Pergamon

Pattern Recognition, Vol. 31, No. 9, pp. 1391–1406, 1998

0031-3203/98 $19.00 + 0.00

PII: S0031-3203(97)00154-4

VLSI FOR MOMENT COMPUTATION AND ITS APPLICATION TO BREAST CANCER DETECTION

H. D. CHENG,*‡ C. Y. WU,* D. L. HUNG†

*Department of Computer Science, Utah State University, Logan, UT 84322-4205, U.S.A
†School of EECS, Washington State University-TC Richland, WA 99352, U.S.A.

(*Received* 22 *August* 1997; *in revised form* 6 *November* 1997)

Abstract—Moment has been one of the most popular techniques for image processing, pattern classification and computer vision. In this paper, we propose two VLSI architectures for computing the regular (geometric) moments and central moments. First, a one-dimensional systolic array is presented. In this architecture, a dynamic time delay controller is used for obtaining the correct data flow. It takes $\max(p, q) \times n + n + 2$ time units to compute the moments of order $(p + q)$. If there are k images, the computational time will be $k[\max(p, q) \times n + n + 2]$. Second, a two-dimensional architecture is presented. It takes $n + n + n - 1 + 2 + 1 = 3n + 2$ time units to compute the moments of order $(p + q)$. If there are k images, the computational time will be $(k - 1) \times 2n + 3n + 2 = 2nk + n + 2$. The proposed approaches are much faster than the existing ones. If a uniprocessor is used, the time complexity is $(p + q)\, n^2$, and if there are k images, the computational time will be $k(p + q)\, n^2$. Finally, a VLSI architecture is presented for calculating the central moments. In this architecture, $3 \times n$ processing elements are used for calculating m_{00}, m_{01}, and m_{10}. The results are sent to a two-dimensional structure for computing central moments. It takes $2n + 3 + \max(p, q) + 2 + n + n - 1 + 1 = 4n + \max(p, q) + 5$ time units to finish the calculation of the central moments. The important issue of VLSI design, algorithm partition, is also addressed. The basic idea of this paper can be extended to compute other kinds of moments easily. We have applied the moments for extracting the features of breast cancer biopsy images and classified them using neural networks. The 100% classification rate has been achieved. © 1998 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved

Moments Features Pattern classification VLSI architectures Algorithm partition Breast cancer detection

1. INTRODUCTION

The task of recognizing an object independent of its position, size, or orientation is very important for many applications of computer vision, pattern recognition, and image processing. In every aspect of developing a pattern recognition system, we should always carefully determine and extract the characteristics of the pattern. When the pattern undergoes rotation, translation, or scaling, the extracted features are more crucial for the recognition result.

Many techniques have been developed to extract features which are invariant under the changes in position, size, or orientation of the images. In early 1960s, Hu[1] published the paper on the moment invariant for two-dimensional (2-D) pattern recognition based on the methods of algebraic invariants. Since then, many researchers applied the moment invariants for pattern classification, image processing, and image description successfully. Teh and Chin[2] evaluated a number of moments for pattern recognition, such as regular moments, Legendre moments, Zernike moments, pseudo-Zernike moments, rotational moments, and complex moments. The properties of each moment were discussed. They demonstrated that the invariance properties were perfect if the image was noise free. However, when there was noise in the image, the invariance properties were not as good as expected. Also, higher-order moments were more sensitive to the noise of images than lower-order moments. When the moments are not orthogonal, the calculated moments will have redundant information, thus cause less accuracy of representing images. Teague[3] also summarized some well-known properties of the zero-order, first-order and second-order moments. He discussed the problems of image reconstruction from the inverse moments, and suggested using the orthogonal moments to recover an image. Yeaser and Psaltis[4] discussed the image recognitive aspects of moment invariants. They focused the discussions on the information loss, suppression, and the redundancy encountered in the complex moments. Dudani[5] used the same set of moment invariants to recognize different types of aircrafts. The similarity in the template matching could be measured by correlation coefficient and the sum of absolute difference.[6] The experimental results were accurate even when the object contained noise, but the computational cost was very high. Extracting low-level features from an image, such as gradient magnitudes, is another good method. However, the computation cost is varied with

‡Author to whom correspondence should be addressed.

the size of image, and it is less accurate when the image contains noise. Chou and Chen[6] proposed a two-stage pattern matching method called "moment-preserving quantization" that reduced the complexity of computation with quite a good accuracy. They also proposed a low-cost VLSI implementation. Their method could locate a 24×24 template in a 64×64 search area 1.5 times faster than "thresholding the magnitude of the gradient" method, and five times faster than "correlation coefficient" method. In reference (7), Wong and Hall used a set of moments which were invariant to translation, rotation, and scale changes to perform the scene matching of radar to optical images. Because of the invariance properties, it is easy to determine the threshold value, and an efficient match can be accomplished. Casey[8] used the second-order moments to specify the transformation of the coordinate wave forms. The hand-printed characters are transformed linearly in order to make a more uniform appearance for recognition. By using moments, the original pattern was mapped into a variation-free domain, and the linear pattern variation of the hand-printed characters was removed. It indicated that doing a moment normalization for the hand-printed character patterns before scanning would reduce the error rate. Cash and Hatamian[9] used 2-D moments to extract the pattern features of the optical character. This paper showed that the pattern features extracted from the moments provided good discrimination between characters, and 98.5–99.7% recognition rates were achieved for the tested fonts. Ghoal and Mehrotra proposed a subpixel edge detection method based on a set of orthogonal complex moments, Zernike moments, of the image[10]. Although using moments invariants to extract the features of objects is efficient and easy, it is lacking from the embedded noise of the objects. Due to the orthogonal and the invariance property of the Zernike moments, the subpixel edge detection was more efficiently accomplished even when the noise existed. Khotanzad and Hong[11] also proposed an invariant image recognition method by using the Zernike moments. Because the non-orthogonal moments do not provide good accuracy for recovering the images from moments and are sensitive to noise, reference (11) used the Zernike moments, that were orthogonal and invariant, to extract the features for pattern recognition. They tested the proposed method using clean and noisy images from a 26-class data. Due to the orthogonality property of the Zernike moments, the process of image reconstruction was simplified and the feature selection was easy and practical. Sardana, Daemi, Sanders and Ibrahim[12] applied the second-order moments to extract the feature vectors that could describe objects efficiently in an n-dimensional space. Liu and Tsai[13] proposed a corner detection method based on the moments. The moments of images were big factors of choosing the threshold value. Belkasim, Shridhar and Ahmadi gave a detail study of the efficiencies of different moment invariants in pattern recognition applications.[14] They proposed a new method for deriving Zernike moments with a new normalization scheme, and obtained a better overall performance even when the data contained noise. Reeves, Prokop and Andrews[15] presented a procedure using moment-based feature vectors to identify a 3-D object from a 2-D image recorded at an arbitrary angle and range. They compared two methods: moments and Fourier descriptors. They proposed a moment called standard moment which was normalized with respect to scale, translation, and rotation, and proved that the standard moment gave a slightly better result than the Fourier descriptors.

The discussions above indicate that moments have become one of the most popular and useful methods for image processing, pattern recognition and computer vision. We apply moments to extracting the features of breast cancer biopsy images and the resultant features are input into neural networks for classifications. A 100% classification rate has been achieved. In order to reduce the computational time, we will study the VLSI implementations of regular moments and central moments. The proposed idea can be extended to compute other moments easily.

1.1. Moments and moment invariants

The regular or geometric 2-D moments of order $(p + q)$ of an area A, for a continuous image $f(x, y)$, is defined as

$$M_{pq} = \int\int_A x^p y^q f(x, y)\,\mathrm{d}x\,\mathrm{d}y, \tag{1}$$

where $p, q \in \{0, 1, 2, \ldots\}$. From equation (1), we can compute the moments of a digital image of area A as

$$M_{pq} = \sum_{(x,y)\in A} x^p y^q f(x, y). \tag{2}$$

In order to obtain translation invariant, we can first locate the centroid of the image, that

$$\bar{x} = \frac{M_{10}}{M_{00}}, \quad \bar{y} = \frac{M_{01}}{M_{00}}. \tag{3}$$

We can define the central moments of a digital image with area A, which are translation invariant:

$$\mu_{pq} = \sum_{(x,y)\in A} (x - \bar{x})^p (y - \bar{y})^q f(x, y). \tag{4}$$

From equations (2) and (4), we can see that it takes a lot of additions and multiplications to compute the regular moments and central moments. Many real-world applications, such as industrial, military, environmental, medical and remote-sensing tasks, require real-time processing. When the moments are used as the features of the images with large sizes, the moment computation can become the bottleneck of the entire system. Several efficient methods have been proposed to speed up moment computation. References (16, 17)

discussed the superiority of boundary-based computation and proposed a simpler algorithm based on Green's theorem. They converted the double integrals into a linear integral around the boundary of the polygon. Both algorithms are only suitable for binary images, i.e, $f(x, y) \in \{0, 1\}$. Reference (16) did not include the time for finding the boundaries (vertices) which could be the majority of the total computation time. Reference (17) gave the consideration of the entire computation and proposed to use an upper triangular systolic structure to speed up the computation. But reference (17) did not include the time for calculating the required linear transform matrix of the algorithm. Chen[18] developed a parallel algorithm for computing moments based on decomposing a 2-D moment into vertical and horizontal moments, and used a so-called cascade-partial-sum method to compute the summation of the partial results. The time complexity for calculating the moments is $O((p+1)(q+1+c_1)\log(n) + (q+1+c_2)n)$ for a linear array, where c_1 and c_2 are the times spent in intermediate summations for the cascade-partial-sum method, and the values will be increased irregularly when p and q increase. Many processors in the 2-D processor array are idle for most of the time of computation. Reference (18) assumed the data were preloaded into the processor array. It would take at least n time units to load the data which was not included in the time analysis. The cascade-partial-sum method was employed to perform the calculation of higher-order moments, and would take more time for the intermediate summations as discussed above. In addition, because the data flow was irregular that caused difficulty in achieving correct timing and because the cascade-partial-sum method used recursive scheme, the memory (number of processors) had to be varied for different order of moments. Therefore, this algorithm is not suitable to be implemented using VLSI architecture[19,20].

