A VLSI MULTIPROCESSOR SYSTEM FOR CIRCLE DETECTION: HARDWARE DESIGN AND LOAD DISTRIBUTION

Ming-Yang Chern (陳 明 揚)

Department of Electrical Engineering
National Chung-Cheng University
160 San-Hsing Village, Min-Hsiung, Chia-Yi, Taiwan
e-mail: ieemyc@ccunix.ccu.edu.tw

ABSTRACT

Circle detection through the use of Hough transform is usually time-consuming. In this paper, we use a VLSI multiprocessor system to generate candidate circle center addresses in parallel, while in each processor only adder operation is needed to determine the accumulator address. To match the updating speed of accumulator memory with the parallel address generation, the accumulator memory is partitioned into modules for parallel access and updating. With the number of memory modules chosen equal to the number of processors, an interleaving scheme for partitioning the circle template table and accumulator memory is proposed. It balances the load of processors and avoids accumulator memory contention. Variations of our scheme are presented and performance is analyzed in this paper as well.

Keywords: Circle detection, Hough Transform, Parallel Processing, VLSI, Multiprocessor

INTRODUCTION

Detecting circles in digitized image is frequently approached through the use of Hough transform [1-3], for the Hough transform (HT) is an effective technique for pattern recognition and has good performance even if applied over images with noise and occlusion [5].

Based on the principle of Hough transform, for any point P(x, y) on a circle in the image space, it may possibly belong to a circle of radius r with circle center located at (i, j) as given by

$$i = x + r\cos\theta$$
$$j = y + r\sin\theta \qquad (1)$$

where θ, ranging from 0° to 360°, represents the direction of circle center with respect to the point P. In other words, a detected edge point of image coordinates P(x, y) can be mapped to a set of locations ((i, j), r) in the 3-dimensional parameter space, according to Equation (1). And the content of every location AC((i, j), r) in the accumulator array must be incremented by one for each edge point mapped to it. The basic operation steps for circle-detection Hough transform can be summarized as follows:

1. Quantize the parameter space between appropriate maximum and minimum values for i, j, and r, respectively; and form an accumulator array AC((i, j), r).
2. Initialize the contents of the whole accumulator array AC to zero.
3. For each edge point P(x, y) in the edge map (image), produced by edge detection, increment all locations of its possible circle centers in the accumulator array, i.e.,

   $$\mathrm{AC}((i, j), r) = \mathrm{AC}((i, j), r) + 1$$

   for (i, j) and r satisfying $i = x + r\cos\theta$, $j = y + r\sin\theta$, in the range of our definition.
4. With certain threshold, extract local maxima from the accumulator array and report their locations (circle centers).

Appropriate threshold value usually depends on the radius r and the need of applications. The count value of the detected location provides a measure of the number of edge points on the circle while the locations of local peaks indicate the parameters of the detected circles.

The major drawback of the above conventional circle Hough transform (CCHT) is the need of large memory space and relatively long processing time. To reduce the accumulator memory requirement, one simple way is to perform a series of Hough-transforms, each of them dealing with a single value of radius [4]. By this scheme, only a two-dimensional accumulator plane is needed, yet the total processing time remains unreduced.

To reduce the processing time, two approaches are generally adopted. One is to modify the circle Hough transform in order to simplify the process, the other is to apply parallel processing. Quite a few modified versions of Hough transform have been proposed [5-12]. Many of them use edge directions while some combines the use of other properties of the circle to reduce the complexity of

Hough transform voting process. In general, these methods take advantage of certain properties of circles to reduce the processing time of the image. Nevertheless, in reducing the algorithm's complexity, the more specific we extract the image features, the more difficult it becomes to perform parallel processing on these modified algorithms. Thus the chance for further improvement of the processing speed is usually limited.

In our attempt to reduce the processing time, the use of VLSI parallel processors is considered. The success of this approach depends on the parallelism of the underlying algorithm, the homogeneity of the data flow, and the simplicity of the processor operations and system I/O. As noted in [8], the approach of using edge directions to guide the voting in accumulator plane(s) facilitates the serial performance but may not lead to the homogeneous operations for the SIMD parallel processing.

Kumar et al. [13] reported a more recent work on parallel circle detection. Three parallel algorithms on an n x n mesh-connected processor array were proposed (where n x n is the image size). The first algorithm is for CCHT; the second one performs circle HT with the guidance of gradient directions. Yet the performance of these two designs is still worse than the best (theoretical) time complexity for an order of magnitude. The third one, based on the tracing of circle edge points, runs with a time complexity of $O(n^2)$ and is the best of the three. Although the third design is efficient, the implementation of the large n x n image-sized processor array is very costly unless the circuit of the element processor can be much simplified. Further research on their design is still needed.

