Efficient Algorithms for the Hough Transform on Arrays with Reconfigurable Optical Buses*

Sandy Pavel, Selim G. Akl
Department of Computing and Information Science
Queen's University, Kingston, Ontario, K7L 3N6, CANADA

Abstract

This paper examines the possibility of implementing the Hough transform for line and circle detection on arrays with reconfigurable optical buses (AROBs). The AROB combines some of the advantages and characteristics of reconfigurable meshes and meshes with optical pipelined buses. It is shown that the Hough transform for line and circle detection in an $N \times N$ image can be implemented optimally in a constant number of steps. The costs of the two algorithms are $O(N^2 p)$ and $O(N^2 p^2)$, respectively, where p is the magnitude of one dimension in the parameter space. These values are optimal with respect to the time complexity of the best known sequential algorithms.

1 Introduction

The Hough Transform (HT) was introduced as a method of detecting the shape of object boundaries in binary images [12]. In order to achieve this, a number of parameters are associated to the patterns of interest. The main objective of the HT is to transform the spatially-extended patterns, represented by the edge pixels in the image, so that they produce spatially-compact features in the associated parameter space. The global shape detection problem is converted into a local, less difficult, peak detection problem.

Suppose that we have an $N \times N$ digitized image which contains a set of n *edge pixels*, $n \leq N^2$, obtained after some preprocessing (edge detection) step. A pixel can be specified by its coordinates (x, y), $0 \leq x, y \leq N-1$, in the image plane. Herein the origin of the system of coordinates $(0,0)$ is considered to be the bottom left corner of the image plane. Thus, the n edge pixels are represented by the set $\{(x_i, y_i) \mid 0 \leq i \leq n-1\}$ for which pixel (x_i, y_i) is "1". For the *line detection*, the edge pixel (x_i, y_i), $0 \leq i \leq n-1$, is transformed into a sinusoidal curve in the (ρ, θ)-parameter plane which is described by:

*This research was supported by the Telecommunications Research Institute of Ontario and the Natural Sciences and Engineering Research Council of Canada.

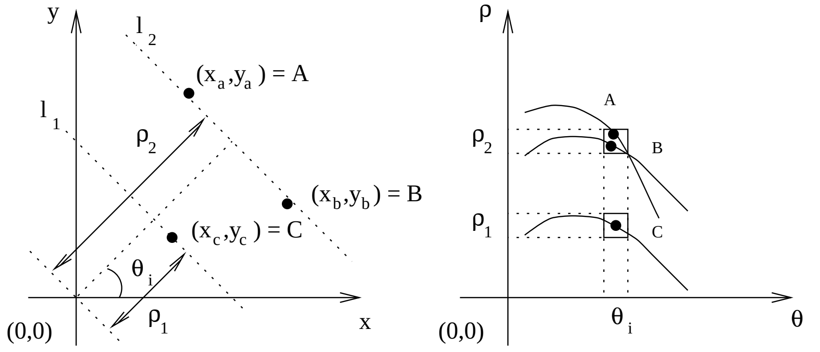

Figure 1: The Hough Transform. (a) The (ρ, θ) parameterization of the image space. (b) The (ρ, θ) parameter space.

$$\rho = \lfloor x_i \cos\theta + y_i \sin\theta \rfloor \tag{1}$$

The pair (ρ, θ) represents a parameterization of a line in the image space where ρ is the perpendicular distance from the origin to that line and θ is the slope of the perpendicular from origin to that line. Figure 1.a shows two *digitally collinear* pixels, A and B, i.e., pixels which are within some ρ_{res} distance from some line l_2. Parameter ρ_{res} is called the *resolution* of the problem. In the case presented in Fig. 1.a the two supporting lines l_1 and l_2 of the three pixels in the image space have the same angle θ_i and two different values ρ_1 and ρ_2 for the normal distances. Their possible contribution in the parameter space is represented in Fig. 1.b.

In the image space the normal distance ρ satisfies $\rho \leq \sqrt{2}N$. If p values are used to represent ρ, then $p \leq \sqrt{2}N$. The slope θ of the perpendicular from the origin is in the interval $[0, \pi]$. Due to the limited resolution and for simplicity the number of distinct values for the θ parameter is also considered to be p, with $p \leq \sqrt{2}N$. Thus, the size of the (ρ, θ)-space is $p \times p$. Assume that the values of parameters θ and ρ are $\theta_0, \theta_2, \ldots, \theta_{p-1}$ and $\rho_0, \rho_2, \ldots, \rho_{p-1}$ respectively. Associated with the parameter space is a $p \times p$ array, $count[\rho, \theta]$. For each edge pixel (x_i, y_i) and each θ_k the value of $\rho_{i,k}$ is computed using equation (1) and the value corresponding to the particular entry of $count[\rho_{i,k}, \theta_k]$ is incremented. Finally, a threshold function can be applied to all the elements of *count*. The time complexity of the sequential algorithm for computing the *count* array is $O(pn)$. This is proportional to the number of edge pixels and the number of quantized values along the θ direction. It is easy to verify that the circle detection algorithm has $O(p^2 n)$ time complexity, due to an increased parameter space (one more parameter), and in the case of ellipse detection the time complexity is $O(p^4 n)$. A detailed algorithm for circle detection is given in section 4.

So far we have assumed that the edge pixels are available and they are the only input data manipulated by these algorithms. Another possibility which occurs in practice is to have the entire $N \times N$ image as input for the algorithm. As $p = O(N)$, the three time complexities for line, circle and ellipse detection become $O(N^3)$, $O(N^4)$ and $O(N^6)$, respectively. The same orders of magnitude are obtained when the number of edge pixels $n = O(N^2)$. The detection of non-analytic curves of arbitrary shape, scale and orientation could take even longer. Thus, the amount of computation required by the HT is proportional to the number of the edge pixels n (or of the image pixels N^2) and also to the dimension of the parameter space. In general the resolution of the parameter space is determined by the accuracy sought and the resources available. As the amount of computation can become very large it is essential to be able to speed up the computation of the HT. Parallel processing was proposed as a possible solution. Parallel algorithms for line detection using the HT have been designed for different parallel architectures including tree [13], linear array [6], pyramid [15], mesh [4, 7, 17, 28, 32, 33] and hypercube [27].