1.2. VLSI and image processing

Many VLSI architectures have been developed to implement parallel algorithms for image processing and pattern recognition. Cheng *et al.*[21–24] have developed several VLSI architectures for different applications. By using VLSI architecture, we can speed up the computation tremendously. In order to improve the performance, using both parallelism and pipelining, a VLSI architecture should have the following characteristics:[19,20]

(1) There are only a few different types of processing elements, and the communication between them is local, simple, and regular.
(2) Data flow is simple and regular. In the best case, the data flow is linear (only one direction).

In this paper, we present two VLSI architectures to compute regular moments and another VLSI architecture to compute central moments. Since the proposed structures use pipelining and parallel techniques extensively, the time complexity for computing the moments is greatly reduced. Also, the structure of each processing element is very simple, regular, and can be easily manufactured by VLSI technology.

2. VLSI ARCHITECTURES FOR THE COMPUTATION OF REGULAR MOMENTS

In this Section, we will discuss the VLSI architectures for calculating moments. The structures of three different processing elements are shown in Figs 1a, 3a and 6b, respectively, and the "time unit" used here is determined by the time needed for an operation. Within a processing element, the operation for calculating $x^p \times y^q \times f(x, y) + Out_{pre}$ must be done concurrently with other operations for reducing computational time.

2.1. One-dimensional VLSI architecture

For an $n \times n$ image, it is possible to use a 1-D VLSI structure to calculate the moments of order $(p+q)$ using the following algorithm:

$f(x, y)$ is the gray level of the image.
x and y are corresponding coordinates of the pixel.
$M_{p,q}$ are the moment of order $(p+q)$ of the image.

```
let x := 1;   M_pq := 0
while(x < n)
begin
y := 1
Out_pre(x, y) := 0
while(y < n)
begin
   Out_nex(x, y) := Out_pre(x, y) + x^p × y^q × f(x, y)
   Out_pre(x, y + 1) := Out_nex(x, y)
   y := y + 1
end/*while(y) */
y := y − 1
M_pq := M_pq + Out_nex(x, y)
x := x + 1
end /* while(x) */
```

The sequential algorithms with loops can be implemented using VLSI architectures[19,20]. Based on the above algorithm, we can input the data into a 1-D systolic array row by row (or column by column) to calculate $x^p \times y^q \times f(x, y)$ for each pixel in that row (column). Add the results in that row, and accumulate results of all rows. Using this procedure, we can calculate the moments of order $(p+q)$. The data have to meet the timing requirement and should be skewed. According to the above algorithm, two types of processing elements are needed for a 1-D systolic array. The first kind of processing elements will have:

Five inputs: $p, q, f(x, y)$, control signal, and the output from the left processing element;

Four outputs: p, q, the output to the right processing element, and a signal which starts the calculation of the processing element;

Three multipliers: one for calculating x^p, one for calculating y^q, and another for computing $x^p \times y^q \times f(x, y)$;

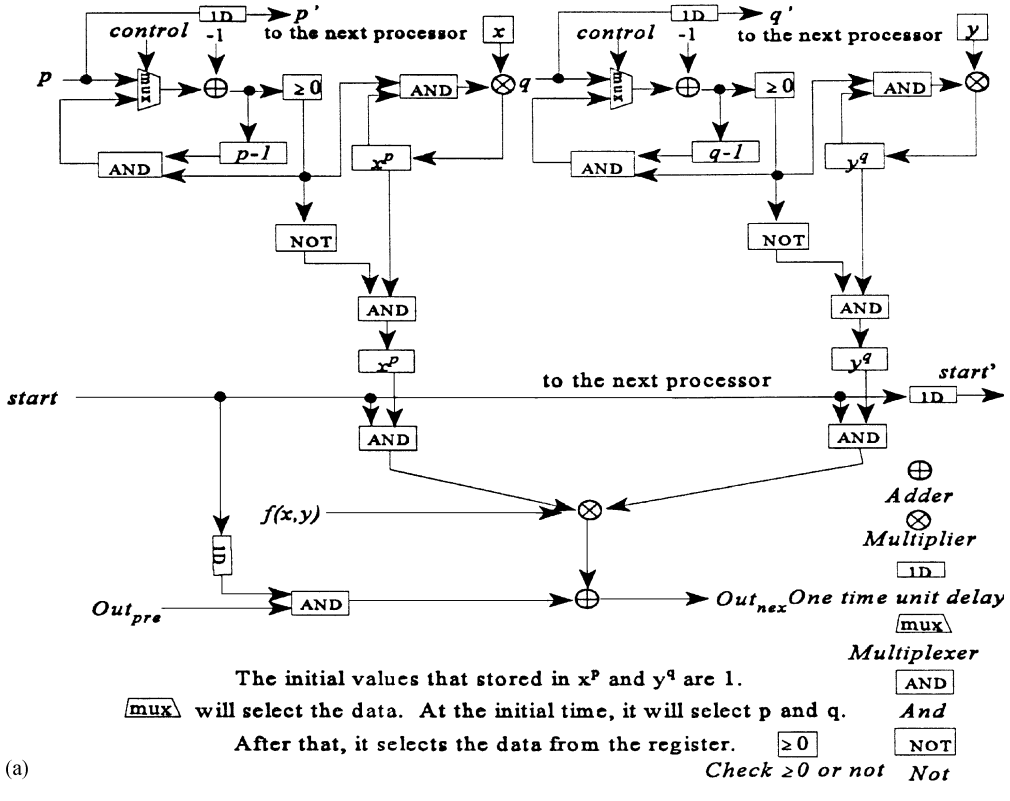
Two registers: to store the corresponding coordinates of the processing element;

One adder: to add the output from the left neighbor processing element and the currently calculated value;

The logical gates are necessary for controlling the timing of operations. Figure 1a shows the structure of the processing element, and its symbolic representation is shown in Fig. 1b. From Fig. 1a, it is clear that within a processing element, the operation for calculating $x^p \times y^q \times f(x, y) + Out_{pre}$ must be concurrent with other operations. The computation time needed for each processing element is $\max(p, q) + 2$. The second kind of processing element is an accumulator which is the $(n + 1)$th processing element.

Now we will discuss the functions performed by each processing element in detail.

2.1.1. Operations. Each processing element will receive $p, q, f(x, y)$, and the output from its previous processing element as the inputs. The data, $f(x, y)$, will be input to the 1-D array row by row (or column by column). At the first time unit, the processing element (1, 1) will receive p and q, store these values in the registers and start to calculate 1^p and 1^q. At the same



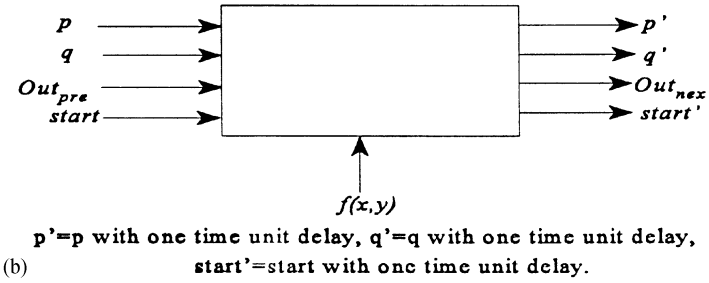


Fig. 1. (a) The structure of processing element for one-dimensional architecture. (b) The symbolic representation of the processing element.

time, p and q will be sent to the delays connected with the next processing element. Since 1^p and 1^q can be computed simultaneously, it takes $\max(p, q)$ time units to finish the computation. Meanwhile, the *start* signal takes $\max(p, q)$ time units to input $f(1, 1)$ and to calculate $Out_{nex} = Out_{pre} + 1^p \times 1^q \times f(1, 1)$. It takes two more time units to do the multiplication and addition (refer Fig. 1a). Thus, the total time to calculate $Out_{nex} = Out_{pre} + 1^p \times 1^q \times f(1, 1)$ is $\max(p, q) + 2$. The *start* signal will be input to processing element (1, 2) after one time unit, i.e., at the time unit $\max(p, q) + 1$. Note, the value of Out_{pre} for processing element (1, 1) is zero. At time unit $\max(p, q) + 2$, the processing element (1, 1) will produce the output and pass it to processing element (1, 2).