So far, few cost-effective parallel architecture (array processor) designs have been proposed for circle detection. To explore the possibility of designing cost-effective VLSI parallel processor, we constrain this research on parallel processor design using only dozens to few hundred processor elements. For the array processor we propose here, it does not make use of gradient direction. With the proposed partitioned memory scheme, the array processor is highly cost-effective for Hough transform-based circle detection and can be implemented on a single VLSI chip.

THE BASIC HARDWARE CONFIGURATION

For some applications, the radius of circles that we want to detect is known in advance. So we can focus on the fixed-radius circle detection. If we do not have such priori knowledge, we may still work on each possible radius one at a time. With fixed radius in Equation (1), we can pre-calculate the $r \cos\theta$ (latter denoted x_r) and $r \sin\theta$ (denoted y_r) values for all angles θ and store them in a table for later use to save computation time. When the point P(x, y) in the image is an edge point, its candidate circle center locations (i, j) can be calculated by adding the (x, y) coordinates with each table-stored (x_r, y_r) value respectively. This table look-up scheme is efficient, because it uses only the integer addition to determine the circle center (i, j). To generate the circle points (x_r, y_r) in the table (circle template), knowing the fixed radius, we may use the midpoint circle scan conversion algorithm [14,15]. There are three coordinates systems involved. The location of the edge pixel is expressed in the image coordinates; the circle-template coordinates expresses the address of a circle point relative to its center; and the accumulator-array coordinates expresses the location of circle center in the accumulator array. The notation of these three systems is shown in Figure 1.

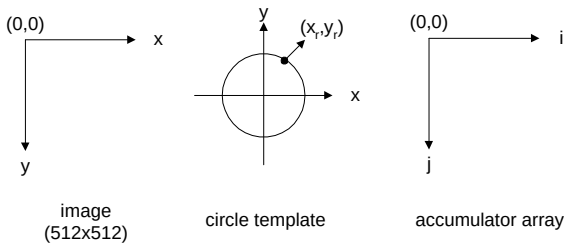

Figure 1. The three involved coordinate systems

Without losing the generality, we assume the image size of 512x512. Then each coordinate pair (x_r, y_r) of the circle template points stored in the table is chosen to be of 9 bits each (one bit for sign and 8 bits corresponding to the maximal radius size of 255).

According to Equation (1), the coordinates of circle center *(i, j)* may be negative or positive, but the accumulator array (memory) is usually accessed with positive addressing. To remedy this problem, we propose a simple address offset scheme in the hardware implementation. The address-offset scheme is also convenient to confine the circle detection to the range of the image space we are interested in. The offset values (x_{off}, y_{off}) are determined by the position of upper left corner of the accumulator array with respect to the origin of the image coordinates system.

Figure 2 shows the relationship between the offset values (x_{off}, y_{off}) and the position of accumulator array with respect to the origin of the image space. With the added address offset, the equations for calculating the accumulator address (*i, j*) must be modified as:

$$i = (x + r\cos\theta) + x_{off} = (x + x_r) + x_{off}$$

$$j = (y + r\sin\theta) + y_{off} = (y + y_r) + y_{off}$$

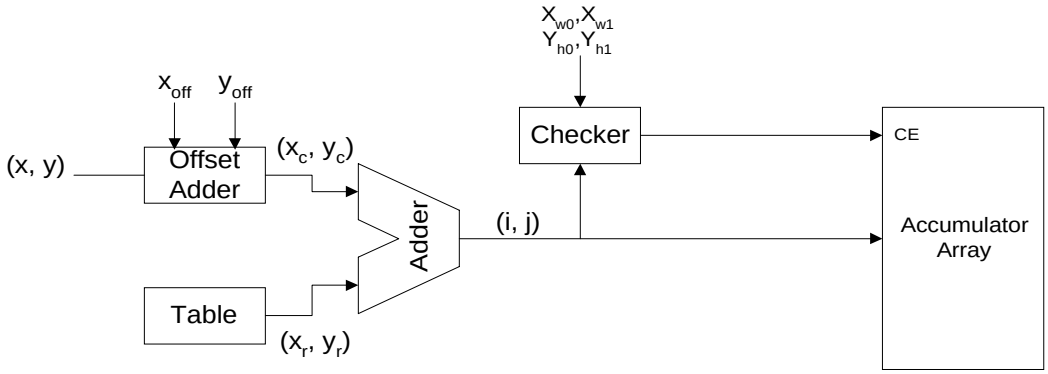

Figure 3. Basic processor hardware configuration for circle detection

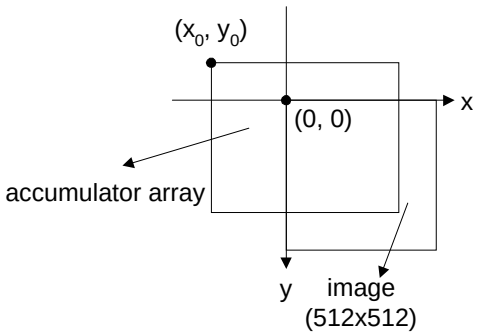

Figure 2. The relationship between offset values ($x_{off} = -x_0$, $y_{off} = -y_0$) and the upper left corner (x_0, y_0) of the accumulator array with respect to the origin of the image space

The basic hardware implementation with added address offset for calculating the (*i*, *j*) address is depicted in Figure 3. Note that the implementation adopts to add the (x_{off}, y_{off}) value first, because this sum can be shared by the parallel processors. Adding this offset first, we need only one adder pair to get the sum for all processors in the system.