A number of algorithms for line detection using the HT of an $N \times N$ image on reconfigurable networks have also been proposed in the literature. The algorithms given in [16, 23] have $O(p \log(N/p))$ time complexity and use an $N \times N$ reconfigurable mesh. In [16] the time complexity is improved to $O((p/N) \log N)$ and $O((p/N^{1/2}) \log N)$ by increasing the two-dimensional array to $N \times N^2$ and $N^{1.5} \times N^{1.5}$, respectively. An $O(p \log p + \log N)$ time algorithm for the HT is given in [1] where an $O(N^2)$-processor architecture, called Reconfigurable Multi-Ring Network, is used. When the input is reduced to the n edge pixels, constant time algorithms are obtained in [18] and [24] using reconfigurable arrays of dimensions $p \times n \times n \times n$ and $n \times \log^2 n \times p \times p$, respectively. As n can be in the order of $O(N^2)$, the two architectures could use $O(pN^6)$ and $O(p^2 N^2 \log^2 N)$ processors, respectively. In [19] three parallel HT algorithms are given for the circle detection. The best of these algorithms runs in $O(N^2)$ time and uses an $N \times N$ mesh architecture.

We propose two algorithms for line and circle detection using the Hough transform. Both algorithms take a constant number of steps and are implemented on the AROB model. The next section presents the AROB model together with some of the basic techniques implied by the use of optics for communication. In section 3 the line detection algorithm is described. The circle detection algorithm is presented in section 4. Section 5 concludes our paper.

2 The AROB Model

An important limitation of classical electrical buses concerns the exclusive access to the bus resources which limits throughput to a function of the end-to-end propagation time. Other fundamental constraints which bound bus interconnections in electronic systems in general are the limited bandwidth, capacitive loading, and cross-talk caused by mutual inductance. Optical communications have emerged as alternative solutions to these problems. Unlike the signal propagation in electronic buses, which is bidirectional, optical channels are inherently directional and have predictable delay per unit length. As it is shown in [21, 8], this allows a pipeline of signals to be created by the synchronized directional coupling of each signal at specific locations along the channel. The possibility in optics to pipeline the transmission of signals through a channel provides an alternative to exclusive bus access. Using this kind of spatial parallelism the end-to-end propagation latency

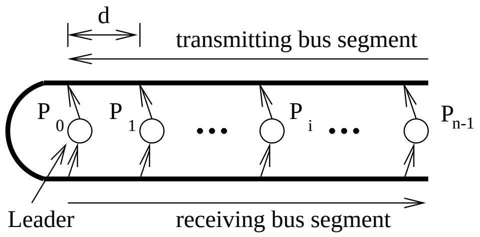

Figure 2: The linear array with pipelined optical buses APPB.

can be amortized over the number of parallel messages active at the same time on the bus.

Linear arrays with optical pipelined buses

Consider a linear array of n processors connected to an optical bus as in Fig. 2. Each processor is connected to the bus through two directional couplers. One is used to write data on the bus and the other to read the data from the bus. Each processor sends data on the upper (*transmitting*) segment of the bus and reads messages from the lower (*receiving*) segment of the bus. The optical bus is constructed from three identical waveguides. One waveguide is used for message transmission, the *message* waveguide, and two are used for carrying address related information, the *reference* (Ref) and *select* (Sel) waveguides. During a write cycle, the data written by a processor into the bus propagate as indicated with arrows in Fig. 2, and may be read by any subsequent node on the bus. Due to the directionality of the signal propagation and the predictable delay of the signal, the same bus may be used to transmit messages between other nodes in the same time. Let us consider that each message is b bits long. Each bit is represented by an optical signal of width w seconds for a binary value of 1. The absence of such a signal represents a 0.

Two conditions are essential for this kind of transmission: (1) all transmissions are synchronized and (2) the length of the optical path on the waveguide between any two adjacent nodes, d in Fig. 2, is larger than or equal to bwc_g, where c_g is the velocity of light in the waveguide. These two conditions ensure that the signals corresponding to two different messages, which travel in a waveguide in the same direction at the same time, do not physically overlap at any point on the waveguide, i.e., are space multiplexed. The *end-to-end propagation time* is $\sigma_b = 2n\tau$ seconds, where n is the number of processors in the linear array and τ, is the time taken for a message to traverse the optical distance d. This system is called linear *Array Processors with Pipelined Buses* (APPB), and its principles have been introduced in [21, 8].

Several approaches can be used to route messages in a linear APPB structure from one processor to another. In [21, 8] a *time waiting function* is introduced. Various message routing patterns can be realized using this function, most of which are suitable for SIMD and/or MIMD implementations. In [29] a time-division multiplexing scheme is given for communications in arrays with optical pipelined buses. This is used in [25] to implement a part of the synchronous communications needed by the AROB. Two cases are considered: (1) message receiving, when the destination processor knows the index of the sender and (2) message sending, when the sender knows the destination address. It is shown that these communication schemes can be implemented in a constant number of bus cycles, [25]. Arbitrary permutations can be performed using these message transmitting techniques.

The address information can also be encoded using the *coincident pulse technique* presented in [3, 20, 30, 29]. This technique can be used when the source processor knows the address (index) of the destination processor. For example, a processor can use the coincident pulse addressing mechanism in order to broadcast a value to all other processors [24]. Many other communication patterns are described in [3, 20, 30] using the same coincident pulse technique. The coincident pulse mechanism seems to avoid the traditional bottleneck which characterize most of the on-line address decoding techniques [29].

A *bus cycle length* is given by the time taken by the end-to-end propagation of the messages together with the time required to process a message at the source and destination processors. The latter operation includes message generation and detection, synchronization, buffering, etc. [8]. The message processing is done in parallel by all source processors, at the beginning of the bus cycle, and/or by all destination processors after the message propagation ends. The ratio, R, between the message processing time and the communication over distance d time is estimated to be on the order of 10 to 1000, see [8]. In order for the message processing time to be of the same order of magnitude as the end-to-end propagation time σ_b, the number of processors should satisfy $n \geq R$. In this case the bus cycle length can be considered to be $O(\sigma_b)$.

There are different factors that limit the size of an optical bus. All passive devices, as the passive couplers or optical fibers, exhibit loss. In the same time each coupler, used by a processor to read a message from the waveguide, taps a percentage of the energy carried by the signal. One solution for these problems is to provide some means of amplitude restoration. The use of an active switch instead of the directional coupler could solve this problem, [11], but it has the main disadvantage that introduces more delay. Otherwise, the number of the devices connected to a bus must be restricted to an admissible value, i.e., which does not introduce errors. This second solution reduces also the value of the end-to-end propagation delay. Two-dimensional arrays with optical pipelined buses have been proposed in order to reduce the number of processors connected to a linear optical pipelined bus from $O(n)$ to $O(\sqrt{n})$, [8, 29, 31, 10].

The two-dimensional AROB

The array with reconfigurable optical buses we propose uses the basic architectural and functional structure of a classical reconfigurable network, RN, [2]. The communication system of this network is modified in order to allow the implementation of the pipelined optical communication mechanisms as they are introduced in [21, 8] and also presented in the previous subsection.