The values of p and q will arrive at processing element (1, 2) at the second time unit, and start to compute 1^p and 2^q. It also takes $\max(p, q)$ time units to calculate 1^p and 2^q. At the $\max(p, q) + 1$ time unit, the calculation for 1^p and 2^q is finished. At the same time, $f(1, 2)$ is input to processing element (1, 2). From the above discussion, at time unit $\max(p, q) + 1$, the *start* signal will arrive at processing element (1, 2). At the same time, the required data, 1^p, 2^q, and $f(1, 2)$ are ready. Thus, when the output from the previous processing element (1, 1), Out_{pre}, is received by processing element (1, 2) at the next time unit, then processing element (1, 2) will perform the multiplication and addition, and produce the output $Out_{nex} = Out_{pre} + 1^p \times 2^q \times f(1, 2)$ at the $\max(p, q) + 3$ time unit. The adjacent data of the same row needs one time unit delay in order to match the timing requirement.

Now, let us consider the data of the next row. In order to calculate $Out_{nex} = Out_{pre} + 2^p \times 1^q \times f(2, 1)$, the x-coordinate of processing element (1, 1) has to be increased by one after 1^p is calculated, i.e., at the pth time unit, x will be increased from 1 to 2. Then, $\max(p, q)$ time units are needed to calculate 2^p and 1^q. Thus, $\max(p, q)$ time units delay is necessary to input the first data in the second row. Here we want to indicate that for the data $f(x, y)$ $x = 1, 2, \ldots, n$ in the same column, they have the same value y^q, which was already computed during computing the moments of $f(1, y)$, and can be input for $f(x, y)$ without computing y^q again. But, we can make a trade-off between the regularity and uniformity of the processing elements and the simplicity of the structure. For most VLSI design, the regularity is more important, and we adapt this principle here. Figure 2 shows the VLSI structure and the arrangement of the data. For an $n \times n$ image, we need $n + 1$ processing elements to form the 1-D structure.

The operations of this structure are summarized as follows:

(1) p and q will be input to and stored in the first processing element at the first time unit. Then they are passed to and stored in the next processing element after one time unit, and so on.
(2) A *start* signal with $\max(p, q)$ time delay is needed for inputting the first data in the first row, and it will be passed to the next processing element after one time unit.
(3) A $\max(p, q)$ time delay is needed to input data of adjacent rows.

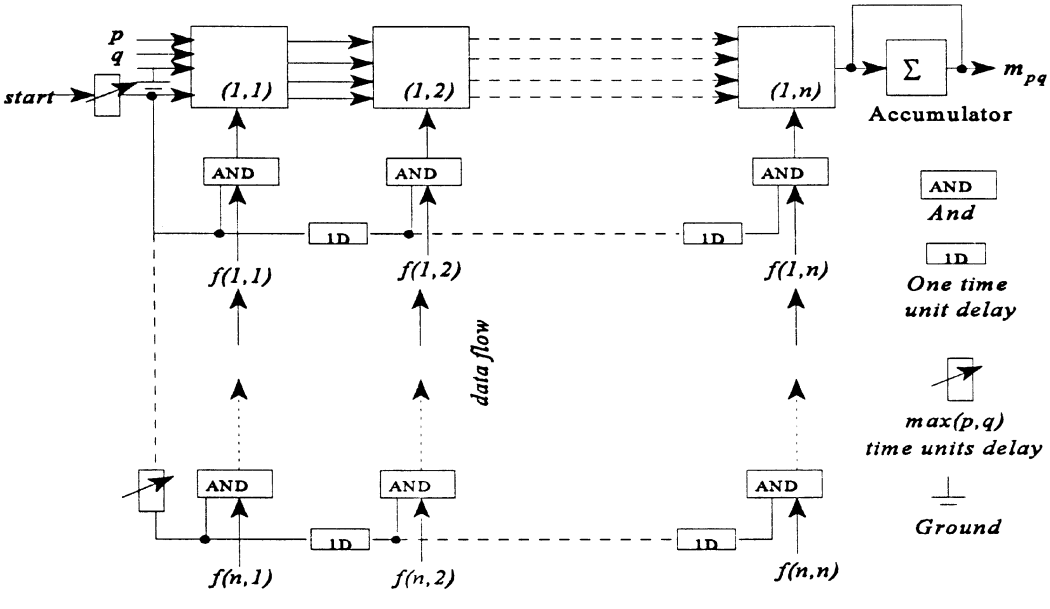


Fig. 2. One-dimensional VLSI architecture.

(4) The input data should be skewed.
(5) The x-coordinate of each processing element will be increased by one after the x^p is computed according to the corresponding data arrangement, and y-coordinate is fixed. Certainly, if we want to input data column by column, the roles of x and y will be switched.

Based on the above discussion, the time complexity is $O(\max(p, q) + 2 + n - 1 + \max(p, q) \times (n - 1) + 1) = O(\max(p, q) \times n + n + 2)$ for computing the $(p + q)$th moment of the image. If there are k images, the time complexity will be $k[\max(p, q) \times n + n + 2]$.

2.1.2. Verification of the proposed structure. From the above discussion, we can verify this 1-D array by induction on y.

Without loss of generality, we assume that each operation takes one time unit, and it takes one time unit for data to travel from one processing element to the next one. Also assume that the data are input row by row.

Theorem. It needs $\max(p, q) + 2 + y - 1 + \max(p, q) \times (x - 1) = \max(p, q) \times x + y + 1$ time units for the $(1, y)$th processing element to produce produce the output $\sum_{(s=1)}^{(s=x)} s^p y^q f(s, y)$.

Proof. It is obvious that it takes $\max(p, q)$ time units to calculate both x^p and y^q.

Basis: Consider the case when $x = 1$, $y = 1$, and processing element (1, 1). According to the discussion in Section 2.1.1, at the $\max(p, q)$th time unit, x^p and y^q are calculated, $f(1, 1)$, and *start* signal will arrive at processing element (1, 1). Then, two more time units are needed to perform a multiplication and an addition. So it takes $\max(p, q) + 2 = \max(p, q) + 2 + 1 - 1 + \max(p, q) \times (1 - 1)$ time units for the first processing element to perform the calculation and produce the output.

Induction step: Our induction hypothesis is that it takes $\max(p, q) + 2 + y - 1 + \max(p, q) \times (x - 1) = \max(p, q) \times x + y + 1$ time units for the $(1, y)$th processing element to produce the output $\sum_{(s=1)}^{(s=x)} s^p y^q f(s, y)$.

Now, consider data $f(x, y + 1)$ and the $(1, y + 1)$th processing element. According to the discussion in Section A, p and q will arrive at the $(1, y + 1)$th processing element at the $(y + 1)$th time unit. The x-coordinate of the (x, y)th processing element, $x > 1$, takes $\max(p, q) \times (x - 1) + y - 1$ time units to increase from 1 to x. Thus, at $\max(p, q) \times (x - 1) + y - 1$ time unit, the $(x, y + 1)$th processing element will start to calculate x^p and $(y + 1)^q$. It takes $\max(p, q)$ time unit to finish the calculation. Therefore, at the $\max(p, q) \times (x - 1) + y + 1 - 1 + \max(p, q) = \max(p, q) \times x + y$ time unit, x^p and $(y + 1)^q$ are calculated. At the same time, $f(x, y + 1)$ will arrive at the $(x, y + 1)$th processing element at the $\max(p, q) \times x + (y + 1) - 1 = \max(p, q) \times x + y$ time unit. Two more time units are needed to perform the multiplication and addition, so it takes $\max(p, q) \times x + (y + 1) - 1 + 2 = \max(p, q) \times x + y + 1$ time units for the processing element to perform the calculation and produce the output. The proof is completed.

Corollary 2.1-1. The regular moments for an $n \times n$ image can be obtained at the $max(p, q) \times n + n + 2$ time unit.

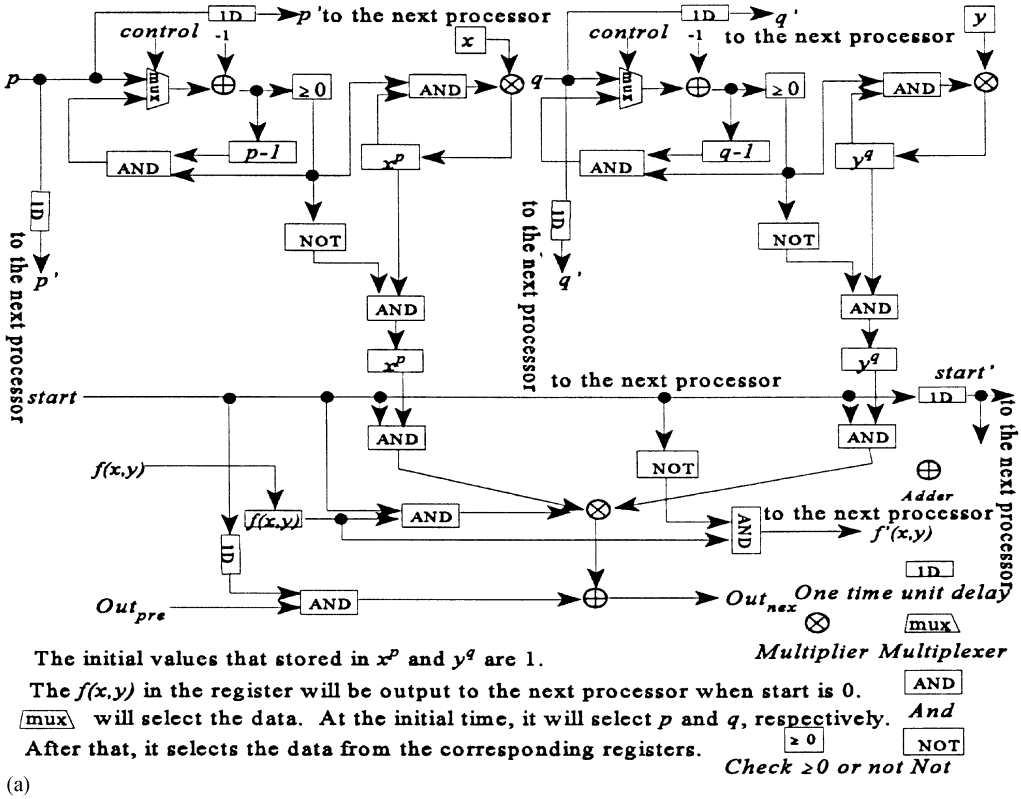
Proof. Follow the theorem, let $x = n$ and $y = n$, the output from the last processing element, $(1, y)$, can be obtained at the $\max(p, q) \times n + n + 1$ time units. Then, at the next time unit (required by the accumulator), $\max(p, q) \times n + n + 2$ time unit, moments of order $(p + q)$ can be obtained.

