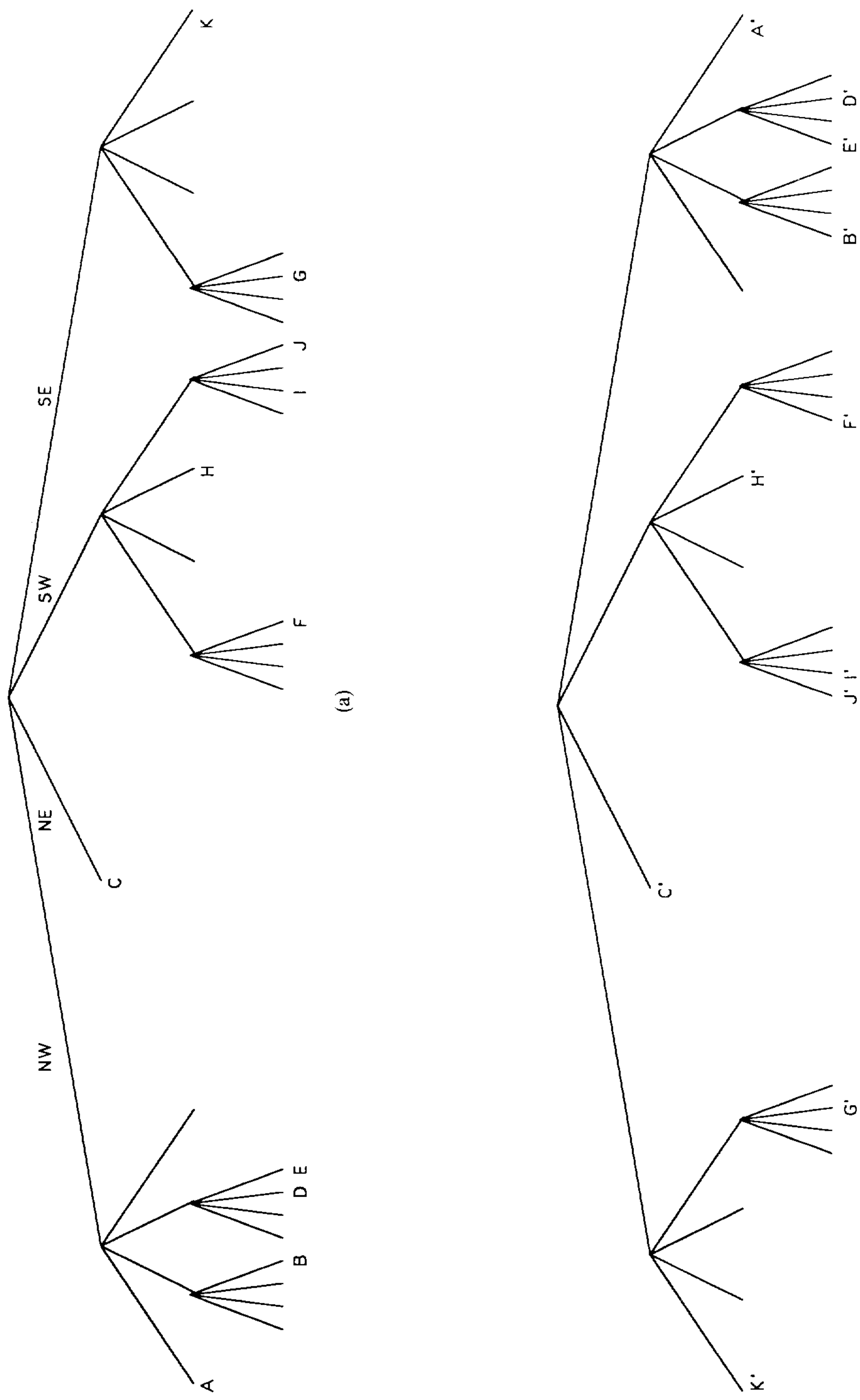


Fig. 5. (a) Quadtree of T; (b) symmetrical of the quadtree of T.

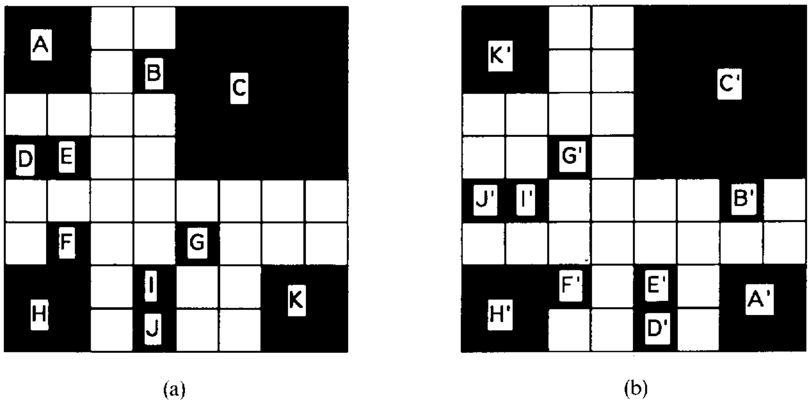


Fig. 6 (a) Blocks of T; (b) Blocks of T'

2.2. *Projection of a quadtree onto a level*

Given a quadtree of a binary image $T[0...2^m-1, 0...2^m-1]$. For L verifying $0 \leqslant L \leqslant m$, we define the following transform.

The *NW-Projection* of the quadtree onto the level L is the graph constituted by the arrays $TP_{lev}[0...2^{m-lev}-1, 0...2^{m-lev}-1]$, $lev=0, ..., L$ defined as follows:

$lev=0, ..., L-1$

$TP_{lev}[a', b'] = lev$ If $(a'2^{lev}, b'2^{lev})$ is a representant of leaf node of the level lev,

$= -1$ else.

and $TP_{lev}[a', b']$ is joined by an edge to $TP_{lev+1}[a'/2, b'/2]$ (Its father).

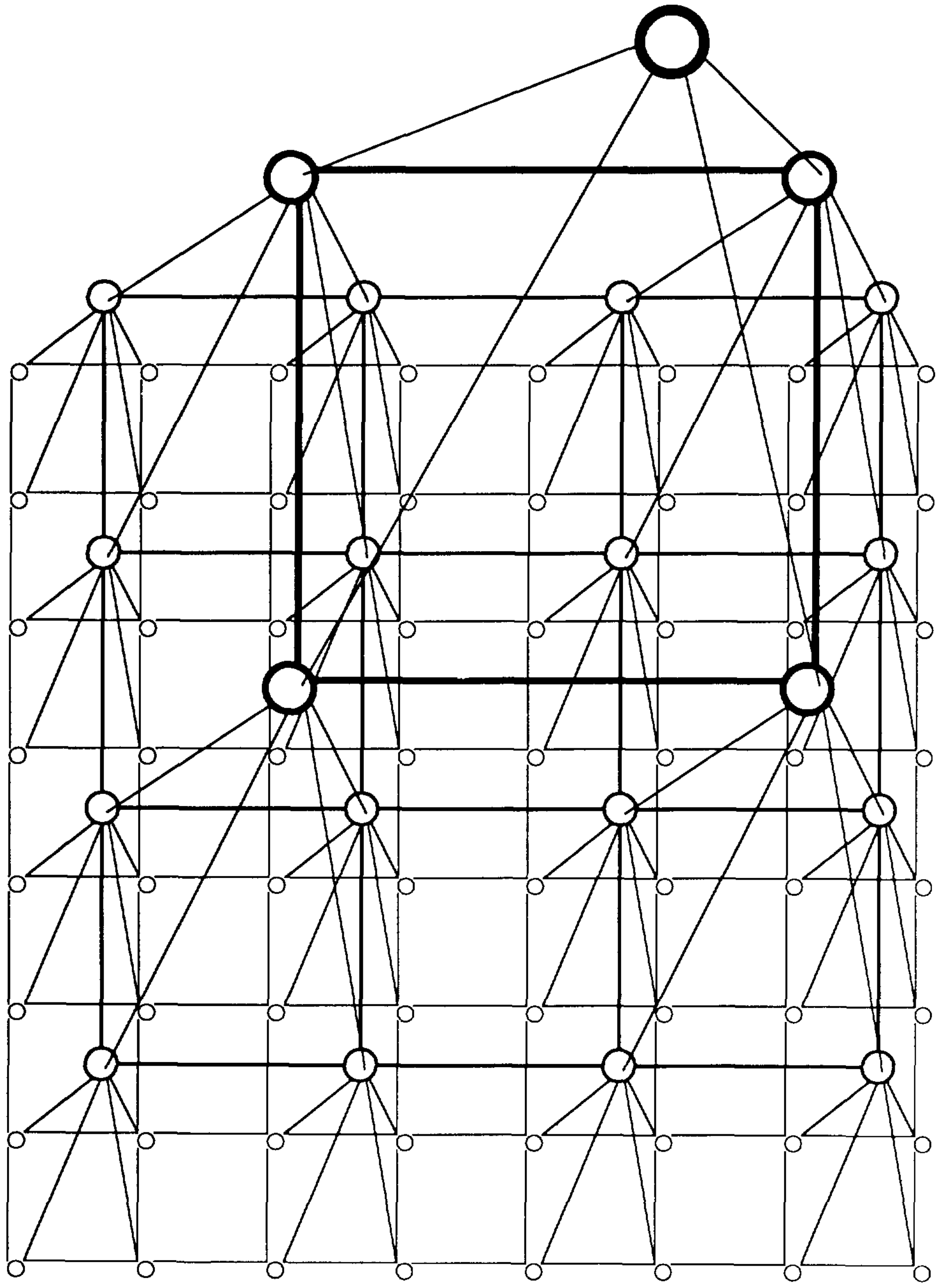
$TP_L[a', b'] = L^+$ if $(a'2^L, b'2^L)$ is the representant NW of a leaf node of the level L^+, $L^+ = L, ..., m$.