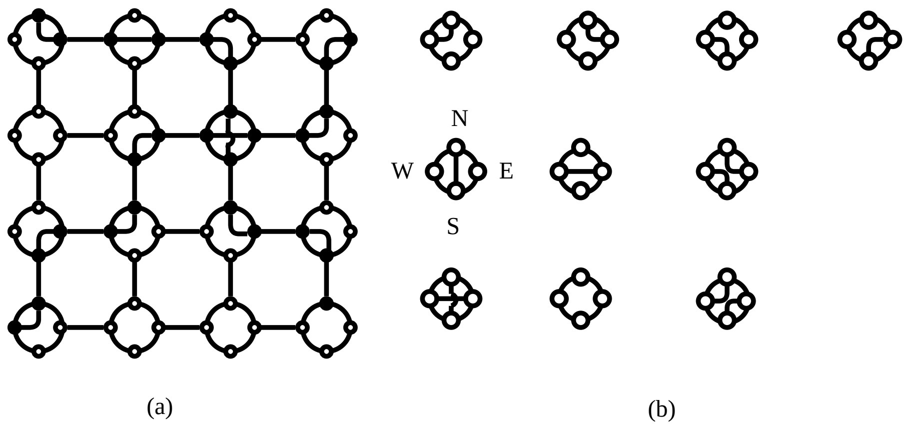

Figure 3: (a) Two-dimensional AROB; (b) Switch configurations.

Consider n^2 processors connected in a two-dimensional reconfigurable structure as depicted in Fig. 3.a. The switching system of a processor is able to connect the four ports (N, S, E, W) in one of the configurations shown in Fig. 3.b. The interconnection network is assumed to use optical waveguides and optical switches for communication and reconfiguration, respectively. We call this structure *Array Processors with Reconfigurable Optical Buses*, AROB. The functional and architectural characteristics of the AROB are described in [25].

It is shown in [25] that the local switch setting provides a mechanism for building arbitrary APPB-like linear structures on the AROB. A processor can be connected to up to four such buses at a time. When pipelined communications are sought, an algorithm to be implemented on the AROB is required to specify a rule in order to uniquely identify the *leader* processor of each linear array obtained by reconfiguration (similar to P_0 in Fig. 2). The algorithms to be simulated on the AROB are not necessarily required to specify to each processor the length of the linear buses to which it is connected or its indices relative to these buses.. A technique to determine the relative position of each processor in each linear bus to which it is connected, and implicitly the length of each bus, is given in [25]. Note that these values are needed in order to be able to implement the time division or coincident pulse techniques

Let a *step* be either an internal operation or a communication step. It has been argued that counting the number of steps of an algorithm simulated on an array with optical pipelined buses allows for a better comparison between algorithms, [9]. This method has the advantage that it abstracts from the details introduced by the technology dependent parameters, as σ_b for example. Obviously this is true only if the same assumptions regarding the bus cycle lengths are made. The assumption made in [21] is that the bus cycle length in a linear APPB is compatible with the computation speed in the nodes. In [8, 10] and [24] the time complexity of different algorithms simulated on arrays with optical pipelined buses is also expressed in terms of the number of steps and it is shown that these results compare favorable with those obtained on different classical networks.

We adopt in this paper the same method of specifying the time complexity in number of steps. However, one must be aware that for a more accurate comparison of our results with those obtained on other architectures, as the RN or the mesh with global broadcasting buses for example, would imply the explicit use of the bus cycle length parameter, $O(\sigma_b)$. Obviously, the communication delays encountered in the actual implementations of the other architectures should also be accounted for.

It is shown in [25] that an AROB with n^2 processors can simulate any n^2-processor RN operation in a constant number of steps. If the propagation times on buses of the same length in the two systems are identical, than the simulation is time and cost optimal. For the RN simulation using the AROB model, the algorithms are no longer required to specify the *leader* of each broadcasting bus because no time division communications are needed. It is important to emphasize the differences between RN and AROB. By contrast with the classical RN, the AROB structure can use pipelining in order to allow multiple broadcasting of different messages on the same arbitrary bus, during the same bus cycle. The internal time-related operation and the coincident pulse techniques, represent other important enhancements over the classical RN. The relative powers of the AROB and RN are more formally presented in [25].

The description of 2D AROBs can be extended to higher dimensions $k > 2$. For example a 3D AROB processor has two extra ports, U and D, for the communications with the neighbor processors in the third dimension. The switch configurations allow any two pairs of these 6 ports to be connected internally, with any port connected to only one of the other ports in the same time.

Binary Prefix Sums on a linear APPB

Given n one-bit values, v_i, $0 \leq i \leq n-1$, the *binary prefix sums* (BPS) problem requires the computation of $ps_i = v_0 + v_2 + \ldots + v_i$, for all $0 \leq i \leq n-1$, where "+" is the addition operation. Consider the linear APPB structure with n processors, each storing one bit of the input sequence. It is shown in [25, 26] that:

Claim 1 *The n-values BPS problem can be solved on a linear APPB with n processors in a constant number of steps.* ■

Note that the BPS of n binary values can be computed in a constant number of steps on a $\sqrt{n} \times n$ [22], or on a $\log^2 n \times n$ RN [14]. If we assume the same propagation times for the signals in the two models, RN and linear APPB, the BPS takes the same time but with considerably fewer processors on the linear APPB model. Delay lines and the coincident pulse technique are also used in [9] to compute the *binary sum* of n binary values on a linear array with pipelined buses enhanced with switching capabilities. We use in the next section the BPS algorithm only to obtain the binary sum of the n bits. This technique, as opposed to that in [9], has the advantage that it can determine the binary sum of the one-bit values stored by the processors connected to an arbitrary bus.

3 The HT for line detection on AROB

Assume that we have an $N \times N \times p$ AROB. Assume also that we have the preprocessed (digitized) $N \times N$ image as the input data, i.e., each edge (feature) pixel (i, j), for i and j in $\{0, \ldots, N-1\}$, has a value of "1" and it is stored in the base array by processor $P_{i,j,0}$. All other background pixels are "0" and are also stored in the base array, one per processor.

We associate a sub-array of processors $P_{*,*,k}$ with each value of the angle parameter θ_k, $0 \leq k \leq p-1$. The symbol "$*$" is used in order to simplify the notation and represents all the values of the index it replaces in the notation. In the first step of the algorithm each processor $P_{i,j,0}$ in the base plane broadcasts pixel (i, j) on column $P_{i,j,*}$. Thus each sub-array $P_{*,*,k}$, $0 \leq k \leq p-1$, gets a copy of the input image. It is assumed also that each processor of the kth sub-array $P_{*,*,k}$ knows the value of θ_k. Usually the angles are chosen such that for all k, $\theta_k = (\pi k)/p$, thus each processor can compute in $O(1)$ time the value of θ_k.