2.2. Two-dimensional VLSI architecture

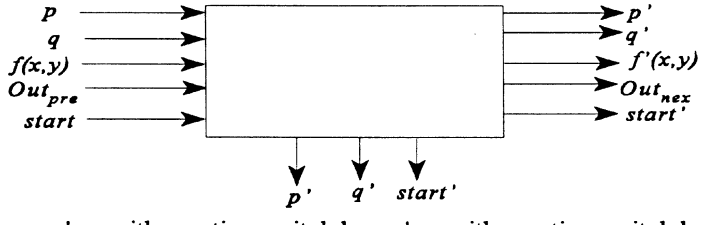
In this section, we present a 2-D VLSI architecture to perform the calculation of the moments with a much smaller time complexity. The basic algorithm for the 2-D architecture is as follows:

```
Let M_pq := 0
while not finished
if start = 0 then
for y := n to 1 do
  for x := 1 to n do
  begin
    store f(x, y) in the register of the (x, y)th processing element
    calculate x^p and y^q
  end /*for x*/
end /*for y*/
if start = 1 then
x := 1
while(x < n)
begin
  y := 1
  Out_pre(x, y) := 0;
  while (y < n)
  begin
    Out_nex(x, y) := Out_pre(x, y) + x^p × y^q × f(x, y)
    Out_pre(x, y + 1) := Out_nex(x, y)
    y := y + 1
  end /*while y*/
  y := y - 1
  M_pq := M_pq + Out_nex(x, y)
  x := x + 1
end /* while x*/
```

For this architecture, we need $n \times (n + 1)$ processing elements. The structure of each processing element is about the same as the one in 1-D VLSI architecture, but more outputs and inputs are needed. Figure 3a shows the structure of each processing element, and Fig. 3b is its symbolic representation.



(a)



(b) p '=p with one time unit delay, q '=q with one time unit delay, start '=start with one time unit delay, and f'(x,y)=f(x,y) with one time unit delay.

Fig. 3. (a) The structure of processing element for 2-D architecture. (b) The symbolic representation of the processing element.

2.2.1. Operations and verification of the proposed structure. In this architecture, we need to input the data from the last column (or row) to make sure that when the calculation is started, the data are in the correct processing elements. A start signal is employed to start the calculation. The *start* signal will not be issued until $f(1, 1)$ is input into processing element (1, 1). It takes n time units for $f(1, 1)$ to arrive at processing element (1, 1). Meanwhile, p, q, and $f(x, y)$ will be input to and stored in the correct processing element. Once the processing element receives p, q, and the corresponding $f(x, y)$, it starts to calculate x^p, y^q, and stores $f(x, y)$ in the register.

After the start signal is issued, processing element (1, 1) will need two more time units to perform the calculation of $Out_{nex} = Out_{pre} + x^p \times y^q \times f(x, y)$. In the next time unit, the start signal will be passed to the next processing element, processing element (1, 2), and to the processing element below, processing element (2, 1), and so on. Since the processing elements will start to calculate x^p and y^q right after p and q arrived, when the *start* signal is input, both x^p and y^q have been calculated. The processing elements will then perform the calculation and produce the output using $f(x, y)$ that is stored in the register of the corresponding processing element. It is clear that after the *start* signal is arrived, the time delay for the output data is one time unit. To calculate the moments for an $n \times n$ image, we need n time units delay for issuing the start signal. Then, another $2 + n + (n - 1) + 1 = 2n + 2$ time units are needed to complete the entire calculation. Totally, it takes $3n + 2$ time units to calculate the moments of order $(p + q)$ for an $n \times n$ image.

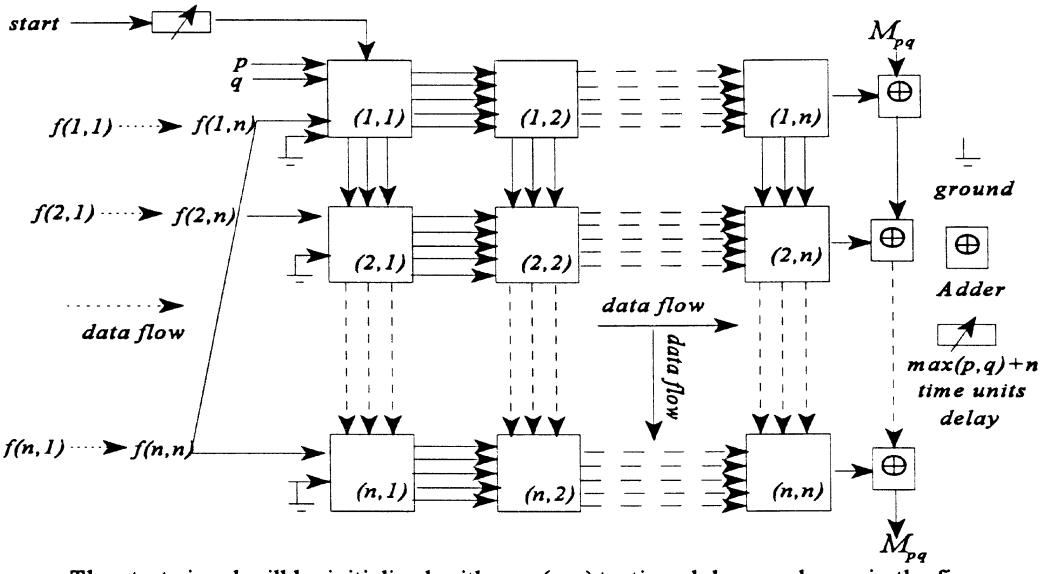


Fig. 4. The 2-D VLSI architecture.

From the above description, it is obviously that the structure will produce correct result with the time complexity projected. We simply verify the proposed structure and the time complexity using the method described in Section 2.1.

Figure 4 shows the entire 2-D VLSI architecture. From Fig. 4 we can see that the data are skewed and the last column of data, $f(1, n), f(2, n), \ldots, f(n, n)$, are input first, the first column of data, $f(1, 1)$, $f(2, 1), \ldots, f(n, 1)$, are input last. Reference (24) has a briefly description about how to arrange the data to ensure the correct data flow and timing.

The 2-D architecture needs only $3n + 2$ time units to finish the computation. If using uniprocessor, the time complexity is $O((p + q) \times n^2)$.

If there are k images to process, when the first image starts to compute, the second image can be input to the processing array, and the start signal will be sent out once for every n time units. The total time complexity is $2nk + n + 2$.

3. TWO-DIMENSIONAL VLSI ARCHITECTURE FOR CENTRAL MOMENTS

In this section, we propose a 2-D VLSI architecture for calculating the central moments. According to equation (3), for calculating $\bar{x}$ and $\bar{y}$, we have to compute M_{00}, M_{01} and M_{10}. From the definition of moments we know:

$$M_{00} = \sum_{(x,y)\in A} x^0 y^0 f(x, y) = \sum_{(x,y)\in A} f(x, y), \qquad (5)$$

$$M_{01} = \sum_{(x,y)\in A} x^0 y^1 f(x, y) = \sum_{(x,y)\in A} y f(x, y), \qquad (6)$$

$$M_{10} = \sum_{(x,y)\in A} x^1 y^0 f(x, y) = \sum_{(x,y)\in A} x f(x, y). \qquad (7)$$

The following algorithm is used to calculate M_{00}, M_{01} and M_{10}.

```
let M00(x):= M01(x):= M10(x):= 0
let M00:= M01:= M10:= 0
for y:= n to 1 do
  for x:= 1 to n do
  begin
    M00(x):= M00(x) + f(x, y)
    M01(x):= M01(x) + y × f(x, y)
    M10(x):= M10(x) + x × f(x, y)
  end /* for x*/
end /* for y*/
for x:= 1 to n do
begin
  M00:= M00 + M00(x)
  M01:= M01 + M01(x)
  M10:= M10 + M10(x)
end /* for */
```

A 2-D VLSI structure for calculating central moments shown in Fig. 6 consists of two subsystems:

- Subsystem A is to calculate $\bar{x}$ and $\bar{y}$.
- Subsystem B is to calculate the central moments.

According to the above algorithm, we can construct a VLSI architecture for computing $\bar{x}$ and $\bar{y}$. This VLSI architecture consists of $3 \times n$ processing elements. The first column is to compute M_{00}. The structure of the processing element in the first column of subsystem A is shown in Fig. 5a. The second column is to compute M_{10}. The processing element of the second column of subsystem A is shown in Fig. 5b. The value of x can be input (requiring input channel) or pre-stored in each processing element. The third column is for computing M_{01}. Each processing element of the third column of subsystem A is as shown in Fig 5c. We need to input or pre-store the value of y in each processing element of this column, and to input the data from the last column. This value will be decreased by one after the value is used until the last data is calculated.