$= -1$ else.

Example. Figs. 7 and 8 show, respectively, the projections of an 8×8 image quadtree, and the arrays TP_{lev} of the NW-projection corresponding to the image T of Fig. 4(a):

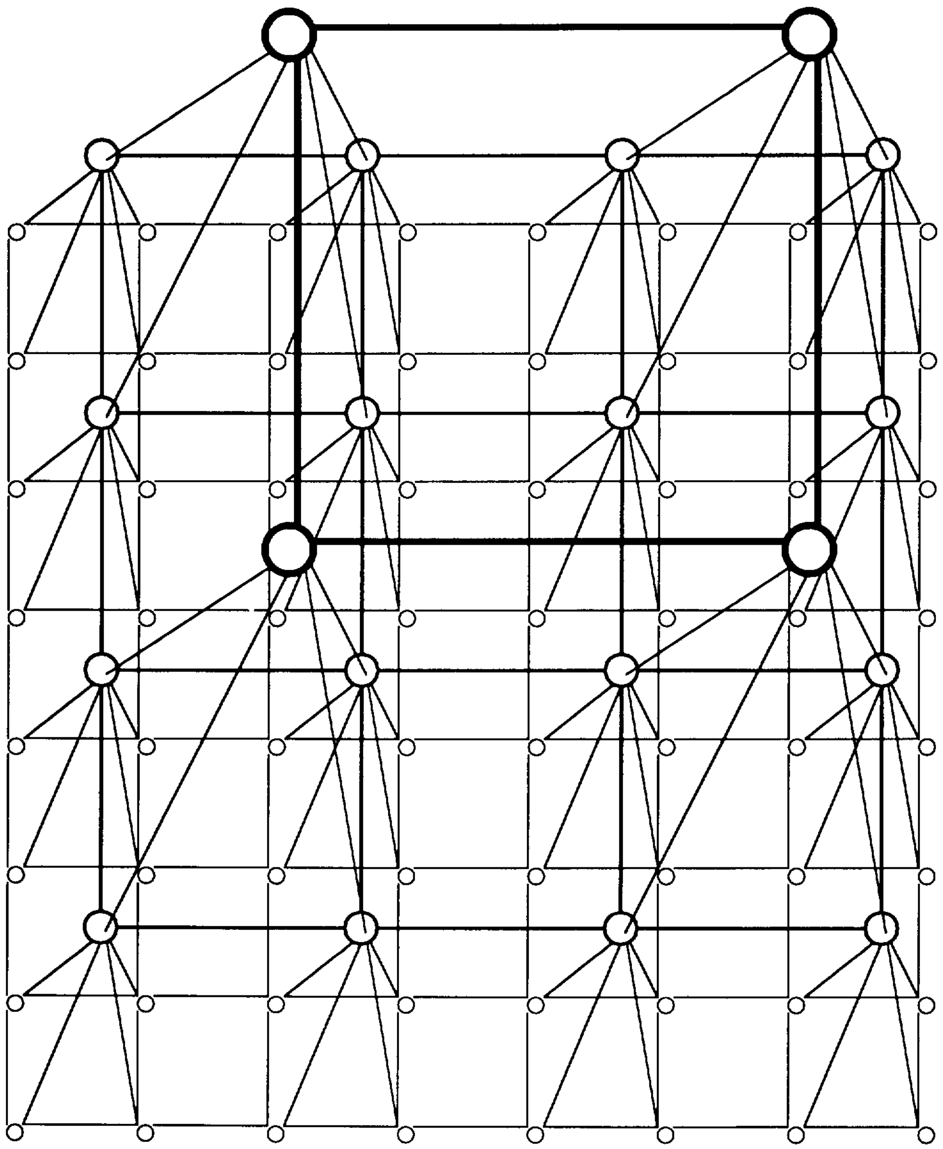
3. Blocks B_*

Given two integers p and q such that $p \geqslant q \geqslant 0$, the block $B_{p,q}$ is the square array of side $2^p + 2^q - 1$ constituted by the subblocks T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8 and T_9 placed as shown by Fig. 9 and defined as follows:



(a)

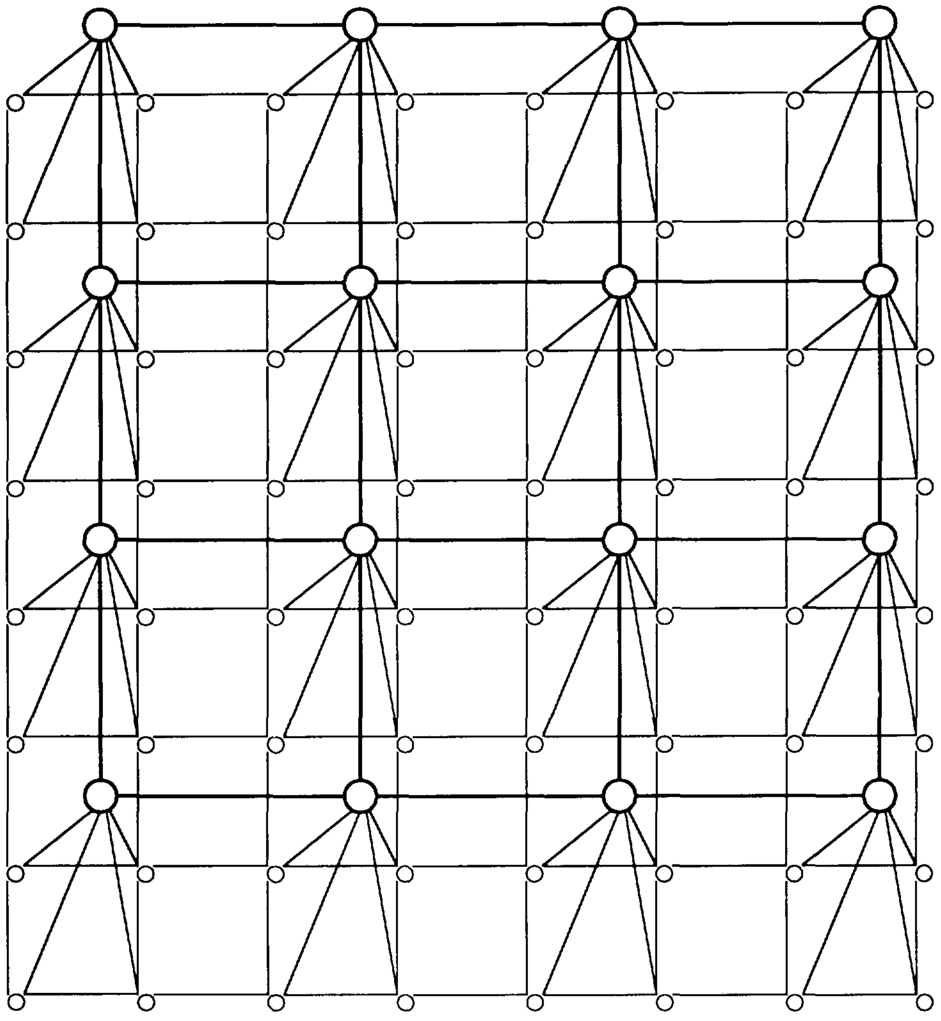
Fig. 7. # Projection onto (a) level 3; (b) level 2; (c) level 1; (d) level 0.



(b)
Fig. 7. (continued)

$T_1[1 \ldots 2^q-1, 1 \ldots 2^q-1]$ is such that $T_1[abs, ord] = abs * ord$,
$T_2[1 \ldots 2^q-1, 1 \ldots 2^p-2^q+1]$ is such that $T_2[abs, ord] = abs * 2^q$,
$T_5[1 \ldots 2^p-2^q+1, 1 \ldots 2^p-2^q+1]$ is such that $T_5[abs, ord] = 2^{2q}$.