The table in Figure 3 contains the address pairs (x_r, y_r) of the points in the circle template, and (x, y) denotes the position of edge point in the image. By adding the edge-point address (x, y) and the accumulator offset quantity (x_{off}, y_{off}), we have the offset edge-point address (x_c, y_c).

$$x_c = x + x_{off}$$

$$y_c = y + y_{off}$$

The values of (x_c, y_c) are then added to each pair of the (x_r, y_r) entries in the circle template table, with their sum to become the candidate circle center address (i, j) corresponding to each location (AC(i, j)) of the accumulator array to be incremented. That is,

$$i = x_c + x_r$$

$$j = y_c + y_r \qquad (2)$$

Before updating the accumulator array, the address (i, j) must be filtered first by an Address Checker. The address checker compares the incoming (i, j) address with the predefined upper and lower bounds of the accumulator array. In Figure 3, the X_{w0}, X_{w1}, Y_{h0}, and Y_{h1} values are preloaded into registers as our defined accumulator array boundary. Only when the address (i, j) is in range: $X_{w0} \leqq i \leqq X_{w1}$ and $Y_{h0} \leqq j \leqq Y_{h1}$, the control signal CE will be active. Then its corresponding position AC(i, j) of the accumulator array will be incremented. This scheme protects the accumulator memory from erroneous access. It confines the detection of circles all centered within a predefined window in the parameter space. Moreover, this provision in effect saves unnecessary accumulator memory access time and updating time as well.

After all the pair entries (x_r, y_r) in the circle template table have each summed with (x_c, y_c) to generate the address needed, the (x, y) position of the next edge point will be read in. And the same process is repeated. When all edge points in the image are processed, the peak detector then extracts peaks from the accumulator array. The locations of the extracted peaks indicate the circle center positions of the detected circles.

THE PARALLEL ADDRESS GENERATION

In the operation of circle Hough transform, for each edge point, there will be usually quite a few candidate circle center addresses (i, j) to be generated. The calculation of (i, j) by addition, though simple, would still become the speed bottleneck. Taking the advantage of the uniform operation in calculating (i, j), we may adopt a number of adder-pairs (which are the element processors, called processing elements or PEs in short) to share the load and to largely facilitate the speed of processing (i.e., the address generation).

For an incoming edge point P(x, y), the value of (x_c, y_c) can be calculated through the Offset Adder. Then the remaining tasks are to add this (x_c, y_c) value with every stored entry (x_r, y_r) of the circle template table. Just like the data flow computer, it is important to provide (or arrange) the operands for each PE to proceed. Now the problem is how to distribute the operands to the available PEs. As mentioned before, the (x_c, y_c) values can be broadcast to all PEs. In order to calculate the (i, j) address in parallel and to balance the load of each PE, the stored entries of the circle template table should be equally or near-equally divided into N partitions, one for each available PE.

The approach of partitioning the circle template table by pages (multiple consecutive rows of fixed size, such as the exponent of 2) looks feasible but may not work well for circles of small sizes. In some cases, a few PEs may not have any jobs to do. In order to minimize the difference on the number of stored entries (x_r, y_r) in the partial table module for each PE, we propose a simple address-based interleaving scheme for partitioning the circle template table. With N the number of PEs in use, the entry (x_r, y_r) allocated to the k-th partial table module must match the relationship:

$$k = \mathrm{mod}(y_r, N) \qquad (3)$$

For convenience in our design, usually the number N is set to an exponent of 2. Taking N = 8 as an example, we use the lower-order 3 bits of the y_r-value to determine the table module number k for the pair-entry (x_r, y_r) to store. Figure 4 shows how the circle template is partitioned into partial table modules for N=8.