To calculate the central moments, we can combine subsystem A with 2-D architecture (subsystem B) in Section 2 to form a new architecture as shown in

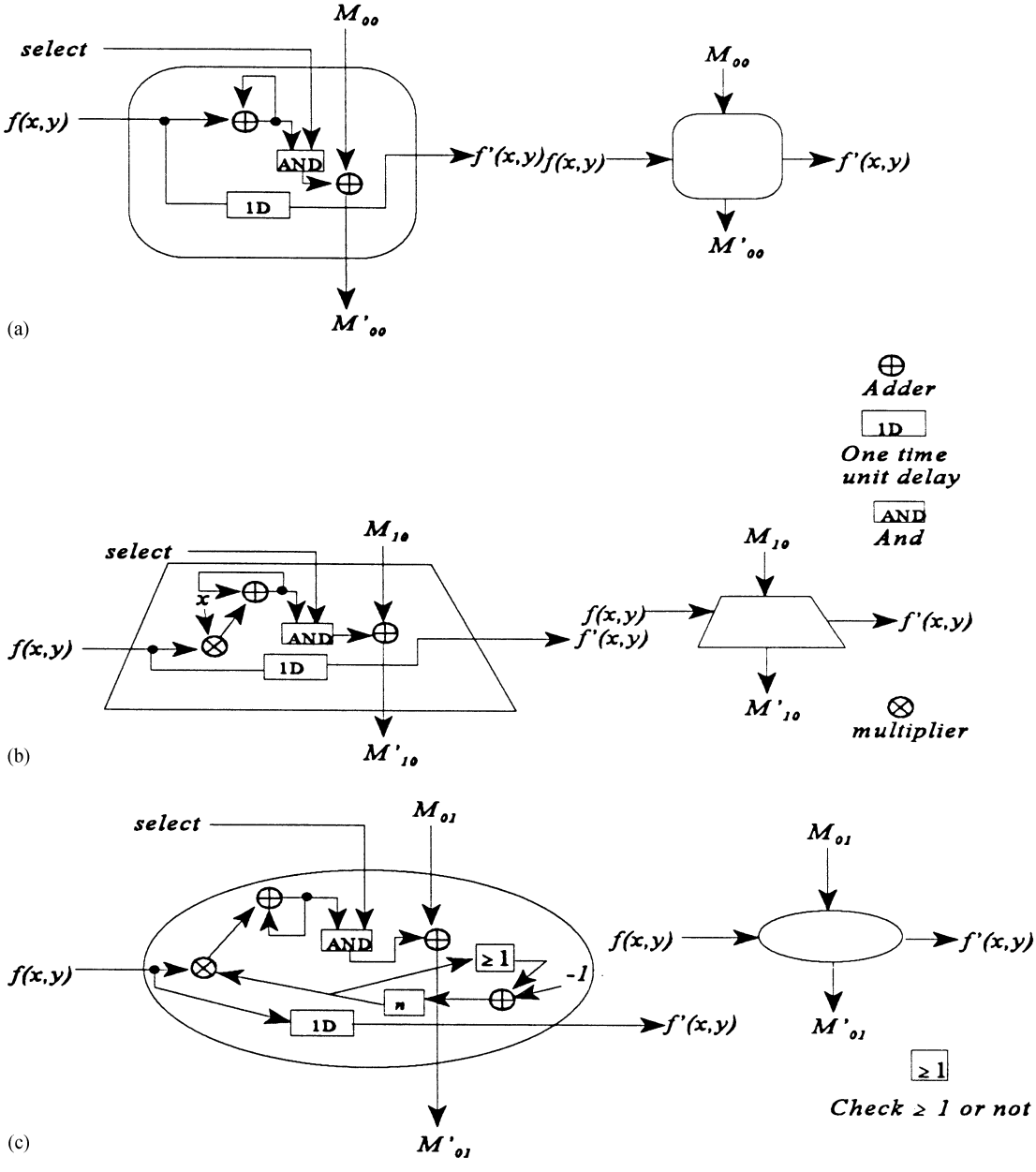


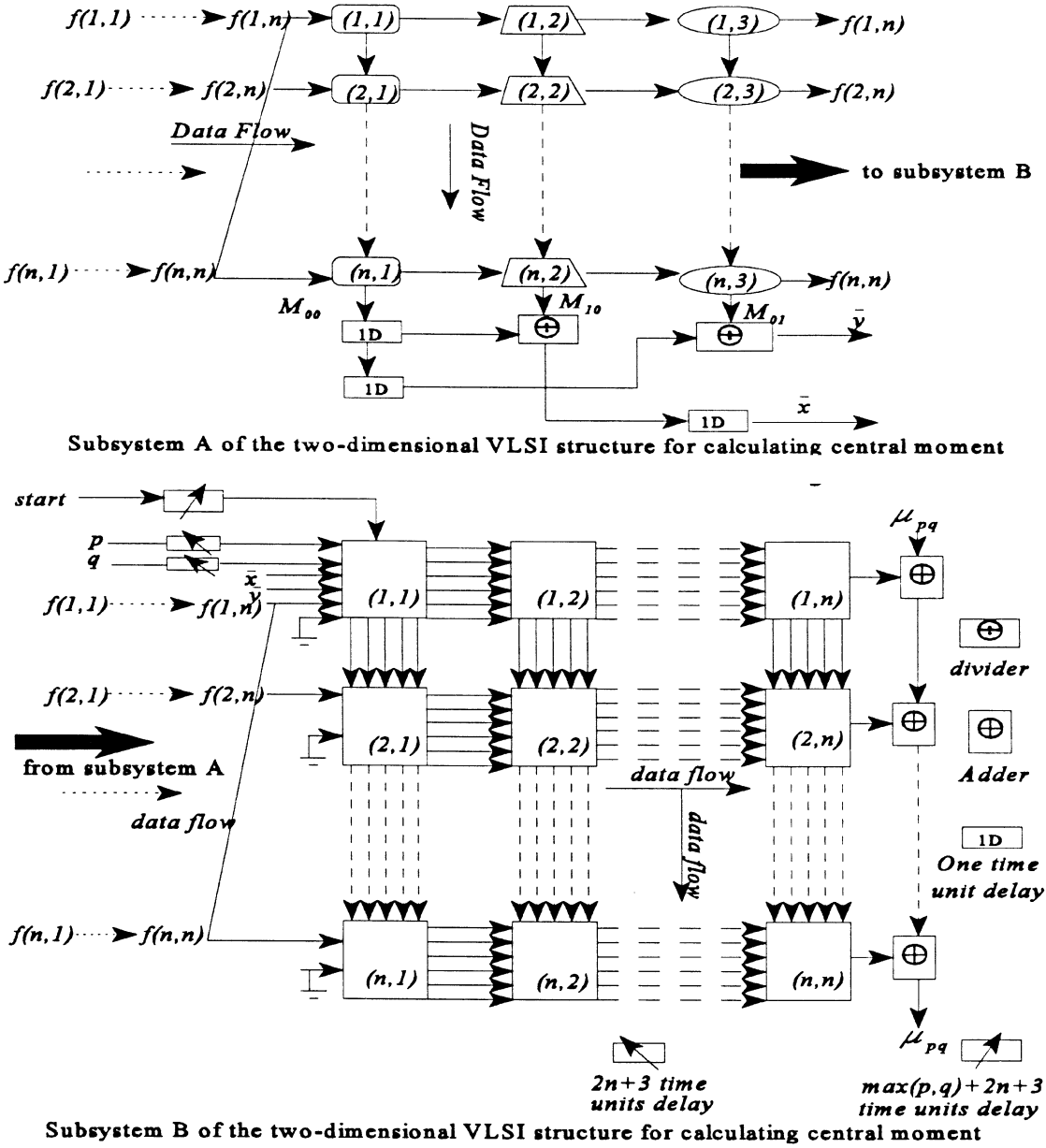
Fig. 5. (a) The structure of the processing element of the first column of subsystem Z. (b) The structure of the processing element of the second column of subsystem A. (c) The structure of the processing element of the third column of subsystem A.

Fig. 6(a). Let us discuss the functions performed by the proposed VLSI architecture.

3.1.1. Operations and verification of the proposed structure. Data $f(x, y)$ will be input in a column-major manner, and the last column of the data, $f(1, n), f(2, n), \ldots, f(n, n)$ are input first; the first column, $f(1, 1), f(2, 1), \ldots, f(n, 1)$ are input last.