In other words, T_1, is the multiplication table of the first 2^q-1 positive integers, the elements of the *abs*th row of T_2 are equal to the product of abs by 2^q and T_5 is the

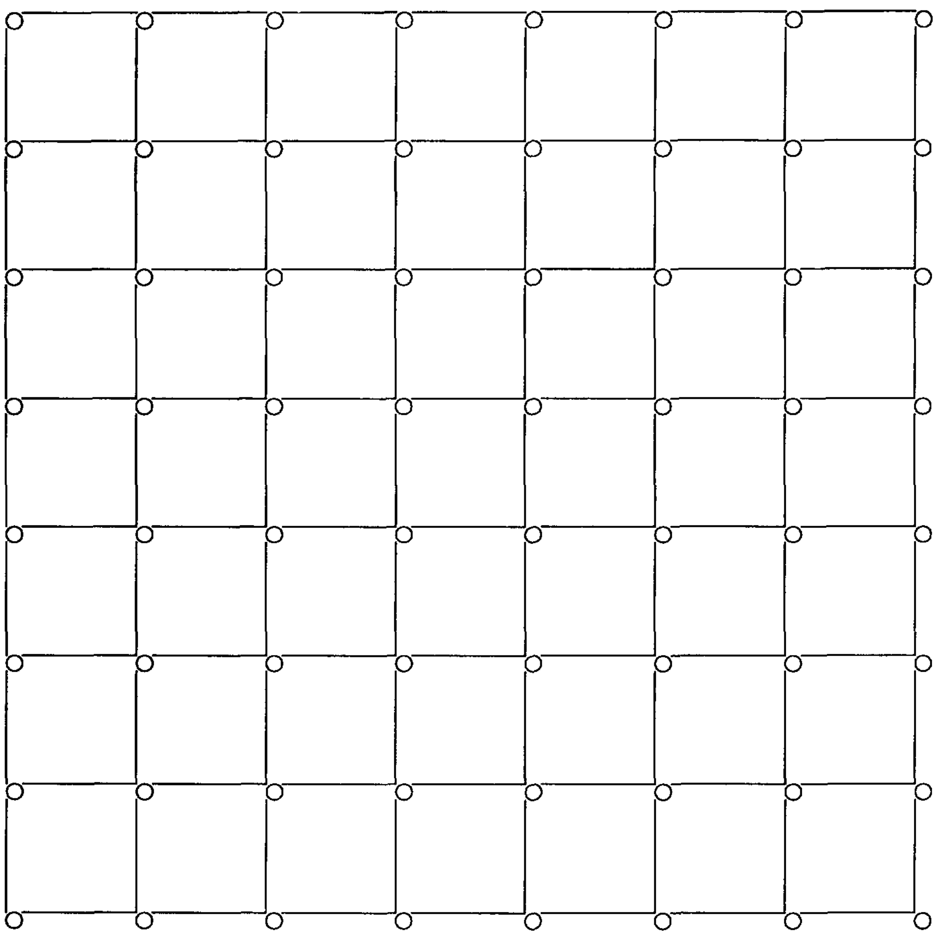


(c)

Fig. 7. (continued)

$(2^p-2^q+1)*(2^p-2^q+1)$ block whose elements are equal to 2^{2q}. T_3, T_7, and T_9 are obtained from T_1, respectively, by a symmetry in relation to the vertical axis a symmetry in relation to the horizontal axis and a compound of the two symmetries. T_6, T_8, and T_4 are obtained from T_2 by, respectively, rotations by $\Pi/2$, Π and $3\Pi/2$. Note that if $q=0$, only T_5 is not empty.

Another definition-building of the blocks $B_{p,q}$ is given by the following algorithm, and illustrated by Fig. 10.



(d)

Fig. 7. (continued)

Algorithm construct_block

```
/*Construct the block $B_{p,q} p \geqslant q \geqslant 0$*/
  (1) Build the first row, that is $B_{p,q}[0, s]$, as follows:
      (1.1) The first and the last elements are equal to 1.
      (1.2) Starting from first (resp. the last) element move to the right (resp. to the left) and give to the current element the value one higher than the previous.
      (1.3) Repeat (1.2) while the current value is less or equal to $2^q$.
      (1.4) Give to the remaining elements the value $2^q$.
  (2) Copy the first row in the first column that is do $B_{p,q}[r, 0] = B_{p,q}[0, r]$.
  (3) Each remaining element $B_{p,q}[r, s]$ of the block is equal to $B_{p,q}[0, s] * B_{p,q}[r, 0]$.
```

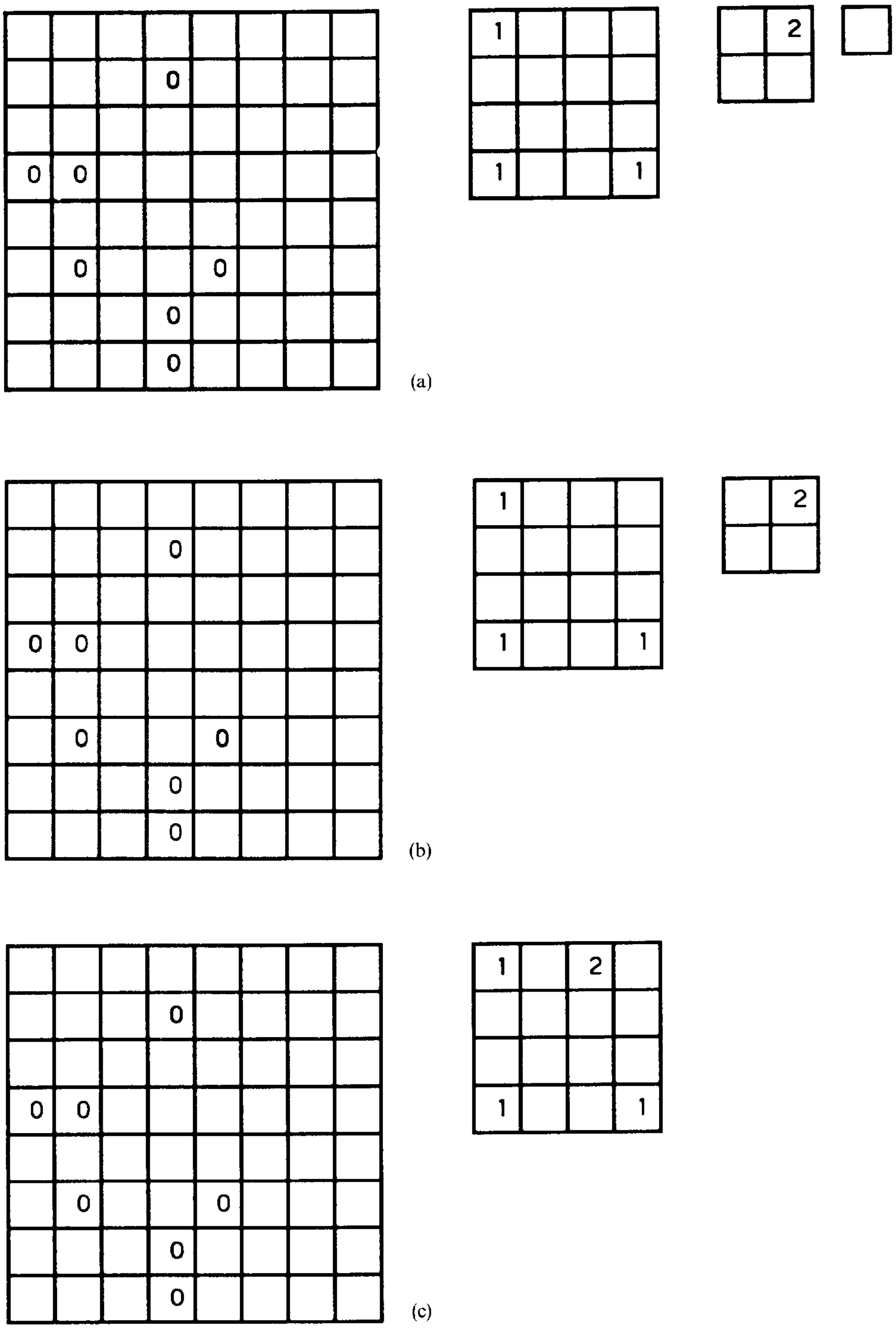
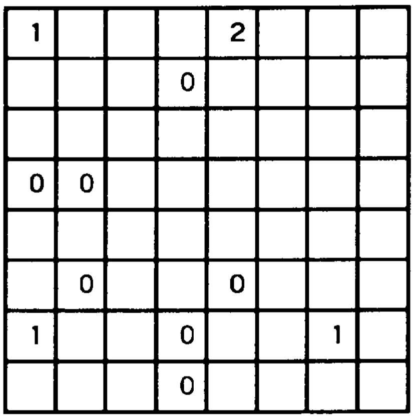


Fig. 8. Arrays of the NW-projections onto (a) level 3; (b) level 2; (c) level 1; (d) level 0.



(d)

Fig. 8. (continued)

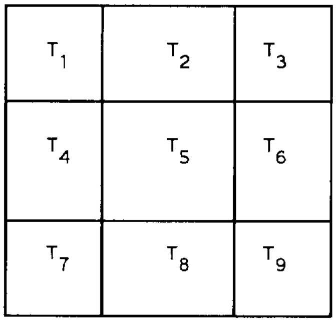


Fig. 9 Subblocks of $B_{*,*}$.

We now give some examples of blocks $B_{*,*}$

Particular case: $q=0$. The block $B_{p,0}$ is of size $2^p * 2^p$ and all its elements are equal to 1. We recognize the blocks used to describe the quadtree of a binary image.

Example 1: $p=2$ *and* $q=1$. The block $B_{2,1}$ shown in Fig. 11.