The next steps of the algorithm are performed independently and in parallel by all arrays $P_{*,*,k}$, $0 \leq k \leq p-1$. For each pixel (i, j), $P_{i,j,k}$ computes $\rho_{i,j} = \lfloor i \cos\theta_k + j \sin\theta_k \rfloor$. All the edge pixels which have the same value $\rho_{i,j} = \rho$ describe a potential line in the image space. This line is represented by the (ρ, θ_k) pair in the parameter space. Thus, in order to compute the value of the $count[\rho, \theta_k]$ accumulator we need to count the number of edge pixels which have associated the same (ρ, θ_k) pair. To simplify the presentation we restrict the angle values in the range $[0, \pi/4]$. By using the symmetry properties, all the other values of θ_k, $\pi/4 < \theta_k \leq \pi$, can be treated similarly.

In what follows we show how the values of $count[\rho, \theta_k]$ can be obtained in a constant number of steps. First we show that all the processors in a plane $P_{*,*,k}$ which have computed and stored the same values for ρ can be connected in a linear APPB-like structure. In order to do this we rely on a number of results previously obtained in [23]. The following observations are made for all k such that $0 \leq \theta_k \leq \pi/4$:

1. For any j, with $0 \leq j \leq N-1$, the ρ-distances satisfy $\rho_{i,j} \leq \rho_{i+1,j}$ for $0 \leq i \leq N-2$. It also can be shown that no more than two consecutive values of ρ in row j can be equal.

2. The values of ρ, computed using equation (1) by two consecutive processors in row j, differ by at most 1. More formally, for all i, j, $0 \leq j \leq N-1$ and $0 < i \leq N-2$, $0 \leq \rho_{i+1,j} - \rho_{i,j} \leq 1$.

3. For all values of i, j, $0 \leq i, j \leq N-2$, $\rho_{i,j} \neq \rho_{i+1,j+1}$.

4. If $\rho_{i,j} = \rho_{i,j+2}$ for $0 \leq i \leq N-1$ and $0 \leq j \leq N-3$, then $\rho_{i,j} = \rho_{i,j+1} = \rho_{i,j+2}$. If two ρ-values in a column i are equal and they are placed two rows apart then the ρ-value in between should have the same value. $\square$

Similar properties can be obtained for the other values of the θ_k. An immediate consequence of these observations is that all processors which have computed the same value of the normal distance ρ, for some angle

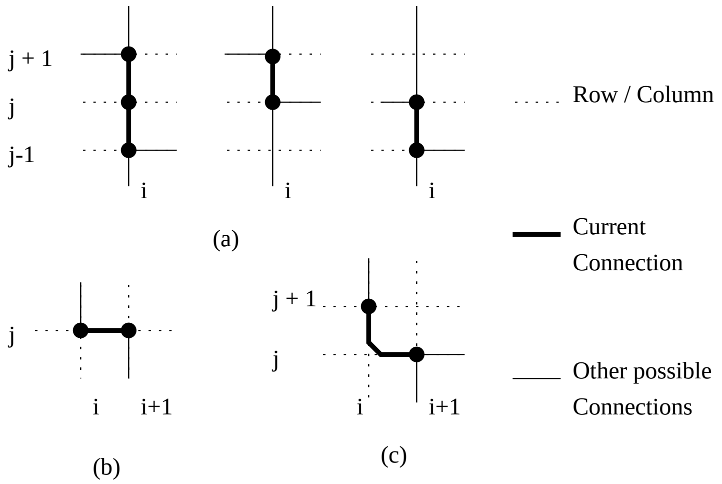

Figure 4: Possible connections of a processor to the bus associated to ρ, for $\theta_k \in [0, \pi/4]$.

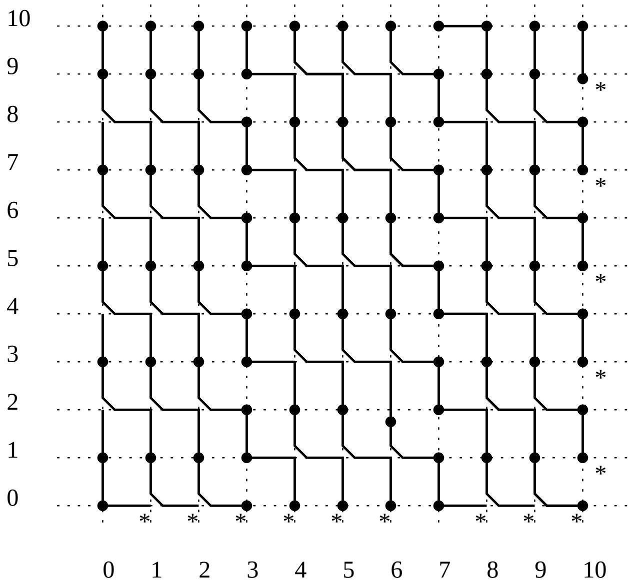

Figure 5: Mesh configuration for $\theta_k = \pi/6$.

θ_k, can be connected in a linear APPB-like structure. This operation requires only local communications, each processor changing information with only some of its four neighbors. More specifically, the possible cases are depicted in Fig. 4. If $P_{i,j,k}$ computes some value ρ for the normal distance associated to pixel (i,j), and the same value is obtained for pixels $(i,j-1)$ and/or $(i,j+1)$, then processors $P_{i,j,k}$ should be connected, using the links on the ith column, to $P_{i,j-1,k}$ and/or $P_{i,j+1,k}$, Fig. 4.a. When in a row j two adjacent processors have the same ρ value the connection can be made as shown in Fig. 4.b. In case processors $P_{i,j+1,k}$ and $P_{i+1,j,k}$ have to be connected, a third intermediate processor $P_{i,j,k}$ is used, Fig. 4.c. If the above rules are used a processor is connected to at most two buses at a time. No two distinct buses are connected to the same port of a processor, the actual setting of a switching system being in one of the states depicted in Fig. 3.b. One end of each bus is located in the bottom row or the right-most column of the array. A possible bus configuration is depicted in Fig. 5 for $\theta_k = \pi/6$. This is designated as the *leader* processor of the linear structure and is marked with a star in Fig. 5. The leader selection requires only a constant number of computations and/or local communication.

Next we apply the BPS in each linear bus constructed this way, and the values of $count[\rho, \theta_k]$ are obtained and stored by each leader. The last step is the thresholding operation which allows us to keep only those values of the *count* array which are larger than some given threshold value. This process is done in parallel by $P_{*,*,k}$, for all values of θ_k, $0 \leq k \leq p-1$. Thus,

Claim 2 *The Hough transform for line detection in an $N \times N$ image can be implemented in a constant number of steps on an $N \times N \times p$ AROB.* ∎

The cost of this algorithm is $O(N^2 p)$. This is optimal with respect to the time complexity of the best known sequential algorithm, when the $N \times N$ image is given as input (or when $n = O(N^2)$). As $p = O(N)$ the total number of processors is in the order of $O(N^3)$. Thus the cost of this algorithm becomes $O(N^3)$.