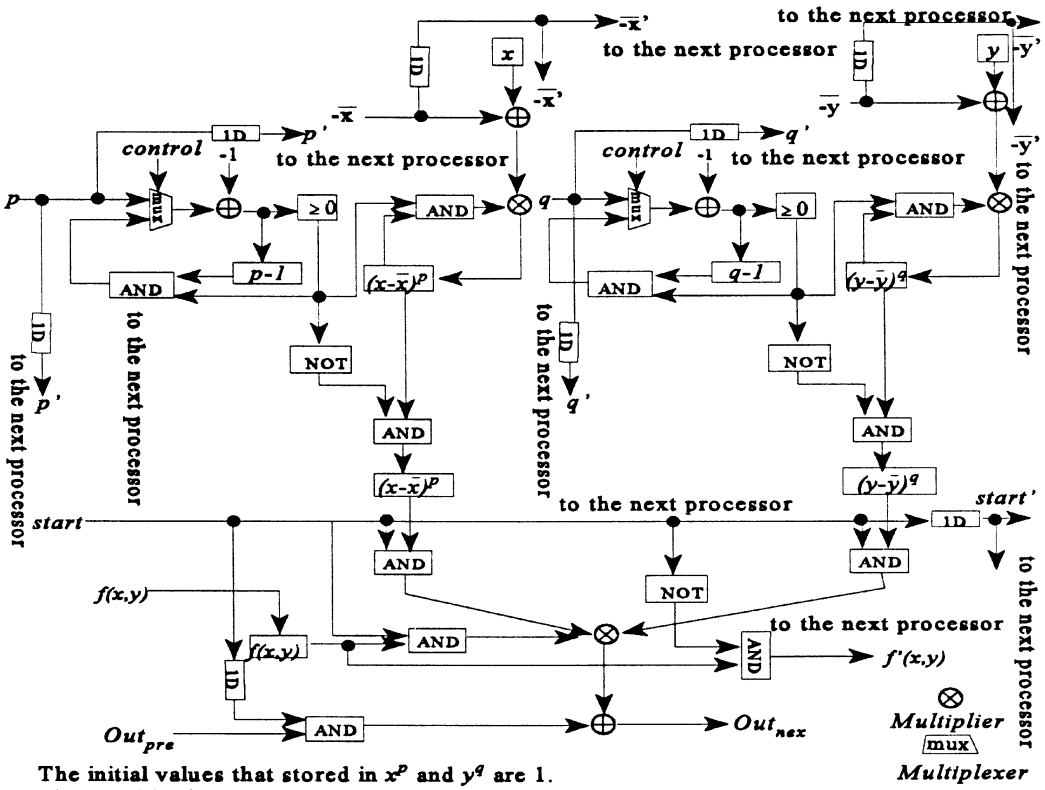
At the first time unit, the data will be input to subsystem A, and at the third time unit, the data will be input to subsystem B. In order to perform the correct calculation, we have to skew the input data to meet the time requirement. The data of the first row will arrive at the corresponding processing elements of subsystem B at the $(n + 3)$th time unit and will be stored in the registers and wait for the *start* signal to perform the calculation.

According to the data arrangement as shown in Fig. 6a, at the $(2n)$th time unit, M_{00} will be calculated and sent to the next column to calculate $\bar{x}$. At the next

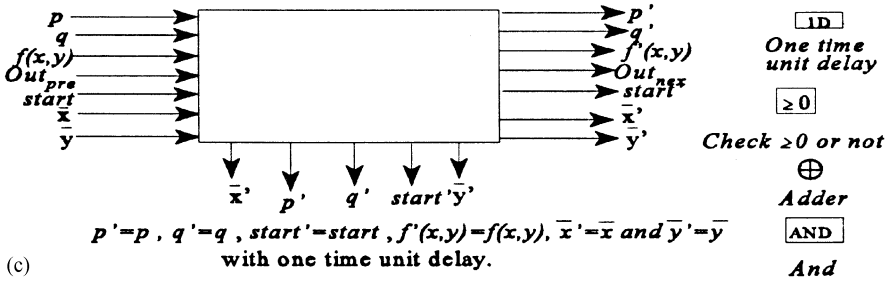


(a)

Fig. 6. (a) The 2-D VLSI structure for calculating the central moment. (b) The structure of processing element for 2-D architecture. (c) The symbolic representation of the processing element.



(b)



(c)

Fig. 6. (*continued*)

time unit, M_{10} will be calculated and used for computing $\bar{x}$, and M_{00} will be sent to the next column to perform the calculation of $\bar{y}$. At the next time unit, the $(2n+2)$ th time unit, M_{01} will be calculated and used for computing $\bar{y}$. Then $\bar{x}$ and $\bar{y}$ will be input to subsystem B.

We can calculate $(x-\bar{x})^p$ and $(y-\bar{y})^q$ at the $(2n+3)$th time unit, and it takes $\max(p, q)$ time units to calculate $(x-\bar{x})^p$ and $(y-\bar{y})^q$. Therefore, we have to arrange the data and *start* signal to be ready at the $(2n+3+\max(p, q))$th time unit. Hence, a *start* signal will start the calculation of subsystem B at the $(2n+3+\max(p, q)+1)$th time unit.

The structure of the processing element and its symbolical representation are shown in Fig. 6b and 6c, respectively. Follow the data arrangement and Fig. 6a, we can see that it needs $2n+3+\max(p, q)$ time units for issuing the *start* signal, and $2n-1+2+1=2n+2$ time units to perform the moments computation. Thus, it takes total $2n+3+\max(p, q)+2n+2=4n+\max(p, q)+5$ time units to finish the entire calculation.

From the above description, it is obvious that the structure will produce the correct result with the time complexity projected. We can verify the proposed structure and the complexity using the method in Section 2.1. The time complexity of this structure is $O(n)$. If a uniprocessor is used, the time complexity is $3n^2+(p+q)\times n^2=O(n^2)$.

4. ALGORITHM PARTITION

In the real-world applications, the images are often with very large sizes. When we use moments as the features, moment computation can become the bottleneck of the system. It is not practical to build a VLSI architecture with a huge amount of processing elements to suit all tasks with different sizes. Also, we do not want to design every new machine for each individual task even the only difference between the tasks is their sizes. Therefore, we should design a fixed-size VLSI architecture to solve this problem.

When a computational task size is larger than the VLSI architecture size, we have to partition the task into smaller subtasks to calculate the moments on a fixed-size VLSI architecture. If we have an image with size $k \times l$ and the 2-D VLSI architecture with size $m \times n$, and k, l are dividable by m and n, respectively, we can easily partition the image and calculate the moment using the size $m \times n$ VLSI architecture. If k and l cannot be divided by m and n, we can fill zeros to the last columns and the last rows of the image, such that, they can be divided by m and n, respectively.

Assume the dimension of the image is $k \times l$ and $\lceil k \div m \rceil = s$, $\lceil l \div n \rceil = t$. We can divide the image into $s \times t$ subimages with size $m \times n$. We use (s, t) to index the subimage. Each subimage can be input to the 2-D array to perform the calculation. In order to perform the correct calculation, two sets of sequential pulses are needed. The first set of sequential pulses with $n + m + \max(p, q)$ time units delay is needed to increase the y index of the coordinates (x, y) of the processing element. Another set of sequential pulses with $n + m + \max(p, q) + t$ delay is needed to reset the y index and to increase the x-coordinate of the corresponding processing element. Therefore, at the first time unit, subimage (1, 1) of the image is input to the VLSI structure, while the 2-D structure calculating the regular moment of the (1, 1)th subimage, the indices of the coordinates of the processing elements are: $(1, 1), (1, 2), \ldots (1, n), (2, 1), \ldots (2, n), (m, 1), \ldots, (m, n)$. When the processing elements finish calculating x^p and y^q, y can be increased to $y + (t - 1) \times n$ to match with the corresponding indices of the coordinates for the next subimage (1, 2), that are: $(1, n + 1), (1, n + 2), \ldots, (1, n + n), (2, n + 1), \ldots, (2, n + n), (m, 1), \ldots, (m, n + n)$. Clearly when the subimage $(1, t)$ is input to the VLSI structure, the indices of the coordinates (x, y) are: $(1, (t - 1) \times n + 1), \ldots (1, (t - 1) \times n + n), (2, (t - 1) \times n + 1), \ldots, (2, (t - 1) \times n + n), (m, (t - 1) \times n + 1), \ldots, (m, (t - 1) \times n + n)$. At the next time unit, the first pulse of the second set of sequential pulses is input, the processing elements reset the y indices to $1, 2, \ldots, n$ and increase x to $x + (s - 1) \times m$. This procedure will divide the $k \times l$ image into $s \times t$ subimages with the size $m \times n$. The VLSI architecture will compute regular moments of each subimage. Thus, an accumulator will add the $s \times t$ moments, and that is the regular moment of the original image. Similarly, the method can be used to calculate the central moments.

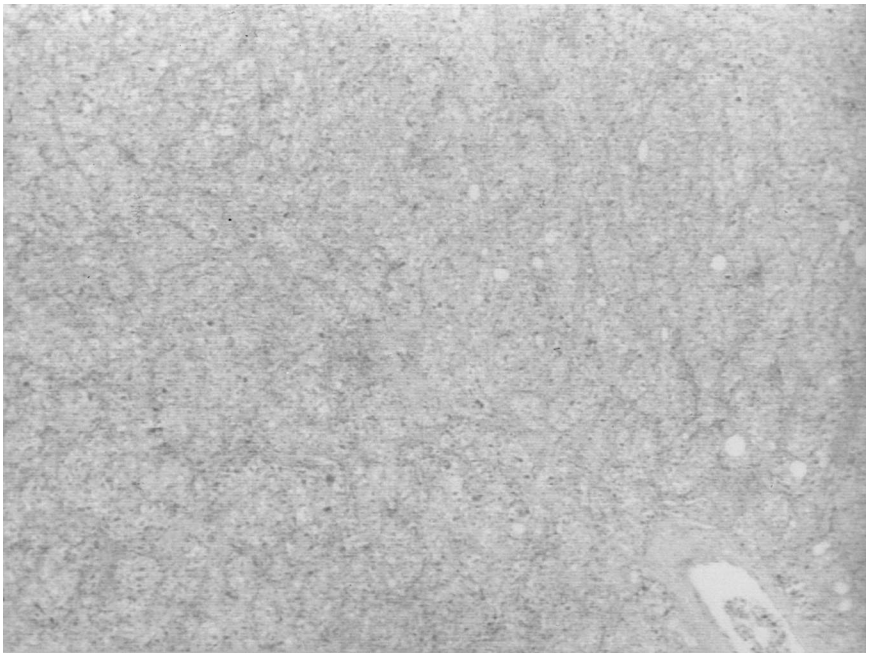


Fig. 7. Case 907408A (size 660×440) with Score 1.