Example 2: $p=3$ *and* $q=2$. The Block $B_{3,2}$ is shown in Fig. 12

Properties of $B_{*,*}$

We have called $B_{*,*}$ a block rather than an array, because we will not use it as an independent array but as a block of an array. In the following, we will deal with arrays $A[0...N-1, 0...N-1]$ defined as sets of blocks $\{B_i=(B_{p_i,q_i},(x_i,y_i))\}$, where (x_i,y_i) is the position in A of the northwestern point of B_i. In other words, we obtain A as follows:

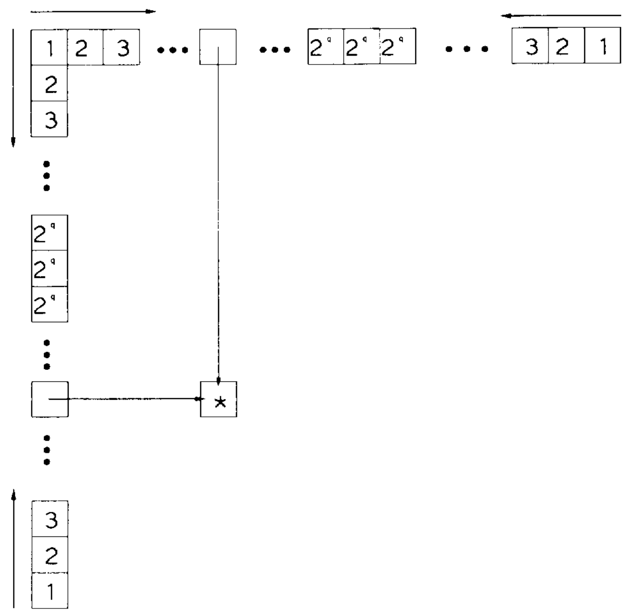


Fig. 10 Building of $B_{*,*}$.

1	2	2	2	1
2	4	4	4	2
2	4	4	4	2
2	4	4	4	2
1	2	2	2	1

Fig. 11

```
**For each** $i$
**begin**
  **For** $r=0, \ldots, 2^{p_i}+2q_i-2$ **Do**
  **begin**
    **For** $s=0, \ldots, 2^{p_i}+2^{q_i}-2$ **Do**
    **begin**
    $A[x_i+r, y_i+s] := A[x_i+r, y_i+s] + B_{p_i,q_i}[r, s]$;
    **end**
  **end**
**end.**
```

1	2	3	4	4	4	4	4	3	2	1
2	4	6	8	8	8	8	8	6	4	2
3	6	9	12	12	12	12	12	9	6	3
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
3	6	9	12	12	12	12	12	9	6	3
2	4	6	8	8	8	8	8	6	4	2
1	2	3	4	4	4	4	4	3	2	1

Fig. 12.

We will say that A has a block B_{p_i,q_i} located at (x_i, y_i), or that (x_i, y_i) is the position of a block B_{p_i,q_i}. Note that two blocks can overlap and have the same position.

Now, let us suppose that we have the NW-representation QD(I) of the quadtree of an image I, the SE-representation QD(T) of the quadtree of a template T and that we want to calculate the template matching CONV of I with T. To obtain the contribution of a couple $(((a,b,lev),(\alpha,\beta,lev')))$ of $\mathrm{QD}(I)\times\mathrm{QD}(T)$ to a given value CONV$[i,j]$, we must proceed as follows:

$(*)$ We put a $2^{lev'}\times 2^{lev'}$ black block over I such that its southeastern point coincides with the point of coordinates $(i+\alpha, j+\beta)$.

$(*)$ The contribution is the surface of the intersection of this block with the black block defined by the triple (a,b,lev).

We have the following properties.

Proposition 1. *The contribution of an element* $(a,b,lev),(\alpha,\beta,lev'))$ *of* $QD(I)\times QD(T)$ (*that is of a couple of black blocks of* $I\times T$) *to the template matching of* I *with* T *is a block* $B_{\mathrm{MAX}(lev,\, lev'),\, \mathrm{MIN}(lev,\, lev')}$ *located at the position* $(a-\alpha, b-\beta)$ *in* $CONV$.

Proposition 2. *A block $B_{p,q}$ such that $p>q\geqslant 0$, located at (x,y) can be decomposed into four blocks $B_{p-1,q}$ located at (x,y), $(x+2^{p-1},y)$, $(x,y+2^{p-1})$, and $(x+2^{p-1},y+2^{p-1})$. This process will be called the decomposition with respect to p.*

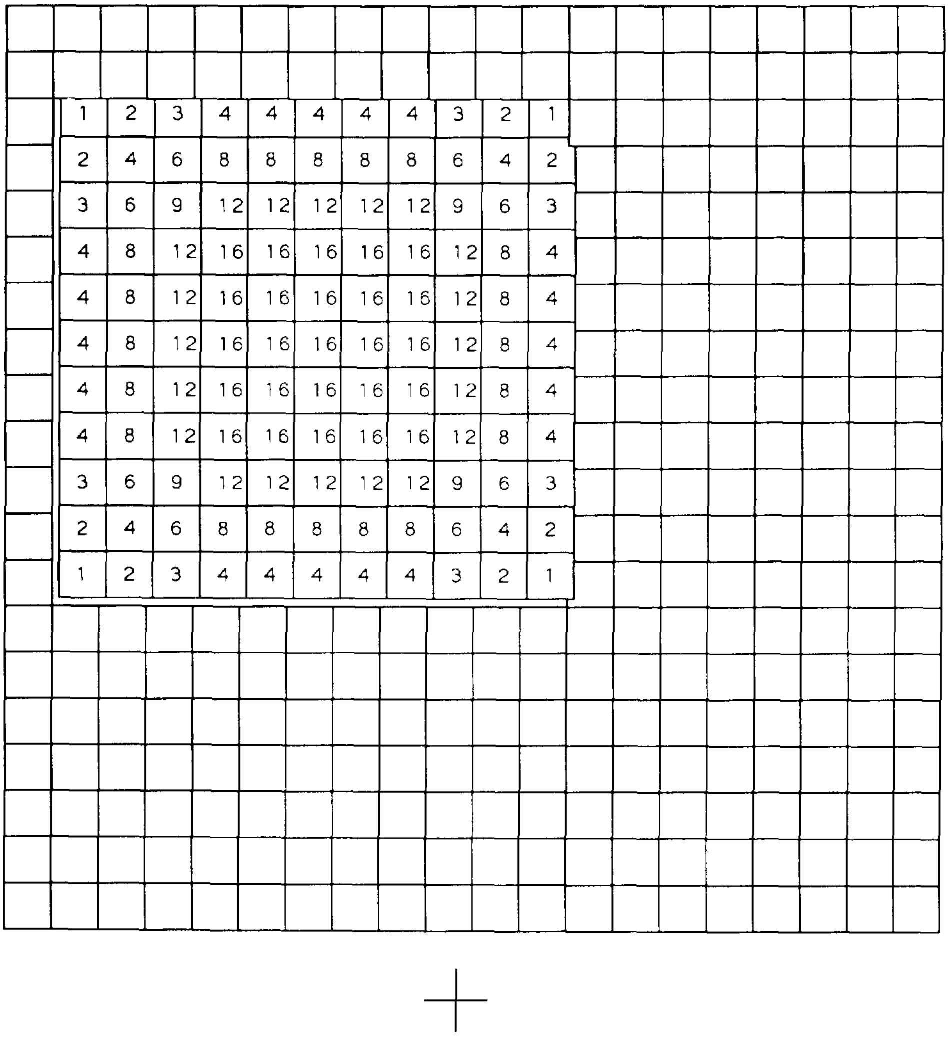
Proposition 3. *A block $B_{p,q}$ such that $p\geqslant q>0$, located at the position (x,y) can be decomposed into four blocks $B_{p,q-1}$ located at (x,y), $(x+2^{q-1},y)$, $(x,y+2^{q-1})$, and $(x+2^{q-1},y+2^{q-1})$. This process will be called the decomposition with respect to q.*

	1	2	3	4	5	6	7	8	8	8	7	6	5	4	3	2	1		
	2	4	6	8	10	12	14	16	16	16	14	12	10	8	6	4	2		
	3	6	9	12	15	18	21	24	24	24	21	18	15	12	9	6	3		
	4	8	12	16	20	24	28	32	32	32	28	24	20	16	12	8	4		
	5	10	15	20	25	30	35	40	40	40	35	30	25	20	15	10	5		
	6	12	18	24	30	36	42	49	49	49	42	36	30	24	18	12	6		
	7	14	21	28	35	42	49	56	56	56	49	42	35	28	21	14	7		
	8	16	24	32	40	48	56	64	64	64	56	48	40	32	24	16	8		
	8	16	24	32	40	48	56	64	64	64	56	48	40	32	24	16	8		
	8	16	24	32	40	48	56	64	64	64	56	48	40	32	24	16	8		
	7	14	21	28	35	42	49	56	56	56	49	42	35	28	21	14	7		
	6	12	18	24	30	36	42	48	48	48	42	36	30	24	18	12	6		
	5	10	15	20	25	30	35	40	40	40	35	30	25	20	15	10	5		
	4	8	12	16	20	24	28	32	32	32	28	24	20	16	12	8	4		
	3	6	9	12	15	18	21	24	24	24	21	18	15	12	9	6	3		
	2	4	6	8	10	12	14	16	16	16	14	12	10	8	6	4	2		
	1	2	3	4	5	6	7	8	8	8	7	6	5	4	3	2	1		

Fig. 13. Decomposition w.r.t. q.