4 An HT algorithm for circle detection on AROB

Circle detection, like line detection, plays an important role in many computer vision applications. This section addresses the problem of circle detection using the AROB model.

The HT for circles maps each edge point in the input image to a cone in the three-dimensional Hough (parameter) space. A circle is given by the equation:

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{2}$$

where (x_0, y_0) is the center of the circle and r is its radius. In the case of circle detection one more parameter is added, as compared to the line detection, and the parameter space can be described by (x_0, y_0, r). Assume that we have as input the image already preprocessed, i.e., each edge pixel is marked by a "1". An edge pixel in the image can be part of many different circles in the image, and its contribution in the parameter space should affect all the (x_0, y_0, r) associated entries. All the edge pixels of the same circle in the image will of course

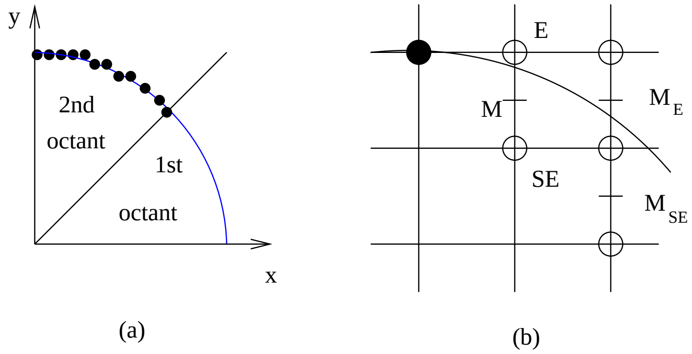

Figure 6: (a) The first quadrant of a circle. (b) The pixel grid.

contribute to the entry which represents the common circle. As in the case of line detection, circle detection is transformed to a more local task, namely, the peak detection in the parameter space.

These operations require the quantization of the parameter space. In the case of line detection both parameters are bounded for a given $N \times N$ image. The slope θ of the perpendicular from the origin is in the interval $[0, \pi]$ and the perpendicular distance ρ from the origin lies in the interval $[0, \sqrt{2}N]$. In the case of circle, the parameters can be unbounded. The center of the circle can lie outside the image and in the same time a part of the circle can be visible in the image. This implies that the radius can also be much larger than $\sqrt{2}N$. For practical reasons we limit the center of the circle to lie within the image boundaries, i.e., x_0 and y_0 take values in the interval $[0, N-1]$. These constraints are satisfactory for most practical applications [19]. As a consequence any circle satisfying the above requirement has an integer radius of up to $[\sqrt{2}N]$. Therefore, each parameter is assumed to be quantized in p values with $p = O(N)$.

The HT algorithm for circle detection is similar to that for the line. Firstly, the accumulator *count* associated to the parameter space is initialized to zero. Then, for each edge pixel (x, y) in the image, all the parameter entries $count[x_0, y_0, r]$ for which equation (2) is satisfied are incremented. In the end, the number stored by an entry $count[x_0, y_0, r]$ represents the number of pixels lying on a circle of radius r and center (x_0, y_0). The overall time complexity of this sequential algorithm is $O(N^4)$. In [19] three parallel algorithms are given for the circle detection. The best of these algorithms runs in $O(N^2)$ time and uses an $N \times N$ mesh architecture. All circles of a given radius are traced in parallel, updating in the same time the parameter accumulator associated to each of these circles. The tracing technique is actually the *midpoint circle scan-conversion algorithm*, given in [5], which is used in graphics to draw circles given in an analytic form. We give next a short description of this algorithm which serves also as an intuitive introduction to our parallel algorithm for circle detection.

Consider the arc of circle depicted in Fig. 6.a. We describe the technique presented in [5, 19] for the sequential identification of the pixels lying only on the second octant of its associated circle. All other octants can be processed similarly. The idea behind this tracing technique is sketched in Fig. 6.b. The shaded pixel is the initial starting point and is the top-most pixel of the circle. The pixels lying to the "east" and "south east" of the top-most pixel are marked with E and SE. M is the midpoint of the line segment joining E and SE. Suppose that the circle is described by equation (2) and that the current pixel in the tracing sequence is the top-most one. The next pixel in the clockwise scan of the circle is either E or SE. The choice between the two can be made by determining the position of M relative to the circle. Clearly the error implied by this approximation is minimized if the next pixel is chosen to be the closest to the real circle. If M lies inside (or on) the circle then pixel E is (chosen to be) closer to the circle and if M lies outside the circle, pixel SE is closer to the real circle. The last pixel becomes the current pixel, and the above process is repeated for the entire arc in the second octant. Some modifications are necessary for the other octants [5]. This technique can be used to trace each circle in the image space. Counting the number of edge pixels encountered in this scan-conversion of a circle gives the value of an entry in the accumulator associated to the Hough space.

Assume that we have an $N \times N$ image stored on an $N \times N$ AROB, one pixel per processor. The idea behind our algorithm for circle detection is somehow similar to the one presented above and uses the same techniques employed with the line detection. The first objective is to connect to the same bus all the processors which lie on the same scan-converted circle. In the next step a simple binary prefix sums algorithm can be used in order to determine the number of edge pixels associated to that circle.

As the maximum number of circle buses to which a processor can be connected at a time is two, we are no longer able to trace all circles which have the same radius in parallel as in [19]. This would require more than two buses to pass through some processors. A better solution is to trace all the circles with the same center, say (x_0, y_0), and different radii in parallel. For simplicity and without loss of generality, the center of the coordinate system is considered to be translated to the current (x_0, y_0) center, i.e., $(x_0, y_0) = (0, 0)$, a circle being described in this case by $r^2 = x^2 + y^2$. We show next that the buses associated to different circles with different radii, r, and the same center (x_0, y_0), can be constructed in parallel with no conflict.

Consider a generic pixel (x, y) in the second octant with $x \geq 1$, $y \geq 1$, and its associated processor $P_{x,y}$. A natural question is how to determine whether there is any circle with center $(x_0, y_0) = (0, 0)$ and some integer radius r, $r \geq 1$, for which (x, y) is part of that scan-converted circle?

Claim 3 *Let (x, y) be a pixel with $x \geq 1$ and $y \geq 1$.*

1. *There is a circle with center $(x_0, y_0) = (0, 0)$ and integer radius r, $r \geq 1$, such that (x, y) is a point in that circle's scan-converted representation if and only if*

$$z_1 \leq r \leq z_2 \tag{3}$$

   where $z_1 = \lceil ((y - 0.5)^2 + x^2)^{1/2} \rceil$ and $z_2 = \lfloor ((y + 0.5)^2 + x^2)^{1/2} \rfloor$.