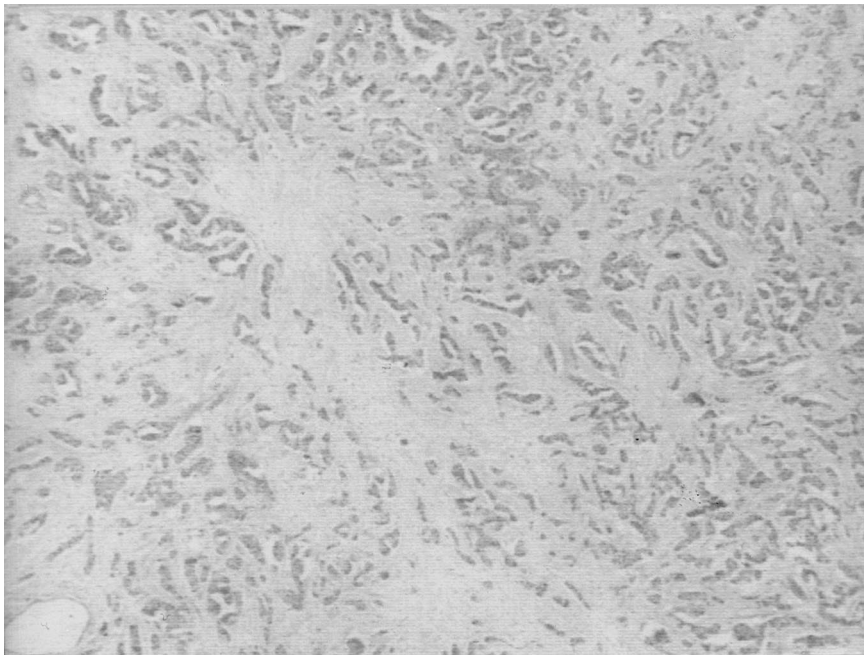


Fig. 8. Case 891280B (size 648×442) with Score 2.

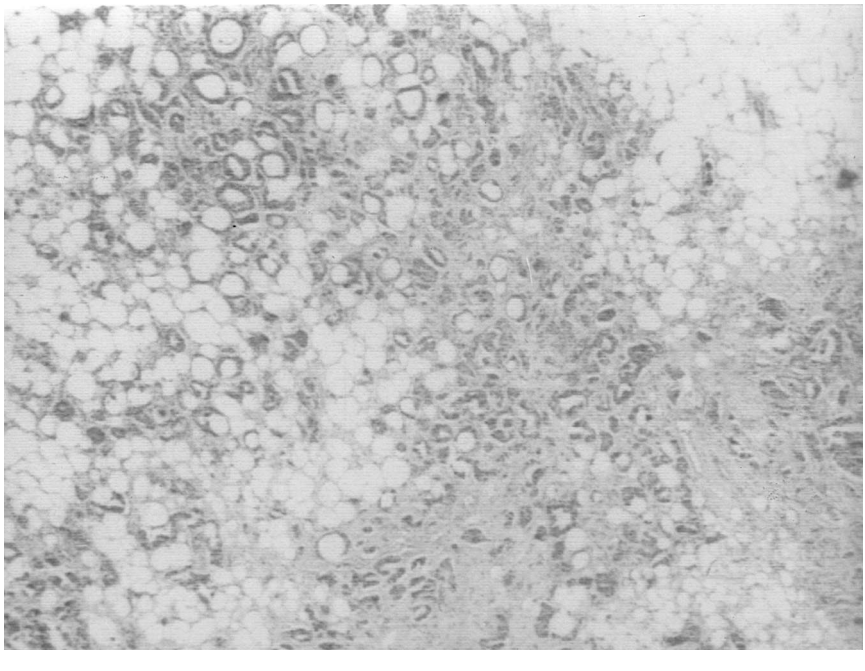


Fig. 9. Case 8912809A (size 648×442) with Score 3.

5. MOMENTS FOR BREAST CANCER DETECTION

We have employed central moments as the features for breast cancer biopsy images. The features are input into neural networks for classification. Based on the nature of neural networks, after training, neural networks can operate extremely fast, and the feature extraction (moment computation) becomes the bottleneck of the entire process. VLSI architectures for moments can solve such a problem.

Table 1. The first 25 central moments of Figs 7–9

Order of moment	Fig. 13	Fig. 14	Fig. 15
m_{00}	0.8889	0.9791	0.9004
m_{01}	0.0000	0.0000	0.0000
m_{02}	0.8644	0.9967	0.9163
m_{03}	− 0.0190	− 0.0190	− 0.5169
m_{04}	0.8284	1.0000	0.9188
m_{10}	0.0000	0.0000	0.0000
m_{11}	− 0.0352	− 0.4442	− 0.5646
m_{12}	0.0005	− 0.2889	0.0200
m_{13}	− 0.0815	− 0.6459	0.0038
m_{14}	0.0010	− 0.0468	− 0.0022
m_{20}	0.8707	0.9858	0.8697
m_{21}	− 0.0255	0.3696	0.1468
m_{22}	0.8429	1.0000	0.8822
m_{23}	− 0.0306	0.0488	0.0626
m_{24}	0.8042	1.0000	0.8815
m_{30}	− 0.2534	− 0.3456	− 0.2952
m_{31}	− 0.1086	− 0.6451	− 0.1421
m_{32}	− 0.2380	− 0.3734	− 0.2880
m_{33}	− 0.2498	− 0.8744	− 0.0522
m_{34}	− 0.2280	− 0.2954	− 0.3043
m_{40}	0.8468	0.9909	0.8343
m_{41}	− 0.0074	0.1853	0.1603
m_{42}	0.8151	1.0000	0.8424
m_{43}	− 0.0034	− 0.1613	0.0679
m_{44}	0.7772	1.0000	0.8416

Table 2. Comparison of the results by computer and physicians

Case number	Score rated by physicians	Score rated by computer
8912809A	3	3
8912809B	2	2
8922992A	2	2
90336T3A	1	1
90336T3B	1	1
905840A	3	3
905840B	3	3
907408A	1	1
907408B	1	1
909511A	2	2

Table 3. Comparison of the time complexities of faster moment calculation approaches. m is the number of boundaries (edges) of polygon, n^* is the number of pixels in each edge, n is the dimension of the image, (according to the discussion in Section 1, Chen's method is not suitable for the VLSI implements)

Method	Time complexity			
	Uniprocessor	Multiprocessor		VLSI
		Linear array	2-D array	
Jiang's method	$O(mn^*)$	×		Y
Li's method	$O(mn^*)$	×		Y
Chen's method	×	$O(n)$	$O(n)$	N
Our method	×	$O(n)$	$O(n)$	Y

Table 4. The time complexities of PC, PE and 1-D array for computing M_{55}. (Certainly, if using large K and computing more moments, the advantage of VLSI architectures will be even great)

Image size	PC (s)	PE (s)	1-D array with K PEs
512×512	2.1	0.147	$0.147/K$
1024×1024	8.4	0.587	$0.58/K$

Breast cancer continues to be a significant public health problem in the United States. One out of eight women will develop cancer at some point during her lifetime. The earlier stage tumors are more easily and less expensively treated. Because of this fact, and because of the high incidence of breast cancer, any improvement in the process of diagnosing the disease may have a significant impact on years of life saved and cost of the health care system.

Here, we will grade the breast cancer in biopsy images. It is necessary for a physician to distinguish between benign lesions and the various degrees of malignant lesions from mammography or biopsy images. Normally, three degrees are classified for the biopsy images of breast cancer. When there is a majority of tubular structure in the image, a high score three is given. When there are only a little or no tubular structure in the image, score one is given. Intermediate differentiation obtains a score of two.[25] To speed up the process and increase the accuracy, we use 25 central moments ($p, q = 0, 1, 2, 3, 4$) as the features of the biopsy images, and a neural network as a classifier to grade the tubules of the images. We have conducted some experiments of tubule grading using breast cancer biopsy images. We first perform feature extraction on 591 breast cancer images. Figures 7–9 are the sample breast cancer images with scores 1, 2, and 3, respectively, and their corresponding moments are listed in Table 1. The image sizes are 660×440, 648×442, and 648×442, respectively. We construct a training set by randomly selecting 472 images from the entire breast cancer images set, and the other 119 images as testing set. Their central moments were computed and used as the inputs of the neural network. After 2 h of training on a Pentium 200 PC, the neural network successfully converged. Then, we use the resulting neural network to classify the testing images into three classes without any mis-classification. Table 2 shows part of the results from the experiments.

The approach to adopt specially designed hardware accelerator for speeding up the moment computation has been studied.[26] Register-transfer level (RTL) design of core functional units was discussed. Testing results based on implementation using FPGA (field programmable gate array) devices show that at an affordable cost, the proposed moment accelerator can speed up the moment computation significantly. We

used Xilinx XC4020E FPGA (density: about 20,000 gates per chip) to implement a PE which needed two XC4020Es, and total gate count of the PE was below 25,000 gates. We also used a Pentium 166 PC to compute the moments for comparison.