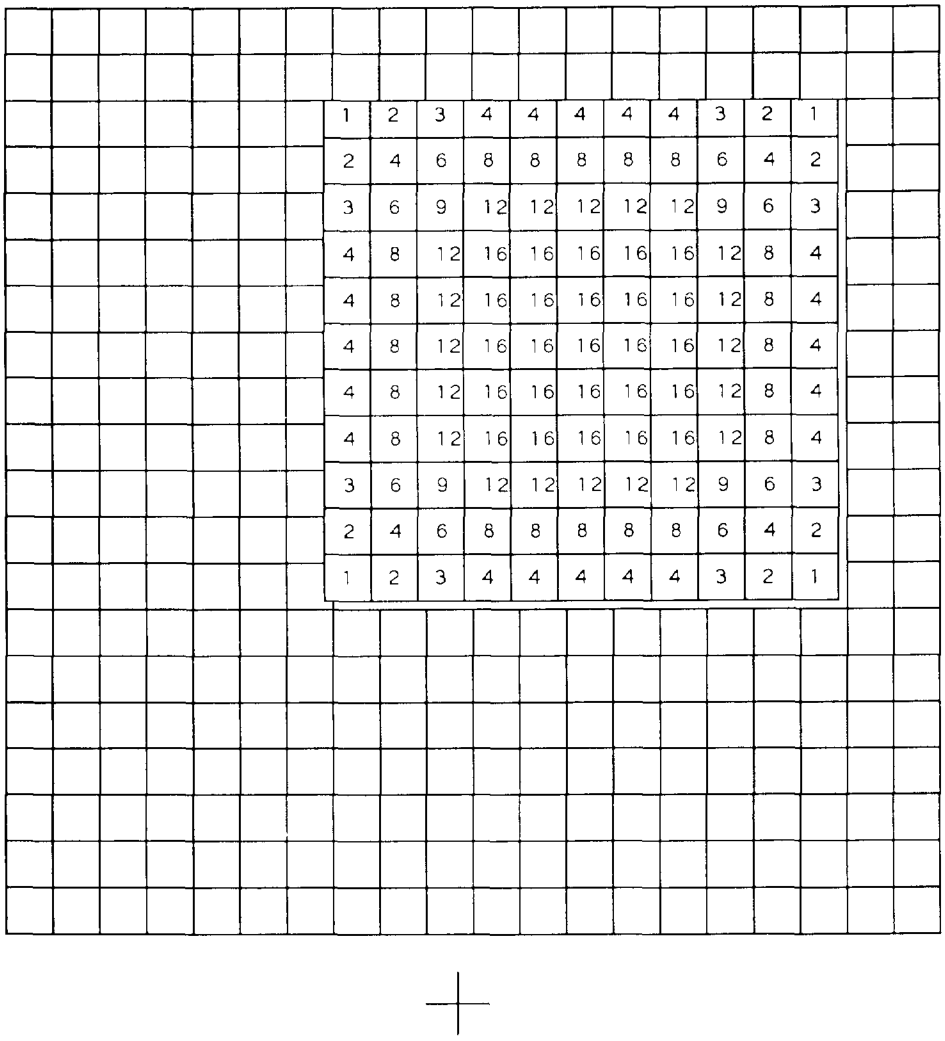
Proposition 4. *Given a block $B_{p,q}$ located at the position (x, y) of an array. The two following operations have the same result: (1) Translate the block by (d_1, d_2) then decompose it w.r.t. p or q and (2) Decompose the block w.r.t. p or q then translate the resulting blocks by (d_1, d_2).*

Example. Now, we give an example of decomposition w.r.t. q. Fig. 13 shows the decomposition of a block $B_{3,3}$ into 4 blocks $B_{3,2}$.



1	2	3	4	4	4	4	4	3	2	1
2	4	6	8	8	8	8	8	6	4	2
3	6	9	12	12	12	12	12	9	6	3
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
3	6	9	12	12	12	12	12	9	6	3
2	4	6	8	8	8	8	8	6	4	2
1	2	3	4	4	4	4	4	3	2	1

Fig. 13. (continued)



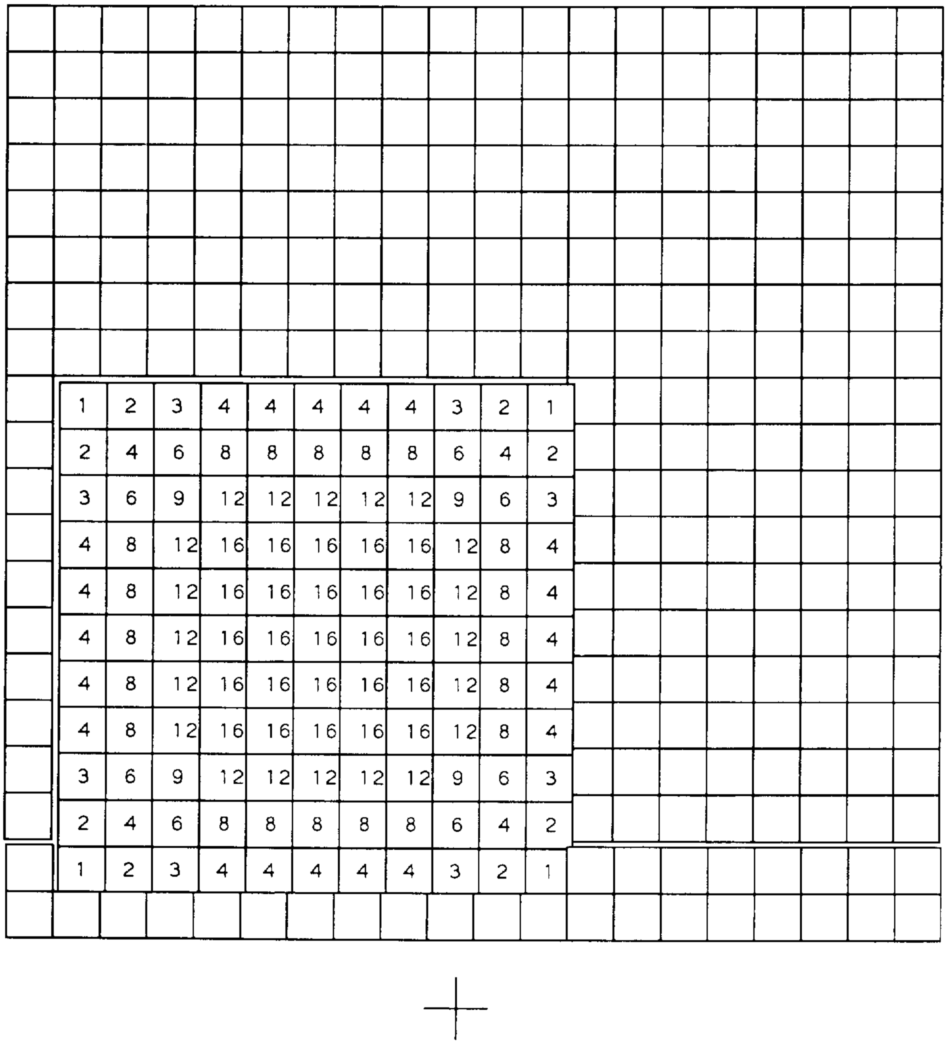
1	2	3	4	4	4	4	4	3	2	1
2	4	6	8	8	8	8	8	6	4	2
3	6	9	12	12	12	12	12	9	6	3
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
3	6	9	12	12	12	12	12	9	6	3
2	4	6	8	8	8	8	8	6	4	2
1	2	3	4	4	4	4	4	3	2	1

Fig. 13. (continued)

4. The algorithm

4.1. Principle

We suppose that we have the NW-representation of the I's quadtree and the SE-representation of the T's quadtree. As we said before (Proposition 1), the contribution of a couple $((a, b, lev), (\alpha., \beta, lev'))$ belonging to $\mathrm{QD}(I) \times QD(T)$ to the template matching of I with T is a block $B_{\mathrm{MAX}(lev,\, lev'),\, \mathrm{MIN}(lev,\, lev')}$ located at $(a-\alpha, b-\beta)$. Let us



1	2	3	4	4	4	4	4	3	2	1
2	4	6	8	8	8	8	8	6	4	2
3	6	9	12	12	12	12	12	9	6	3
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
4	8	12	16	16	16	16	16	12	8	4
3	6	9	12	12	12	12	12	9	6	3
2	4	6	8	8	8	8	8	6	4	2
1	2	3	4	4	4	4	4	3	2	1

Fig. 13. (continued)

transform these coordinates:

$$a-\alpha=a+2^m-1-\alpha-2^m+1,$$

$$b-\beta=b+2^m-1-\beta-2^m+1.$$

That is

$$a-\alpha=a+\alpha_s-2^m+1, \tag{1}$$

$$b-\beta=b+\beta_s-2^m+1, \tag{2}$$

where $(\alpha_s, \beta_s, lev')=\mathrm{Sym}_m((\alpha, \beta, lev'))$.