2. *If such an r exists then it is unique and $r = z_1 = z_2$.*

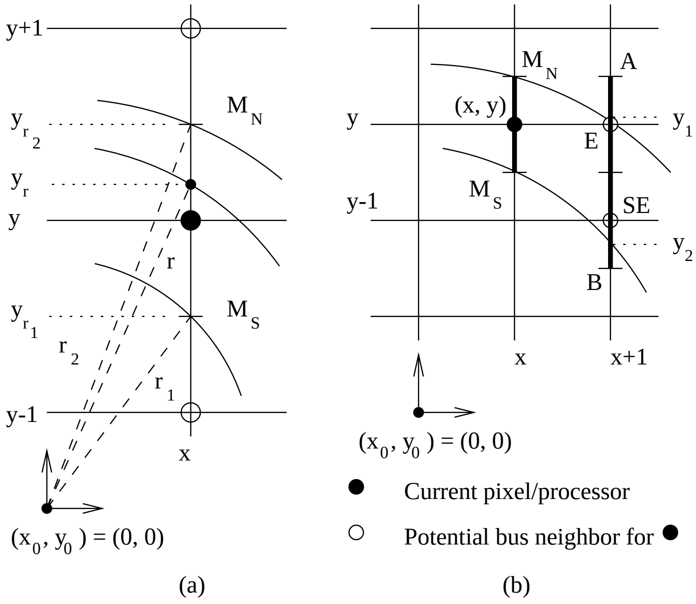

Figure 7: (a) Illustration for proof of Claim 3. (b) Illustration for proof of Claim 4.

Proof : Using Fig. 7.a the proof of statement (1) is immediate. Let M_N be the point at the middle vertical distance between (x, y) and $(x, y+1)$ and M_S the point at the middle vertical distance between (x, y) and $(x, y-1)$. There is a circle with integer radius r and center $(0, 0)$ such that (x, y) is a point in its scan-converted representation if and only if that circle intersects the vertical which passes through (x, y) somewhere between M_N and M_S. This condition can be written as $y_{r_1} \leq y_r < y_{r_2}$, where y_{r_1} and y_{r_2} are real values and are the radii of the two circles, with centers in $(0, 0)$, which pass through M_S and M_N, respectively. Since $y_{r_1} = y - 0.5$ and $y_{r_2} = y + 0.5$ one can verify that $y_{r_1} \leq y_r < y_{r_2}$ is equivalent to $((y-0.5)^2 + x^2)^{1/2} \leq r < ((y+0.5)^2 + x^2)^{1/2}$. As r must be an integer, this expression becomes $z_1 \leq r \leq z_2$.

Statement (2) can be proved by contradiction, assuming that there are two circles of integer radii r and $r+1$ such that the circles intersect the vertical in x between M_N and M_S. Let y_r and y_{r+1} be the two points of intersection in the second octant between these circles and the vertical in x. The distance between y_r and y_{r+1} is strictly greater than 1 and $y_{r_1} \leq y_r < y_{r+1} \leq y_{r_2}$. This means that the distance between y_{r_1} and y_{r_2} is strictly greater than 1, which is impossible. Thus, if such a circle exists for pixel (x, y) then it is unique. ∎

Examples : Let $x = 2$ and $y = 4$. Using equation (3), $\lceil 4.03 \rceil \leq r \leq \lfloor 5.76 \rfloor$ which is equivalent to $5 \leq r \leq 5$. For $x = 9$ and $y = 10$ we obtain $14 \leq r \leq 13$ and thus there is no such r. See Fig. 9. □

With the results in Claim 3 we have established a technique which can be used by each processor in the

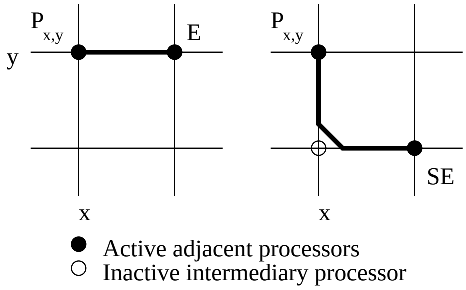

Figure 8: (a) The two possible connections between two neighbors on a scan-converted circle. (b) The internal switch settings used in the second octant.

array to determine if its associated pixel is part of any scan-converted circle with center (x_0, y_0) and integer valued radius. Furthermore, if such a circle exists then it is unique and its radius can be determined in constant time. The entire test takes constant time.

Each processor which has determined a valid value for r must connect itself to other processors with the same r in order to establish the linear bus associated to that circle, if such processor exists. Again, we restrict our discussion to the second octant, $\pi/4 \leq \theta_k \leq \pi/2$. We describe a processor $P_{x,y}$ as *active* if it is connected to the bus associated to the corresponding circle and in the same time its pixel is part of the scan-converted circle. As we explain later, it is also possible for a processor to be connected to a bus without the corresponding pixel being part of the associated circle. This is called *inactive* processor.

Claim 4 *Let (x, y) be a pixel which satisfies equation (3), i.e., there is a circle with center $(x_0, y_0) = (0, 0)$ and integer radius r such that (x, y) is part of that circle scan-converted representation. Then, either $P_{x+1,y}$ or $P_{x+1,y-1}$ is the next active processor adjacent to $P_{x,y}$ in the clockwise scan-converted order on that circle.*

Proof : For a proof we can use Fig. 7.b. The two possible extreme cases, when (x, y) is part of some scan-converted circle, are represented by the two arcs of circle which intersect the vertical passing through (x, y) in M_N and M_S. One can easily verify that these circles intersect the vertical in $x + 1$ somewhere between point A and point B, Fig. 7.b. Thus, either processor E $(P_{x+1,y})$ or SE $(P_{x+1,y-1})$ are the active neighbors of $P_{x,y}$. ■

The two possible connections are depicted in Fig. 8.a. To establish the bus connection between $P_{x,y}$ and the E neighbor the direct row link is used. To connect $P_{x,y}$ with SE $P_{x,y-1}$ is used as an intermediate, or *inactive*, processor (by convention). Here, the processor is *inactive* as the data stored by it is irrelevant for this circle, and the processor is used only to establish a connection between $P_{x,y}$ and SE. At the same time, by reciprocity, $P_{x,y}$ must be the neighbor of either $P_{x-1,y}$ or $P_{x-1,y+1}$. The four possible internal switch connections for the second octant implied by these rules are depicted in Fig. 8.b. The worst case scenario is given by the bottom-left configuration where a processor can be at the same time active and inactive but relative to two different buses. Note that no conflict arises in this case, making it possible to construct all buses in parallel. An example of