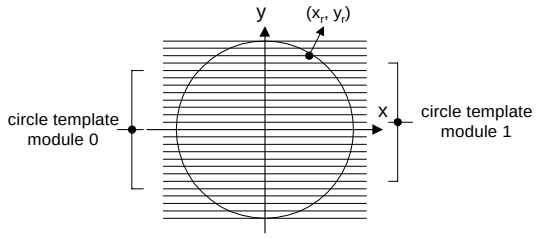

Figure 4. The partitioning of circle template table by interleaving scheme

ACCUMULATOR MEMORY PARTITIONING

With multiple processors generating the address (i, j) in parallel, it is important that the accumulator array (memory) can be updated in parallel as well. The accumulator array must be partitioned into multiple memory modules to match the speed of PEs. Since memory updating takes about the same cycle time for (i, j) address calculation, we choose the number of memory modules to be N, which is the same as the number of processors (PEs). Now the next problem is how to partition the memory. And what is the scheme that can avoid memory contention?

As the circle template table is divided into N modules by interleaving scheme as mentioned, the N processors in our design are each devoted to one different module of the circle template. Thus if the accumulator memory is partitioned in the same row-interleaving way, for any one specific incoming edge point P(x, y), all the accumulator address (i, j) generated from a specific processor would fall into the one memory module. On the other hand, the addresses generated from different processors would fall into different memory modules, since the circle template entries of different modules undergo the same displacement in the address calculation.

The need of partitioning accumulator memory modules by the same row-interleaving scheme is quite obvious. It allows a one-to-one mapping between the address generation PEs and the accumulator memory modules for any edge point P(x, y). This scheme simplifies our design and avoids memory contention.

With choice of ways to assign the memory module numbering, here we let the h-th row of the accumulator memory be allocated to the *l*-th memory module based on the expression in (4).

$$l = \mathrm{mod}(h, N) \qquad (4)$$

where N is the number of memory modules (i.e. the number of PEs as well).

Using Equation (3) for circle template modularization and Equation (4) for accumulator memory, the circle centers (i, j) calculated by each PE for an edge point P(x, y) correspond to a specific memory module. Unless the row address of the edge pixel is changed, the circle centers (i, j) calculated by one PE are all mapped into the same memory module. And this characteristic allows some convenience of data flow arrangement in our design to be presented latter.

Figure 5 presents a basic configuration of our design. It uses a switching network for the connection between PEs and accumulator memory modules. The table of each PE contains the values of the pairs (x_r, y_r) associated with circle template module (i.e. the circle template module k is preloaded into the table k). When the edge point P(x, y) is read from edge map, the (x, y) values are addressed offset by the Offset Adder. Then (x_c, y_c) values are broadcast to all PEs. The values of each stored entry (x_r, y_r) in each table and the (x_c, y_c) value are then added together to become circle center address (i, j). The (i, j) values from the same PE connects to each corresponding accumulator memory module through the switching network. Before the memory module is updated, the (i, j) value is checked first by address checker. If it is in

the range of our defined window (the range values is preloaded into the checker), the checker sends the control signal CE to the memory module to allow the increment of the corresponding accumulator location AC(i, j).

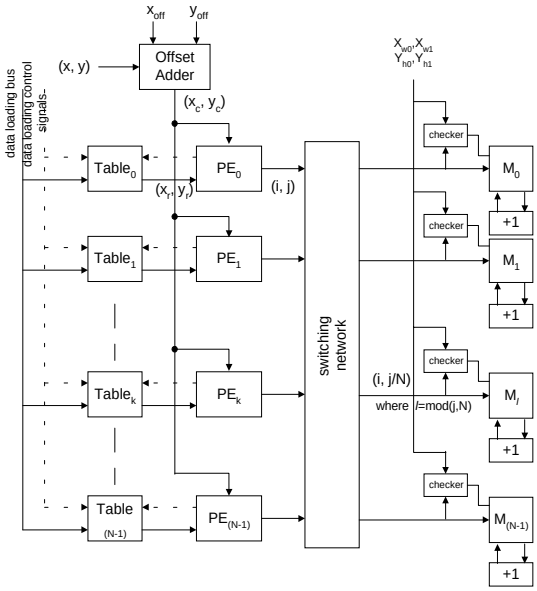

Figure 5. The parallel processor configuration for circle detection, with a switching network for the PE to accumulator module connection

It is possible to use the lower-order bits of row address of each calculated (i, j) to control the switching network. Taking N=16 as an example, we may use the lower-order 4 bits of the row-address (j) to determine which memory module M_k this j-th row in the accumulator memory should belong to. From the regularity of the switching behaviors, we can even use only the y-address of the incoming edge point to control the switching of the whole interconnection network. The switching control thus can be quite simplified. Nevertheless, besides propagation delay, the switching network requires quite a few switches (channel-width x N^2 switches). This induces the thought of alternative designs to reduce the switching network, or even better, completely eliminate the need of such a network.

Connection between Processors and Accumulator Memory Modules

In order to reduce the interconnection network between the PEs and the accumulator memory modules, we explore the various possibility in the (i, j) address generation scheme. For each PE, if it can always generate the (i, j) addresses which fall into the same accumulator memory module, then the architecture of one-to-one fixed connection can be employed and the switching network will be no longer needed.