The comparison of the time complexities of the faster algorithms discussed in Section 1 is shown in Table 3. Also, the comparison of the time complexities of the PC, PE and 1-D array is listed in Table 4. For example, in order to compute M_{55} for an image with size 512×512, if using a Pentium 166 PC, a PE and a 1-D array of 100 PEs, it takes 2.1 s. 0.147 s, and 1.47 ms, respectively. We may use more PEs or a 2-D array to speed the computation further.

6. CONCLUSIONS

We study two VLSI implementations for calculating the moments of order $(p + q)$. For the 1-D structure, it needs fewer processing elements, and the total calculation time is $\max(p, q) \times n + n + 2$ time units. For the 2-D structure, it needs more processing elements, but the total calculation time is $3n + 2$ time units. If there are k images, the computational time for the 1-D structure is $k[\max(p, q) \times n + (n + 2)]$. For the 2-D architecture, it takes $(k - 1) \times 2n + 3n + 2 = 2nk + n + 2$. If a uniprocessor is used, the time complexity is $k \times (p + q)\, n^2$. We also present a 2-D VLSI architecture for calculating the central moment which takes $4n + 5 + \max(p, q)$ time units. The important issue in VLSI design, algorithm partition, is also studied.

We have employed central moments as the features for breast cancer grading. The preliminary results demonstrate the 100% accuracy. Moments are very useful for many applications and the proposed architecture can speed up the computation greatly, therefore, it will have wide applications in the areas of image processing, pattern recognition and computer vision, especially, for real-time processing.

REFERENCES

1. M. K. Hu, Visual pattern recognition by moment invariants, *IRE Trans. Inform. Theory* **IT-8**, (February 1962).
2. C. H. Teh and R. T. Chin, On image analysis by the methods of moments, *IEEE Trans. Pattern Anal. Mach. Intell.* **10**(4) (July 1988).
3. M. R. Teague, Image analysis via the general theory of moments, *J. Opt. Soc. Am.* **70** (8), 920–930 (August 1980).
4. Y. S. Abu-Mostafa and D. Psaltis, Recognitive aspects of moment invariants, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-6**(6) (November 1984).
5. S. A. Dudani, K. J. Kenneth and R, B. McGhee, Aircraft identification by moment invariants, *IEEE Trans. Comput.* **26**, 39–46 (1977).
6. C. H. Chou and Y. C. Chen, Moment-preserving pattern matching, *Pattern Recognition* **23**, (5) 461–474 (1990).
7. R. Y. Wong and E. L. Hall, Scene matching with invariant moments, *Comput. Graphics Image Process.* **8**, 16–24 (1978).
8. R. G. Casey, Moment normalization of handprinted characters, *IBM J. Res. Dev.* (1970).
9. G. L. Cash and M. Hatamian, Optical Character Recognition by the method of moments, *Comput. Vision Graphics Image Process.* **39**, 291–310 (1987).
10. S. Ghosal and R. Mehrotra, Orthogonal moment operators for subpixel edge detection, *Pattern Recognition* **26**, (2) 295–306 (1993).
11. A. Khotanzad and Y. H. Hong, Invariant image recognition by Zernike moments, *IEEE Trans. Pattern Anal Mach. Intell.* **12**, (5), 489–497 (May 1990).
12. H. K. Sardana, M. F. Daemi, A. Sanders and M. K. Ibrahim, A novel moment-base shape description and recognition technique, *IEE the 4th Intel. Conf. on Image processing and it Applications*, pp. 147–150 (April 1992).
13. S. T. Liu and W. H. Tsai, Moment-preserving corner detection, *Pattern Recognition* **23**, (5) 441–460 (1990).
14. S. O. Belkasim, M. Shridhar and M. Ahmadi, Pattern recognition with moment invariants: a comparative study and new results, *Pattern Recognition*, **24** (12), 1117–1138 (1991).
15. A. P. Reeves, R. J. Prokop and S. E. Andrews, Three-dimensional shape analysis using moments and fourier descriptor, *IEEE Trans. Pattern Anal. Mach. Intell.* **10** (6) (November 1988).
16. X. Y. Jiang and H. Bunke, Simple and fast computation of moments, *Pattern Recognition*, **24** (8) 801–806 (1991).
17. B. C. Li and J. Shen, Fast computation of moment invariants, *Pattern Recognition* **2** (8) 807–813 (1991).
18. K. Chen, Efficient parallel algorithms for the computation of two-dimensional image moments, *Pattern Recognition*, **23**, (1/2) 109–119 (1990).
19. H. D. Cheng, Y. Y. Tang and C. Y. Suen, Parallel image transformation and its VLSI implementation, *Pattern Recognition* **23** (10) 1113–1129 (1990).
20. H. T. Kung, Why systolic architectures?, *Computer*, 37–46 (January 1982).
21. H. D. Cheng and K. S. Fu, VLSI architectures for string matching and pattern matching, *Pattern Recognition*, **20** (1) 125–141 (1987).
22. H. D. Cheng and K. S. Fu, VLSI architecture for dynamic time-warp recognition of handwritten symbols, *IEEE Trans. Acoustics Speech Signal Proces.* **ASSP-34** (3) (June 1986).
23. H. D. Cheng, H. S. Don and L. T. Kou, VLSI architecture for digital picture comparison, *IEEE Trans. Circuits Syst., Special Issue on VLSI Implementation for Digital Image and Video Processing Applications*, **36** (10) 1326–1335 (1989).
24. H. D. Cheng and C. Tong, Clustering analyzer, *IEEE Trans. Circuits and Systems*, **38** (1) 124–128 (1991).
25. H. D. Cheng, X. Q. Li, D. Riordan, J. N. Scrimger, A. Foyle and M. A. MacAulay, Parallel approach for tubule grading in breast cancer lesions, *Inform. Sci. An Int. J. Applic.* **4** (2) 119–141 (September 1995).
26. D. L. Hung, H. D. Cheng and S. Sengkhamyong, A reconfigurable hardware accelerator for moment computation', *IEEE Int. ASIC Conf.* (7–10 September (1997) accepted).

About the Author—HENG-DA CHENG received his Ph.D. in Electrical Engineering from Purdue University, West Lafayette, Indiana in 1985. Dr Cheng was a Visiting Assistant Professor, Electrical and Computer Engineering Department, University of California, Davis, an Assistant Professor, Computer Science Department, Concordia University, Montreal, and an Associate Professor, School of Computer Science, Technical University of Nova Scotia, Halifax, Nova Scotia. Now he is an Associate Professor,

Department of Computer Science, and an Adjunct Associate Professor, Department of Electrical Engineering, Utah State University, Logan, Utah. He has published over 150 technical papers and is the co-editor of the book, Pattern Recognition: Algorithms, Architectures and Applications (World Scientific, 1991). His research interests include parallel processing, parallel algorithms, artificial intelligence, image processing, pattern recognition, computer vision, fuzzy logic, genetic algorithms, neural networks and VLSI architectures. He was Program Co-Chairman of Vision Interface '90, Program Committee Member of Vision Interface '92 and Vision Interface '96, Session Chairs and Member of Best Paper Award Evaluation Committee of the International Joint Conference on Information Sciences, 1994 and 1995, Program Committee Member of the 1995 International Conference on Tools with Artificial Intelligence, and Program Committee Member of the 17th International Conference on Computer Processing of Oriental Languages, 1997. He is the chairman of the First International Workshop on Computer Vision, Pattern Recognition and Image Processing (CVPRIP'98), 1998. He serves as reviewer for many scientifical journals and conferences. He has been listed on *Who's Who in America*, *Who's Who in the World*, *Who's Who in Communications and Media*, *Who's Who in Finance and Industry*, *Who's who in Science and Engineering*, *Men of Achievement*, 2000 *Notable American Men*, *International Leaders in Achievement*, *Five Hundred Leaders of Influence*, *International Dictionary of Distinguished Leadership*, etc. He is appointed as a *Member of the International Biographical Center Advisory Council*, the International Biographical Center, England, and a *Member of the Board of Advisors*, the American Biographical Institute, USA. He is a Senior Member of the IEEE Socieity, and a Member of the Association of Computing Machinery. He is also an Associate Editor of *Pattern Recognition* and an Associate Editor of *Information Sciences.*

About the Author—CHEN-YUAN WU was born in Taipei, Taiwan, in 1965. He received his B. S. degree in Computer Science from Taunghai University, Taichung, Taiwan, in 1989, and the B.S. in Electrical Engineering from Utah State University, Logain, Utah, in 1996. He is currently working toward his master degree in Computer Science. His research interests include Pattern Recognition, Image Processing, Parallel Architecture, and Computer Vision.

About the Author—DONALD L. HUNG received his B. S. degree in electrical engineering from Tongji university in Shanghai, China, and his M. S. and Ph.D. degrees in systems and electrical engineering respectively, both from Case Western Reserve University in Cleveland, Ohio. His research interests include application-driven architectures and VLSI system design, high performance and adaptive computing machines. He joined the Department of Electrical Engineering at Gannon Univeristy, Erie, Pennsylvania as an assistant professor in 1990 and became an associate professor in 1995. Since August 1995 he has been with the School of Electrical Engineering and Computer Science, Wahsington State University, at the Tri-Cities campus.