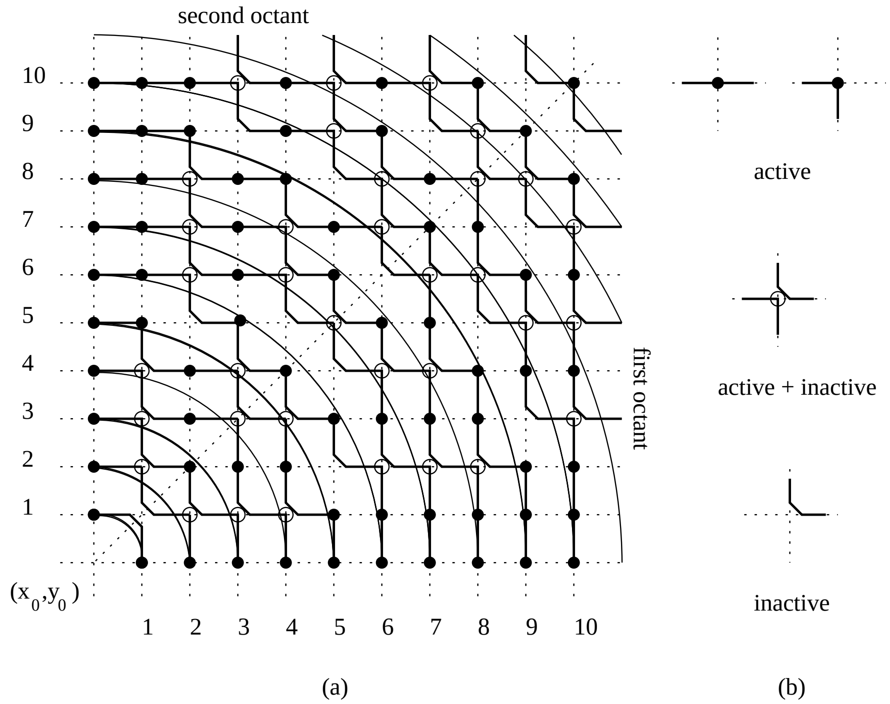

Figure 9: (a) The bus connections in the first quadrant. (b) Processor connections in the second octant.

parallel bus construction for the first quadrant is depicted in Fig. 9.a. The different processor connections for the second octant are shown in Fig. 9.b. The only exception from these rules is made for the interconnection of the four processors around P_{x_0,y_0}, i.e., P_{x_0+1,y_0}, P_{x_0,y_0-1}, P_{x_0-1,y_0} and P_{x_0,y_0+1}. For this interconnection P_{x_0-1,y_0-1}, P_{x_0-1,y_0+1}, P_{x_0+1,y_0+1} and P_{x_0+1,y_0-1} are used as intermediate, inactive processors.

A bus can correspond to either a full scan-converted circle or to an arc of a scan-converted circle. Examples of circles with center (x_0, y_0) which have at least one arc contained in the image are depicted in Fig. 10.a . The square represents the border of the $N \times N$ image. In order to apply the binary prefix sum algorithm a *leader* processor must be specified for each constructed arc or circle bus. The leader processors are represented with shaded circles in Fig. 10. There are two different cases. In the first case the entire scan-converted circle with center (x_0, y_0) and radius r is contained in the image. Thus, all associated processors are connected to a linear bus and the top-most processor, i.e. P_{x_0,y_0+r}, is designated as the leader of this bus. All these leaders are grouped in the rounded box marked with "1" in Fig. 10.b. Note that as r has an integer value, the P_{x_0,y_0+r} processor always exists when the entire circle is contained in the image. In the second case, a scan-converted circle is represent by up to four arcs, the first processor in the clockwise order of each arc can be set as the leader. The other four rounded boxes in Fig. 10.b contain the leaders associated to these arcs. The process of leader detection uses only local computations and can be done in a constant number of steps.

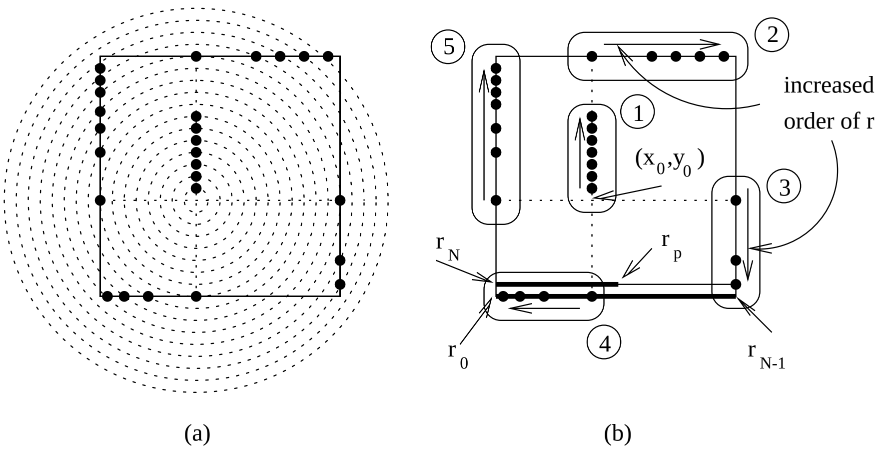

Figure 10: (a) Relative positions of circles within an image. (b) The leader processors and their grouping.

The partial values of the $count[x_0, y_0, r]$ accumulators are obtained by applying the BPS algorithm in each linear bus associated to a circle or arc of circle. Each active processor participates with its pixel value in this BPS. The inactive processors use the 0 value. The leader stores the accumulator value together with the radius value r. In each of the five groups in Fig. 10.b these values are already sorted over r, which are also distinct. The increasing order of r is shown in Fig. 10.b using arrows in each rounded box. The objective is to group all the accumulator values in the bottom-most rows of the array in the sorted order of r. As $p \leq \sqrt{2}N$, at most two such rows are used. A possible final order of the p values of r is suggested in Fig. 10.b. As no two values within each rounded box have the same destination, simple permutation routing can be used to move data to the two bottom rows. The five rounded boxes are treated sequentially because contention can occur when different partial accumulators, associated to the same entry $count[x_0, y_0, r]$, are routed to the same processor. This process is done in parallel for all values of (x_0, y_0). We therefore have:

Claim 5 *The Hough transform for circle detection in an $N \times N$ images can be implemented in a constant number of steps on an $N \times N \times p^2$ AROB.* ■

The cost of this algorithm is $O(N^2p^2)$. This is optimal with respect to the time complexity of the best known sequential algorithm.

5 Conclusions

Constant time parallel HT algorithm for line detection are given in the literature for the case where the input is given as n edge pixels. The best constant time algorithm is obtained in [24] using a reconfigurable array of dimensions $n \times \log^2 n \times p \times p$. As n can be $O(N^2)$ and $p = O(N)$, the actual number of processors is

$O(N^4 \log^2 N)$. The best algorithm for parallel (HT) circle detections runs in $O(N^2)$ time and uses an $N \times N$ mesh architecture, [19].