To achieve the above premise, each PE connecting to a specific accumulator memory module must access to appropriate circle template module in responding to the input edge-point address (x, y). This means, each PE must process one appropriate (x_r, y_r) module, in responding to the current offset edge-point address (x_c, y_c), such that the resultant (i, j) addresses will map to the one specific accumulator module connected.

The one-to-one fixed connection scheme eliminates the need of interconnection network and thus much reduces the complexity of our parallel processor system. The key issue resides on the problem of how to determine which (x_r, y_r) circle template module should be selected and how to access to the selected module for each PE. Three variations of the processor design for solving this problem are to be presented in the following.

Processor with Full Template Table

The parallel processor configuration with each processor having a full circle template table is shown in Figure 6. Without losing the generality, we assume that the k-th accumulator memory module is connected to the k-th PE. The table of each PE contains all the stored pair-entries $\{(x_r, y_r)\}$, and the PE can access any pair (x_r, y_r) in the table.

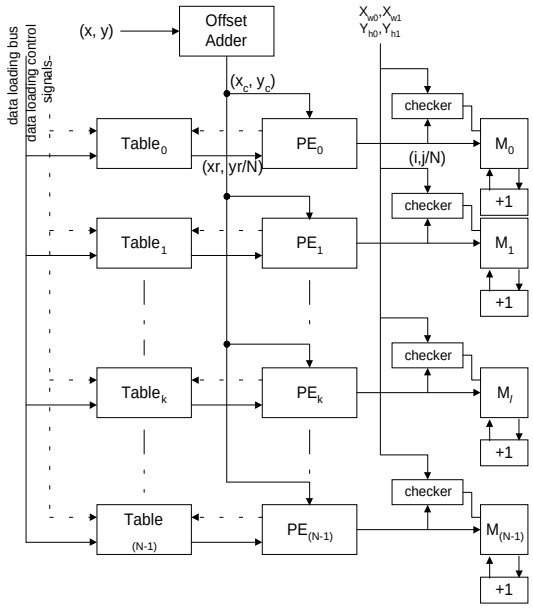

Figure 6. Parallel processor configuration of which each processor has a full circle template table

For a given edge point P(x, y) addressed with offset, the k-th PE may select to process the group of table entries which belong to the same circle template module. Since we desire to have all the generated (i, j) addresses falling into the k-th accumulator module, we must have:

$$\text{mod}(j, N) = k \qquad (5)$$

From equation (2), we have $j = y_r + y_c$. Thus the above equation can be rewritten as:

$$\text{mod}((y_r + y_c), N) = k$$

$$\text{mod}((y_r + y_c - k), N) = 0$$

$$\text{mod}(y_r, N) = \text{mod}(-y_c + k, N) \qquad (6)$$

where y_r is the y-value of the pair (x_r, y_r); y_c is edge point's y-value with offset; k is the numbering of the accumulator module for the k-th processor ($0 \leqq k \leqq N-1$); and N is the total number of processors. For N=16, we may use the lower-order 4 bits in $(-y_c + k)$ to determine which group of entries (x_r, y_r) should be selected. As a result of this mechanism, all the (i, j) addresses generated by the k-th PE will definitely have the lower-order 4 bits of the j-address equal to k. In fact, only the remaining bits of the (i, j) address is used to address the connected accumulator memory module.

Since each PE must access to all the entries $\{(x_r, y_r)\}$ of the circle template, a large table memory is needed in each PE. Based on the geometric symmetry of circle in Figure 4, we may store only the right half of circle in the template table and use 2's complement to generate the left half (i.e. the entry (x_r, y_r) generates $(-x_r, y_r)$ through the 2's complement device). The entry (x_r, y_r) may generate (i, j) and $(-x_r, y_r)$ values at the same time, so no extra time is consumed. This scheme reduces the table (memory) size to one half; it saves hardware resources.

The above proposed design has two major drawbacks: (1) Each PE needs to determine which entries (x_r, y_r) are to be used to generate (i, j) for the connected accumulator module. (2) The table's memory requirement is somewhat large, though the full table can be reduced to a half table. The mechanism of using partial circle template (i.e., the circle template module) in the table is thus proposed in the following.

Each Processor with One Partial Table

The parallel processor configuration with each processor having one partial table is shown in Figure 7. This new configuration is almost the same as the one shown in Figure 6 except its table size and the table circulation capability.