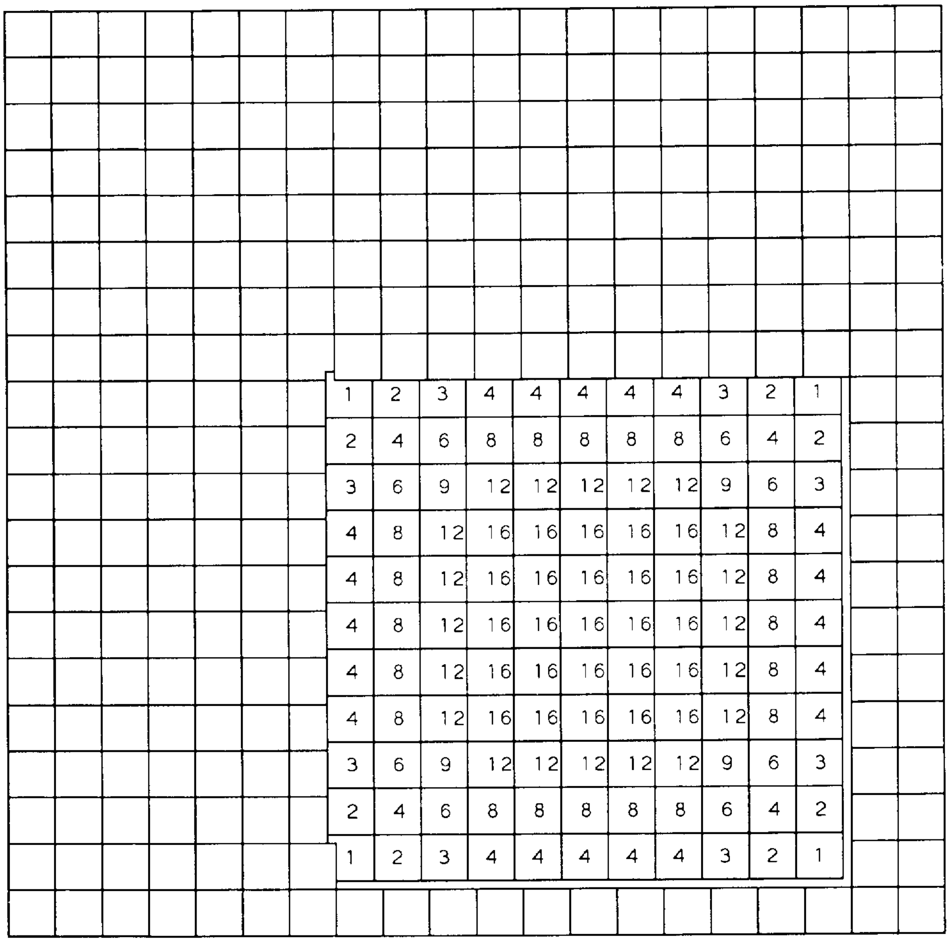


Fig. 13. (continued)

The relations (1) and (2) mean that to obtain the template matching CONV of I with T, we must proceed as follows.

```
**Foreach** element $((a, b, lev), (\alpha_s, \beta_s, lev'))$ of $QD(I) \times QS(T)$
**begin**
    (1) Put a block $B_{\mathrm{MAX}(lev,\, lev'),\, \mathrm{MIN}(lev,\, lev')}$ at $(a+\alpha_s, b+\beta_s)$.
    (2) Shift the block by $-(2^m-1, 2^m-1)$.
    (3) Decompose this block using the Properties 2 and 3.
**end**.
```

Now, let us consider a level lev such that $0 \leqslant lev \leqslant m$ and let us suppose that QD(I) has an element $(a = a'2^{lev}, b = b'2^{lev}, lev)$ We have the following proposition.

Proposition 5. *If we put the NW-projection of the symmetrical of the quadtree of T onto the level lev, such that the NW corner of its highest level (that is lev) coincides with the point (a', b', lev) of the $(n+1)$-levels pyramid, then the point $(\alpha_s, \beta_s, lev')$ of this projection will coincide with the point $((a+\alpha_s)/2^{lev''}, (b+\beta_s)//2^{lev''} = \mathrm{MIN}(lev, lev'))$ of the $(n+1)$-levels pyramid.*

The principle of our algorithm will be the following:

$(*)$ At the beginning, every processor $\mathrm{PE}(pos_i, pos_j, 0)$ contains $\mathrm{CONV}(pos_i, pos_j)$ initialized to 0, and each element (a, b, lev) of $\mathrm{QD}(I)$ is contained by $\mathrm{PE}(a/2^{lev}, b/2^{lev}, lev)$.

First step (Proposition 5)

$(*)$ For $lev \geq m$, for each element (a, b, lev) of $\mathrm{QD}(I)$, for each $(*, *, m)$ descendant of $\mathrm{PE}(a/2^{lev}, b/2^{lev}, lev)$, we put the NW-projection of $\mathrm{QS}(T)$ onto the level m, such that its root coincides with $(*, *, m)$ and then put a block $B_{m, lev'}$ in each point of the pyramid that coincides with a value lev' from the projection of $\mathrm{QS}(T)$.

$(*)$ For $lev \leqslant m$, for each element (a, b, lev) of $\mathrm{QD}(I)$, we put the NW-projection of $\mathrm{QS}(T)$ onto the level lev, such that the NW corner of its highest level (that is lev) coincides with $(a/2^{lev}, b/2^{lev}, lev)$ and then put a block $B_{\mathrm{MAX}(lev, lev'), \mathrm{MIN}(lev, lev')}$ in each point of the pyramid that coincides with a value lev' from the projection of $\mathrm{QS}(T)$.

Second step (Propositions 2 and 3)

$(*)$ Decompose the blocks $B_{*, lev}$ contained by the level lev of the pyramid, such that when they reach the level 0, we obtain blocks $B_{*, 0}$. For this, every processor $\mathrm{PE}(*, *, lev)$ containing a block $B_{*, lev}$ must send a block $B_{*, lev-1}$ to each son of its.

Third step (Proposition 2)

$(*)$ Decompose the blocks $B_{*, 0}$ within the level 0. For this, every processor $\mathrm{PE}(pos_i, pos_j, 0)$ containing a block $B_{p, 0}$ must send a block $B_{p-1, 0}$ to each one of the processors $\mathrm{PE}(pos_i + 2^{p-1}, pos_j, 0)$ $\mathrm{PE}(pos_i, pos_j + 2^{p-1}, 0)$ and $\mathrm{PE}(pos_i + 2^{p-1}, pos_j + 2^{p-1}, 0)$. At the end of this decomposition we have only blocks $B_{0,0}$ that we add to CONV.

Fourth step

$(*)$ Shift CONV by $-(2^m - 1, 2^m - 1)$.

4.2. Description of the algorithm

Every processor $\mathrm{PE}(pos_i, pos_j, lev)$ needs $O(m)$ memory and contains the following variables:

$(*)$ VAL_LI which is equal to 1 if $\mathrm{QD}(I)$ has an element $(pos_i * 2^{lev}, pos_j * 2^{lev}, lev)$, and to 0 if it has not.

(*) QDP is a 4-tuple initialized to (0, 0, –1, –1) and that will be updated as it will be explained in the following.

(*) STATE which is equal to BEFORE if the level has not received the first element of QS(T) yet, AFTER if it has already received the last one and DURING otherwise.

(*) FT_PRC and FT_PRS which depend on the present and the previous elements of QS(T) in the processor.

(*) The processors PE(pos_i, pos_j,0) have arrays $Tab[0...m]$ which are defined by Tab[i] = number of blocks $B_{i,0}$ whose position is (pos_i, pos_j).

The elementary operations used by the algorithm are the following:

(*) PE( ) transmits QDP to its four sons, this is performed by the procedure SEND ( ) defined above.

(*) PE ( ) updates QDP.

(*) PE(pos_i, pos_j, lev) sends VAL_LI to PE($pos_i + x_{lev}, pos_j + y_{lev}, lev$), where $0 < x_{lev}, y_{lev} < 2^{n-lev}$ are the same for all the processors of the level. This is performed by the procedure SHIFT( ). defined above.

(*) PE(pos_i, pos_j, 0) sends a data Tab [ ] to PE($pos_i + 2^q, pos_j$, 0), PE($pos_i, pos_j + 2^q$, 0) and PE($pos_i + 2^q, pos_j + 2^q$, 0) where $0 < q < m$ is the same for the Tprocessors of the level 0. This is performed by the SHIFT( ).

The algorithm using these operations to achieve the four steps described above is as follows:

First and second steps

(*) The elements (α_s, β_s, lev') of QS(T) are ordered by the path Π_4. They are introduced at the apex in the form QDP = ($\alpha_s, \beta_s, lev', -1$), and each processor receiving this 4-tuple transmits it to its sons, after having updated it if necessary.

(*) To describe the different updating of QDP, we must envisage 2 cases:

(**) If $lev \geqslant m$, each processor PE(pos_i, pos_j, lev) such that VAL_LI = 1 updates QDP to ($\alpha_s, \beta_s, lev', lev$) (that is creates a block $B_{lev,lev'}$). Then, for each son PE(pos_{i1}, pos_{j1}, m) of PE(pos_i, pos_j, lev) we must send a block $B_{m,lev'}$ to PE($pos_{i1} * 2^m + \alpha_s)/2^{lev'}$, ($pos_{j1} * 2^m + (\beta_s)/2^{lev'}, lev'$) which can be done as follows:

(***) Between the levels $lev - 1$ and m, each processor PE($pos_{i2}, pos_{j2}, lev_1$) that receives the 4-tuple (* , * , $lev', lev_1 + 1$) updates it to (*, *, lev', lev_1).