In this paper it is shown that the Hough transform can be implemented optimally in $O(1)$ steps. The costs of the two algorithms for line and circle detection are $O(N^3)$ and $O(N^4)$, respectively. These values are optimal with respect to the complexity of the best sequential algorithms and compare favorably to the best known parallel algorithms.

The number of processors can be further reduced but the price which must be paid is the increased time complexity of the algorithms. The line detection algorithm can be implemented in $O(t)$ time on an $N \times N \times \frac{p}{t}$ AROB with $t \leq p$. The circle algorithm can take $O(t)$ time on an $N \times N \times \frac{p^2}{t}$ AROB, $t \leq p^2$.

References

[1] Bhandarkar, S., Arabnia, H.R. The Hough transform on a reconfigurable multi-ring network, *Journal of Parallel and Distributed Computing*, **24**, 1995, 107-114.

[2] Ben-Asher, Y., Peleg, D., Ramaswami, R., Schuster, A.The power of reconfiguration, *Journal of Parallel and Distributed Computing*, vol. 13, 1991, 139-153.

[3] Chiarulli, D. M., Melhem, R.G., and Levitan, S. P. Using coincident optical pulses for parallel memory addressing. *The Computer Journal*. Dec. 1987.

[4] Cypher, R.E., Sanz, J.L., Snyder, L., The Hough Transform has $O(N)$ complexity in SIMD N by N mesh array architectures, *Proceedings IEEE Computer Society Workshop on Computer Architectures for Pattern Analysis and Machine Intelligence*, 1987, 115-121.

[5] Foley, J.D., van Dam, A., Feiner, S.K., Hughes, J.F. *Computer Graphics: Priciples and Practice*, Addison-Wesley Publishing Company, Inc., 1990.

[6] Fisher, A.L., Highnam, P.T., Computing the HOugh transform on a scan line array processor, *IEEE Transactions on PAMI*, vol. 11, no. 3, 1989, 262-265.

[7] Guerra, C., Hambrusch, S., Parallel algorithms for line detection on a mesh, *Journal of Parallel and Distributed Computing*, vol. 6, 1989, 1-19.

[8] Guo, Z., Melhem, R. M., Hall, R. W., Chiarulli, D. M., and Levitan, S. P. Pipelined communications in optically interconnected arrays. *Journal of Parallel and Distributed Computing*. **12**, 1991, 269-282.

[9] Guo, Z. Sorting on array processors with pipelined buses. *Proceedings 1992 International Conference on Parallel Processing*. 1992, III 289-292.

[10] Guo, Z. Optically Interconnected Processor Arrays with Switching Capability. *Journal of Parallel and Distributed Computing.* **23**, 1994, 314-329.

[11] Heuring, V. P., Jordan, H. F., and Pratt, J. P. Bit-serial architecture for optical computing. *Applied Optics.* **31**(17), 1992, 3213-3224.

[12] Illingworth, J., Kittler, J. A survey of hough transform, *Computer Vision, Graphics, and Image Processing*, **44**, 1988, 87-116.

[13] Ibrahim, H.A.H., Kender, J.R., Shaw D.E., The analysis and performance of two middle-level vision tasks on a fine-grained SIMD tree machine, *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1985, 387-393.

[14] Jang, J.-W., Park, H., Prasanna, V.K. A fast algorithm for computing a histogram on reconfigurable mesh, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **17**(2), 1995, 97-106.

[15] Jolion, J., Rosenfeld, A., An $O(\log n)$ pyramid Hough transform, *Pattern Recognition Letters*, vol. 9, 1989, 343-349.

[16] Jenq, J.F., Sahni, S. Reconfigurable mesh algorithms for the Hough transform, *Journal of Parallel and Distributed Computing*, vol. 20, 1994, 69-77.

[17] Kannan, C.S., Chuang, H.Y.H., Fast Hough transform on a mesh connected processor array, *Information Processing Letters*, 33, 1990, 243-248.

[18] Kao, T.-W., Horng, S.-J., Wang, Y.-L., Chung, K.-L., A constant time algorithm for computing Hough transform, *Pattern Recognition*, **26**(2), 1993, 277-286.

[19] Kumar, S., Ranganathan, N., Goldof, D., Parallel algorithms for circle detection in images, *Pattern Recognition*, **27**(8), 1994, 1019-1028.

[20] Levitan, S. P., Chiarulli, D. M., and Melhem, R. G. Coincident pulse technique for multiprocessor interconnection structures. *Applied Optics.* **29**(14), 1990, 2024-2033.

[21] Melhem, R. G., Chiarulli, D. M., and Levitan S. P. Space multiplexing of waveguides in optically interconnected multiprocessor systems. *Computer Journal.* **32**(4), 1989, 362-369.

[22] Olariu, S., Schwing, J. L., and Zhang, J. Integer problems on reconfigurable meshes, with application. *Proceedings 1991 Allerton Conference.* vol. 4, 1991, 821-830.

[23] Olariu, S., J.L. Schwing, J. Zhang, Computing the Hough Transform on Reconfigurable meshes, *Proceedings of the Conference on Vision Interface'92*, 1992, 169-174.

[24] Y. Pan, A more efficient constant time algorithm for computing the Hough transform, *Parallel Processing Letters*, vol. 4, no. 1-2, 1994, 45-52.

[25] Pavel, S., Akl S. G., On the power of arrays with reconfigurable optical buses, Technical Report 95-374, Queen's University, Kingston, 1995.

[26] Pavel, S., Akl S. G., Matrix Operations using Arrays with Reconfigurable Optical Buses, to be published in the Special Issue on Algorithms for Enhanced Mesh Architectures of the *Journal of Parallel Algorithms and Applications*.

[27] Pan, Y., Chuang, H.Y.H., Parallel Hough transform algorithms on SIMD Hypercube arrays, *1990 International Conference on Parallel Processing*, 1990, 83-86.

[28] Pan, Y., Chuang, H.Y.H., Faster line detection algorithms on enhanced mesh connected arrays, *IEE Proceedings-E*, 140, vol. 2, 1993, 95-100.

[29] Qiao, C, and Melhem, R. G. Time-division communications in multiprocessor arrays. *IEEE Transactions on Computers*. **42**(5), 1993, 577-590.

[30] Qiao, C, Melhem, R. G., Chiarulli, D.M., Levitan S.P. Optical multicasting in linear arrays, *International Journal Optical Computing*, **2**, 1991, 31-48.

[31] Qiao, C. Efficient matrix operations in a reconfigurable array with spanning optical buses. To appear in *Parallel Processing Letters*.

[32] Rosenfeld, A., Ornelas, J., Hung, Y., Hough transform algorithms for mesh-connected SIMD parallel processors, *Computer Vision, Graphics and Image Processing*, 41, 1988, 293-305.

[33] Silberberg, T.M., The Hough transform on geometric arithmetic parallel processor, *Proceedings IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Datbase Management*, 1985, 387-393.