Based on the relationship in Equation (2), the contents of each circle template module are initially preloaded in the partial table of each corresponding PE. On the other hand, the edge points to be processed are sorted into N sets according to the lower-order bits of their y-addresses. With the pre-known offset value (x_{off}, y_{off}), we may let an edge point (x, y) belong to the set with ID number = $\text{mod}(y+y_{off}, N) = \text{mod}(y_c, N)$. For edge points of the same set performed with the same circle template module (i.e., the same partial table), the resultant addresses will all map to the same accumulator memory module. This is obvious, as we can see from the equation:

$$\text{mod}((y_r + y_c), N) = \text{mod}((y_r + y + y_{off}), N) = k$$

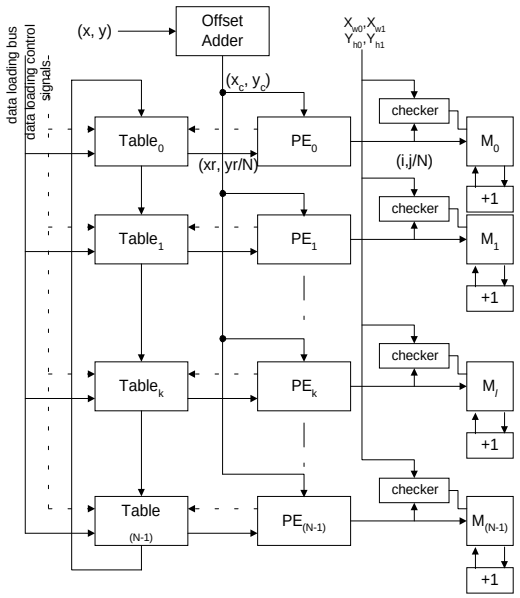

Figure 7. Parallel processor configuration of which each processor has one partial table

With the set number defined above, the sorted edge points are input to our system sequentially and set by set in the order of ascending set number. Every time before the first edge point of the next set is processed, the circle template module in each PE must be shifted to the partial table of the next PE for matching the new-coming set of edge points to produce addresses belonging to the same accumulator memory module. The one-to-one fixed connection between the processors and accumulator memory modules thus can be maintained.

This proposed design reduces the memory requirement much by using the partial table scheme, but it needs the shifting time for each new set of the edge points. To reduce the processing time further, the approach of using alternating partial tables can be adopted.

Each Processor with Two Partial Tables

In the alternating partial table scheme, two partial tables are implemented for each processor. During the operation, one of these two tables contains the entries of the current circle template module, while the other is used to load the circle template for the next incoming set of edge points. The PE accesses the data from its two partial tables

in an alternating way. Thus using two partial tables can eliminate the effect of the circle template data shifting time delay in the previous design.

SYSTEM OPERATION

We use the proposed design to detect circles of specific radius. At first, the pre-calculated circle template entries of the specific radius are loaded into the table of each PE. Then the address offset values are loaded into the Offset Adder registers, while the check-bound (window limits) values are loaded into the address-checker registers for confining the range of circle center (i, j) within our defined domain. With all the preloaded data ready and the accumulator memory been reset, the edge points in the image are then sequentially read in to generate candidate circle center locations for Hough transform.

After all the edge points are processed, the above-threshold peaks in the accumulator array must be extracted. The peak point detected in the accumulator array indicates the detection of a circle, which is centered at that peak position and of the specific radius. The count value of the peak point represents the number of the edge points on that circle. As to the hardware for detecting peaks from the accumulator memory, there are different ways to design it. In our study, an on-chip parallel hardware for such peak detection has been reported in [16].

The proposed array processor is designed for detecting circles of a specific size (radius). For circles of unknown size or within a limited range of radius values, the detection of circles of each possible radius can be successively applied on our processor. On the other hand, since the PE circuit in our processor design is quite simple and the number of PEs is around hundred or fewer, it is feasible to implement such array processor on one single VLSI chip. Thus in case we want to facilitate the processing speed, we may use multiple VLSI array processor chips to construct a parallel circle detector hardware for the 3-parameter ((i, j), r) space.

THE PERFORMANCE ANALYSIS

Our proposed parallel processor design is based on the much-simplified operations for the circle detection Hough transform. With the use of table look-up technique, the calculation of the candidate circle center address is reduced to the operation of addition only. The concurrent and pipelined operations of specialized hardware components, such as the offset adder, the address checker, and the accumulator memory incrementor, do contribute to the speed-up of our processor over the step-by-step operation of the usual general-purpose processor CPU for several folds.

On the other hand, the number of processors N contributes to the speed-up factor in another dimension. Theoretically, the upper limit of speed-up due to the N-processor parallel processing is N times. With the increase of N, the accompanying increase of processing speed depends on whether the workload can be evenly distributed to all processors. In our circle detection array processor design, the workload is the number of circle template entries to be accessed in each PE. The PE, which has the largest number of circle template entries, would become the bottleneck of the whole array processor. Under the row-interleaving scheme, the top row and the bottom row of a circle usually have the largest number of circle template entries. The plot of speed-up factor versus N for a few different radii in Figure 8 shows the trend. For N = 32, the speed-up factor is about 50% of the ideal case. While for N = 64, the average speed-up factor for r = 20 to 120 is only about 30%.