(***) Between the levels m and lev' the block must be transmitted without modification until it reaches its destination, that is the processor PE(pos_{i1}, * $2^m + \alpha_s)/2^{lev'}$, (pos_{j1}, * $2^m + \beta_s)/2^{lev'}$ lev') The path going from PE(pos_{i1}, pos_{j1}, m) to PE(($pos_{i1} * 2^m + \alpha_s)/2^{lev'}$, ($pos_{j1} * 2^m + \beta_s)/2^{lev'}, lev'$) traverses only one processor in each level lev_1 such that $lev' \leqslant lev_1 \leqslant m$, this processor is the only predecessor of PE(($pos_{i1} * 2^m + \alpha_s)/2^{lev'}$, ($pos_{j1}$, * $2^m + \beta_s)/2^{lev'}, lev'$) belonging the level lev_1 (PE(($pos_{i1} * 2^m + \alpha_s)/2^{lev'}$, ($pos_{j1}$, * $2^m + \beta_s)/2^{lev'}$, lev') itself if $lev_1 = lev'$). Hence the transmission of QDP is done as follows: each processor PE(*, , lev_1) which receives a 4-tuple (* , * ,lev', m) such that $lev' \leqslant lev_1$ does not modifyit if it is a predecessor of PE(($pos_{i1} * 2^m + \alpha_s)/2^{lev'}, pos_{j1} * 2^m + \beta_s)/2^{lev'}, lev'$) and updates it to (* , * , lev', –1) else. PE($pos_{i2}, pos_{j2}, lev_1$) is a predecessor of ($pos_{i1} * 2^m + \alpha_s)/2^{lev'}$,

$(pos_{j1} * 2^m + \beta_s)/2^{lev'})$ if $pos_{i1} * 2^m + \alpha_s$ (resp. $pos_{j1} * 2^m + \beta_s$) has the same $(n - lev_1)$ high bits as $pos_{i2} * 2^{lev_1}$ (resp. $pos_{j2} * 2^{lev_1}$). Since these two processors are successors of the same processor of the level m, we know that $pos_{i1} * 2^m + \alpha_s$ (resp. $pos_{j1} * 2^m + \beta_s$) has the same $(n - m)$ high bits as $pos_{i2} * 2^{lev_1}$ (resp. $pos_{j2} * 2^{lev_1}$), hence we need to compare only the bits between the positions $m - 1$ and lev_1. By induction, it is easy to prove that $\mathrm{PE}(pos_{i2}, pos_{i2}, lev_1)$ needs to compare the bits of the position lev_1 only. The bit of the position lev_1 in $pos_{i1} * 2^m + \alpha_s$ (resp. $pos_{j1} * 2^m + \beta_s$) is the same as that of α_s (resp. β_s) because $pos_{i1} * 2^m$ (resp. $pos_{j1} * 2^m$) is a multiple of 2^m. We conclude that: When a processor $\mathrm{PE}(pos_{i2}, pos_{j2}, lev_1)$ such that $0 \leqslant lev_1 < m$ receives a 4-tuple $(\alpha_s, \beta_s, lev', m)$ such that $lev' \leqslant lev_1$, it compares the bits of the position lev_1 of $pos_{i2} * 2^{lev_1}$ and α_s (resp. $pos_{j2} * 2^{lev_1}$ and β_s). If equality, it transmits QDP to its sons without modification, else updates it to $(*, *, lev', -1)$.

(***) Starting from the level lev' the block $B_{m,lev'}$ must be decomposed into blocks $B_{m,lev'-1}, B_{m,lev'-2}, \ldots, B_{m,0}$. This can be done as follows: If a processor $\mathrm{PE}(*, *, lev_2)$ receives a 4-tuple $(*, *, lev_2 + 1, m)$ it updates it to $(*, *, lev_2, m)$.

(**) If $lev < m$, the destination of the block is $\mathrm{PE}((pos_i * 2^{lev} + \alpha_s)/2^{lev''}, (pos_j * 2^{lev} + \beta_s)/2^{lev''}, lev'' = \mathrm{MIN}(lev, lev'))$. We can write

$$\begin{aligned} pos_i * 2^{lev} + \alpha_s &= pos_i * 2^{lev} + (\alpha_s/2^{lev})2^{lev} + (\alpha_s \bmod 2^{lev}) \\ &= (pos_i + \alpha_{s1}) * 2^{lev} + \alpha_{s2}, \\ pos_j * 2^{lev} + \beta_s &= pos_j * 2^{lev} + (\beta_s/2^{lev})2^{lev} + (\beta_s \bmod 2^{lev}) \\ &= (pos_j + \beta_{s1}) * 2^{lev} + \beta_{s2}. \end{aligned}$$

The creation of the block $B_{\mathrm{MAX}(lev,\, lev'),\, \mathrm{MIN}(lev,\, lev')}$ (that is first updating of QDP) will be done by the processor $\mathrm{PE}(pos_{i1} = pos_i + \alpha_{s1}, pos_{j1} = pos_j + \beta_{s1}, lev)$. Hence, we need first to do a shift within the level lev on VAL_LI by $(\alpha_{s1}, \beta_{s1})$, then if VAL_LI $= 1$ in a processor of this level, this processor will create a block $B_{\mathrm{MAX}(lev,\, lev'),\, \mathrm{MIN}(lev,\, lev')}$, that is it updates QDP to $(\alpha_s, \beta_s, lev', lev)$.

(***) After that, this block must be sent to $\mathrm{PE}((pos_{i1} * 2^{lev} + \alpha_{s2})/2^{lev''}$, $(pos_{j1} * 2^{lev} + \beta_{s2})/2^{lev''}, lev'')$ without any modification. For this, every processor $\mathrm{PE}(pos_{i2}, pos_{j2}, lev_1)$ receiving this 4-tuple will decide if it is destined for it, that is if this processor $\mathrm{PE}(pos_{i2}, pos_{j2}, lev_1)$ is a predecessor of $\mathrm{PE}((pos_{i1} * 2^{lev} + \alpha_{s2})/2^{lev''}$, $(pos_{j1} * 2^{lev} + \beta_{s2})/2^{lev''}, lev'')$ as follows: it compares the bits of the position lev_1 of $pos_{i2} * 2^{lev_1}$ (resp. $pos_{j2} * 2^{lev_1}$) and α_{s2} (resp. β_{s2}). Given the relation between $(\alpha_{s2}, \beta_{s2})$ and (α_s, β_s), this is the same thing as comparing the bits of the position lev_1 of $pos_{i2} * 2^{lev_1}$ (resp. $pos_{j2} * 2^{lev_1}$) and α_s (resp. β_s). If equality, the processor concludes that the block is destined for it, else it is not and the processor updates the 4-tuple to $(\alpha_s, \beta_s, lev', -1)$. This must be done while not reaching the level $lev'' - 1$.

(***) Starting from the level lev'' the block $B_{*,lev''}$ must be decomposed into blocks $B_{*,lev''-1}, B_{*,lev''-2}, \ldots, B_{*,0}$. This can be done as follows: If a processor $\mathrm{PE}(*, *, lev_2)$ receives a 4-tuple $(*, *, lev', lev_2 + 1)$, it updates it to $(*, *, lev', lev_2)$, else if it receives a 4-tuple $(*, *, lev_2 + 1, lev)$ it updates it to $(*, *, lev_2, lev)$.

Third and fourth step

See above.

Let us recapitulate. The algorithm is the following.

```
Algorithm TM_Pyr
begin
/*Initialize the variables*/
(*) FT_PRC := (0,0).
(*) FT_PRS := (0,0).
(*) STATE := BEFORE.
(*) QDP := (0,0,−1,−1).
/*First step*/
While (STATE $\neq$ AFTER)
  begin
  (*) SEND(QDP);
  (*) Receive $(\alpha_s, \beta_s, q, p)$;
  (*) QDP := $(\alpha_s, \beta_s, q, p)$;
  (*) If ((STATE = BEFORE) and (QDP.$q \neq -1$)) then STATE:= DURING;
  else If ((STATE = DURING) and (QDP.$q = -1$)) then STATE:= AFTER;
  (*) if (STATE = DURING)
  begin
  (**) If (QDP.$p \neq -1$) /*Receives a block from its father*/
    begin
    If ($lev \geqslant m$) QDP.$p$:= $lev$;
    else
      begin
      If (MIN(QDP.$q$, QDP.$p$) = $lev+1$) MIN (QDP.$q$, $QDP.p$):= $lev$;
      else /*(MIN(QDP.$q$, QDP.$p$) < $lev+1$)*/
        begin
        If ((the bits of the position $lev$ of $\alpha_s$, and $pos_i * 2^{lev}$)
        or (those of $\beta_s$ and $pos_j * 2^{lev}$) are different) then QDP.$p$:= −1;
        end;
      end;
      end;
    (**) FT_PRC:= FT_PRS;
    (**) FT_PRS:= $(\alpha_s/2^{lev}, \beta_s/2^{lev})$;
    (**) $(d_{i,lev}, d_{j,lev})$:= FT_PRS − FT_PRC;
    (**) For $lev = m-1, \ldots, 0$ SHIFT($d_{i,lev}, d_{j,lev}$, VAL_LI);
    (**) If (VAL_LI = 1) then QDP.$p$:= $lev$; /* Creation of a block*/
    end;
    (*) If ($lev = 0$)
    begin
    If   (QDP.$p \neq -1$)   then   $Tab$[MAX(QDP.$q$, QDP.$p$)]:= $Tab$[MAX(QDP.$q$,
QDP.p)] + 1;
    else If (STATE = AFTER) then Start the second step;
    end;
  end;
/*Second step*/
/*For the processors of the level 0 only*/
For $ord = m, \dots, 1$ Do
  begin
  $val := Tab[ord]$;
  SHIFT ($2^{ord-1}, 0, val$);
  $Tab[ord-1] := Tab[ord-1] + val$;
  SHIFT ($0, 2^{ord-1}, val$);
  $Tab[ord-1] := Tab[ord-1] + val$;
  SHIFT ($-2^{ord-1}, 0, val$);
  $Tab[ord-1] := Tab[ord-1] + val$;
  end;
CONV := $Tab[0]$;
/*Third step*/
/*For the processors of the level 0 only*/
  (*) SHIFT ($-(2^m-1), -(2^m-1)$, CONV);
  end.
/*PE($pos_i, pos_j$, 0) contains CONV[$pos_i, pos_j$] in $Tab[0]$*/
```

Example. As an illustration of this algorithm, let us suppose that $n=10$, $m=6$ and that QD(I) has an element (0, 0, 5) and QD(T) an element (23, 15, 3). The contribution of this couple of elements to the template matching is a block $B_{5,3}$ located at (0-23, 0-15) = (1001, 1009).

This contribution is computed by the algorithm as follows:

- The element 64-1-23, 64-1-15, 3) = (40, 48, 3) of QS(T) is entered at the apex (level 10 of the pyramid) in the form QDP = (40, 48, 3, −1).
- Between the levels 10 and 6 the predecessors of PE(0, 0, 5) do not modify this 4-tuple.
- Since $40 = 1 * 2^5 + (40 \bmod 2^5)$ and $48 = 1 * 2^5 + (48 \bmod 2^5)$, when the level 5 of the pyramid receives the 4-tuple (40, 48, 3, −1) it shifts the values of VAL_I( ) by (1, 1).
- After this shift, PE(1, 1, 5) has VAL_I = 1 (from PE(0, 0, 5)).
- PE(1, 1, 5) creates a block $B_{5,3}$ that it updates QDP to (40, 48, 3, 5).
- This value of QDP is destined for PE($(0+40)/2^3, (0+48)/2^3$, MIN(5, 3)) = PE(5, 6, 3). It must reach this processor – and only this processor – without modification.
- PE(1, 1, 5) transmits QDP to PE(2, 2, 4), PE(2, 3, 4), PE(3, 2, 4), and PE(3, 3, 4) (its sons). These four processors PE(pos_i, pos_j, 4) compare the bits of the position 4 of $pos_i * 2^4$ (resp. $pos_j * 2^4$) and $\alpha_s = 40 = 101000$ (resp. $\beta_s = 48 = 110000$).

- Only PE(2, 3, 4) has equality. It does not modify QDP. The others update it to (40, 48, 3, −1).
- PE(2, 3, 4) transmits QDP to PE(4, 6, 3), PE(4, 7, 3), PE(5, 6, 3) and PE(5, 7, 3).
- After comparison of bits, all processors but PE(5, 6, 3) update QDP to (40, 48, 3, −1).
- Decomposition of the block $B_{5,3}$ starts: Each processor PE($*$, $*$, *lev*) descendant of PE(5, 6, 3) receives QDP = (40, 48, *lev* + 1, 5) and updates it to (40, 48, *lev*, 5).
- At the end of this decomposition, each descendant PE($*$, $*$,0) of PE(5, 6, 3) has a block $B_{5,0}$. These processors cover the area whose northwestern point is of coordinates $(5*2^3, 6*2^3)=(40,48)$.
- Now, these blocks need to be decomposed w.r.t. *p*.
- Finally, the result of the decomposition is shifted by (−63, −63). After this shift, the contribution of the element ((0, 0, 5), (23, 15, 3)) of QD(I) × QD(T) cover an area whose northwestern point is of coordinates (40-63, 48-63) = (1001, 1009).

4.3. Complexity

(1) The cost of vertical transmissions is $\log N + \mathrm{NL}(T)$.

(2) Since the elements of QS(T) are ordered by Π_4, an upper bound of the cost of the VAL_LI's horizontal shifts in each level *lev* is $(M/2^{lev})^2$. It results that the cost of all the shifts of VAL_LI is upper bounded by $M^2+(M/2)^2+\cdots+2^2=(4/3)(M^2-1)$.

(3) The cost of the decomposition within the level 0 (second step) is $3(2^{m-1}+2^{m-2}+\cdots+1)=3(M-1)$.

(4) The cost of the horizontal shifts of CONV (the last step) is $2(M-1)$. Hence, an upper bound of the total complexity is $\log N+\mathrm{NL}(T)+(4/3)\ (M^2-1)+5(M-1)$. Given that the number of black leaves of the T's quadtree is upper bound by $(3/4)M^2$, this total complexity is upper bounded by $\alpha \log N+\beta M^2$, where α and β are constants.

5. Conclusion

In this paper we presented a parallel algorithm on a pyramid, performing the template matching of an image I with a template T when both the image and the window are represented by their quadtrees. The complexity of this algorithm is comparable with the one presented in the only paper dealing with the same problem [2], since the two complexities are of the form $\alpha \log N+\beta M^2$. The difference between the two works is the data structure used to describe the image and the template: quadtree and matrix.

References

[1] D.H. Ballard and C.M. Brown, *Computer Vision* (Prentice-Hall, Englewood Cliffs, NJ, 1985).

[2] J.H. Chang, O.H. Ibarra, T.-C. Pong and S.M. Sohn, Two-dimensional convolution on a pyramid computer, *Internat. Conf. on Parallel Processing* (1987) 780–782.

[3] C. Chakrabarti and J.F. Jàjà, A parallel algorithm for template matching on an SIMD mesh connected computer, *Internat. Conf. on Pattern Recognition*, Vol II (1990) 362–367.

[4] Z. Fang, X. Li and L.M. Ni, Parallel algorithms for image template matching on hypercube SIMD computers, *Comput. Architecture Pattern Anal. Image Database* (1985) 33–40.

[5] Z. Fang, X. Li and L.M. Ni, Parallel algorithms for 2-D convolution, *Internat. Conf. on Parallel Processing* (1986) 262–269.

[6] I. Garagantini, An effective way to represent quadtrees, *Commun. ACM*, **25** (12) (1982) 905–910.

[7] H.T. Kung and S.W. Song, A systolic 2-D convolution chip, in: *Multicomputers and Image Processing Algorithms and Programs* (Academic Press, New York, 1982) 373–384.

[8] J.P. Lauzon, D.M. Mark, L. Kikushi and J.A. Guevara, Two dimensional runencoding for quadtree representation, *Comput. Vision, Graphics Image Process.* **30** (1985) 56–69.

[9] S.-Y. Lee and J.K. Aggarwal, Parallel 2-D convolution on a Mesh connected array processor, *Computer Vision Pattern Recognition* (1986) 305–310.

[10] M. Maresca and H. Li, Morphological operations on mesh connected architecture: a generalized convolution algorithm, *Computer Vision Pattern Recognition* (1986) 299–304.

[11] V.K. Prasana Kumar and V. Krishnan, Efficient image template matching on hypercube SIMD arrays, *Internat. Conf. on Parallel Process.* (1987) 765–771.

[12] S. Ranka and S. Sahni, Convolution on an SIMD mesh connected computer, *Internat. Conf. on Parallel Process.*, Vol 3 (1988) 212–217.

[13] S. Ranka and S. Sahni, Convolution on an SIMD hypercube multicomputers, *Internat. Conf. on Parallel Process.*, Vol 3 (1988) 84–91.

[14] S. Ranka and S. Sahni, Image template matching on MIMD hypercube multicomputers, *J. Parallel Distributed Comput.* **10** (1990) 79–84.

[15] A. Rosenfeld and A.C. Kak, *Digital Picture Processing* (Academic Press, New York, 1982).

[16] H. Samet, The Quadtree and related hierarchical data structures, *Comput. Surveys.* **16** (1984) 187–260.

[17] H. Samet and M. Tamminen, Computing geometric properties of images represented by linear quadtrees, *IEEE Trans Pattern Anal. Mach. Intelligence* **PAMI-7** (1985) 229–240.

[18] H. Senoussi and A. Saoudi, Quadtree algorithms for template matching on pyramid computer, Internal Report, LIPN, Institut Galilée, Université Paris XIII.

[19] H. Senoussi and A. Saoudi, Quadtree algorithms for template matching on pyramid computer, in: *Proc. Asian Conf. on Computer Vision*, Osaka Japan (1993 November 23–25).

[20] H. Senoussi and A. Saoudi, Quadtree algorithm for template matching on mesh connected computer, in *Proc, Computer Architectures for Machine Perception '93*, CAMP '93, New Orleans, Louisiana, USA (15–17 December 1993), to appear.

[21] P.J. Varman and I.V. Ramakrishnan, A tree algorithm for two-dimensional convolution, *Internat. Conf. on Pattern Recognition* (1984) 358–360.

[22] E.L. Zapata, J.I. Benavides, O.G. Plata and F.F. Rivera, Image template matching on hypercube SIMD computers, *Signal Processing* **21** (1990) 49–60.