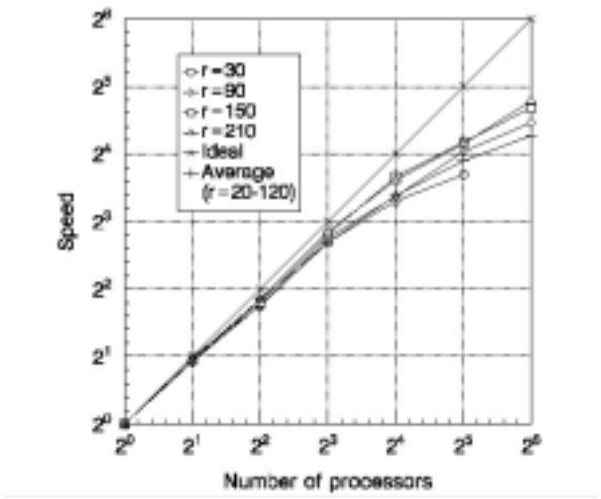

Figure 8. The plot of speed-up factor versus the number of processors for circles of various sizes (under the row-interleaving scheme)

For large radius and relatively small N, the workload is about evenly distributed and the speed-up factor follows N, since there is only a small percentage of variation among the workload of all PEs. As N comes near the size of the radius, the performance is not smoothly improved but dependent on the largest number of template entries in PEs. (ref. to Fig. 9, the plot has more fluctuation for large N.)

From the plot in Figure 9, we find that the performance would be much deteriorated for large N (unless the circle radius is large). To balance the load and thus improve the speed, we propose a cross-interleaving scheme. Under this new scheme, the modularization of the template table and accumulator memory is in both y- and x-directions. And as a result, in general, the workload (table

entries) would be more evenly distributed. The comparison of these two schemes is shown in Figure 10. The cross-interleaving scheme shows more rippling in its plot. For small N = 16, the performance is near. Yet for large N (such as N = 64), it has obviously better performance.

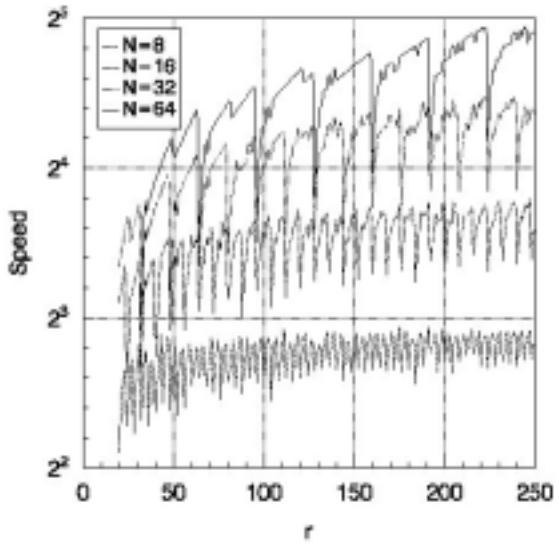

Figure 9. The plot of speed-up factor versus radius for fixed number of processors (under the row-interleaving scheme)

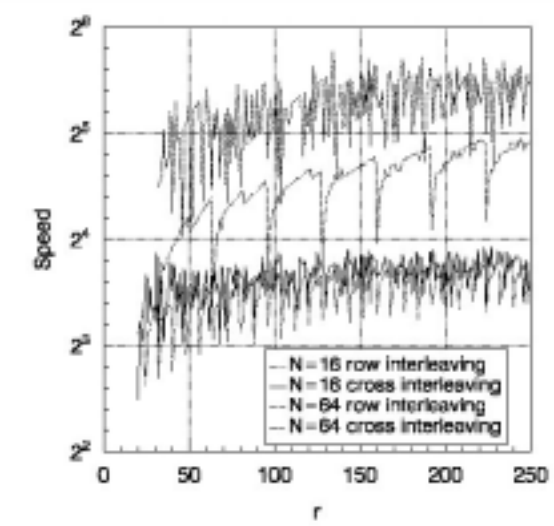

Figure 10. The plot of speed-up factor versus radius for some fixed number of processors under different schemes

The cross-interleaving scheme does improve the performance for the case of large N, yet it is still 30% to 60% below the ideal case depending on the radius. Considering that the Hough transform is a statistical detection method, a small percentage of undetected edge points would not affect the result. Thus we propose to cut a small percentage of the table entries (from those PEs having larger number of entries). The simulated result is shown in Figure 11. The plot shows a large improvement even we adopt the simple row-interleaving scheme. For N = 64, the 5% cut of circle template entries is competitive to the cross-interleaving scheme. With 10% cut, the performance is even better and reaches the ideal speed for some circle radii.

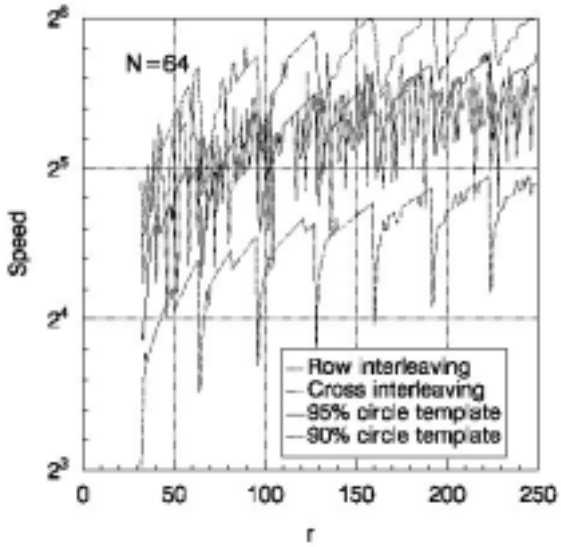

Figure 11. The plot of speed-up factor versus radius using 64 processors under different schemes

CONCLUSION

In this paper, we present a VLSI parallel processor for high-speed circle detection. In our proposed design, the circle center candidates (i, j) can be determined using only the addition operations. While the accumulator array is partitioned into memory modules so that it can be updated in parallel as well. We adopt the row-interleaving scheme to modularize the circle template for the partial table in each PE, and to allocate one accumulator memory module for each PE. This scheme achieves to distribute the (i, j) calculation evenly among the multiple PEs, and avoid the contention on accumulator array memory updating.

From the variations of our memory-interleaving scheme, we see the high potential to reach ideal performance, while it can be extended for other geometrical shape detection. As our design has a high degree of modularity, regularity and simplicity, it is highly suitable for VLSI implementation. In practice, our proposed array processor can be implemented on a single VLSI chip and is highly cost-effective for parallel circle-detection.

ACKNOWLEDGEMENT

This research was supported by the grant NSC-89-2215-E-194-010, from the National Science Council of the Republic of China.

REFERENCES

[1] P.V.C. Hough, "Method and means for recognizing complex patterns," U.S. Patent 3069654, 1962.

[2] R.O. Duda and P.E. Hart, "Use of Hough transformation to detect lines and curves in pictures," *Comm. ACM*, Vol. 15, pp. 11-15, 1975.

[3] C. Kimme, D.H. Ballard, and J. Sklansky, "Finding circles by an array of accumulators," *Comm. ACM*, Vol. 18, pp. 120-122, 1975.

[4] G. Gerig and F. Klein, "Fast contour identification through efficient Hough Transform and simplified interpretation strategy," *Proc. 8th Int. Joint Conf. Pattern Recognition*, pp. 495-500, 1986.

[5] J. Illingworth, and J. Kittler, "A survey of the Hough transform," *Computer vision, Graphics and Image Processing*, Vol. 44, pp. 87-116, 1988.

[6] E.R. Davies, "A modified Hough scheme for general circle location," *Pattern Recognition Letters*, Vol. 7, pp. 37-43, 1988.

[7] H.K. Yuen, J. Princen, J. Illingworth, and J. Kittler, "Comparative study of Hough Transform methods for circle finding," *Image and Vision Computing*, Vol. 8, pp. 71-78, 1990.

[8] R. Chan and W.C. Siu, "New parallel Hough transform for circles," *IEE Proceedings-E*, Vol. 138, pp. 335-344, 1991.

[9] R.K.K. Yip, P.K.S. Tam, and D.N.K. Leung, "Modification of Hough transform for circles and ellipses detection using a 2-dimensional array," *Pattern Recognition*, Vol. 25, pp. 1007-1022, 1992.

[10] P. Kierkegaard, "A method for detection of circular arcs based on the Hough transform," *Machine Vision and Applications*, Vol. 5, pp. 249-263, 1992.

[11] C.T. Ho, and L.H. Chen, "A fast ellipse/circle detector using geometric symmetry," *Pattern Recognition*, Vol. 28, pp. 117-124, 1995.

[12] N. Guil and E.L. Zapata, "Lower order circle and ellipse Hough transform," *Pattern Recognition*, Vol. 30, pp. 1729-1744, 1997.

[13] S. Kumar, N. Ranganathan, and D. Goldgof, "Parallel algorithms for circle detection in images," *Pattern Recognition*, Vol. 27, pp. 1019-1028, 1994.

[14] B.K.P. Horn, "Circle generators for display devices," *Computer Graphics and Image Processing*, Vol. 5, pp. 280-288, 1976.

[15] D. Hearn, and M.P. Baker, Computer Graphics, 2nd Ed., Prentice-Hall, Chapter 3, 1994.

[16] M.Y. Chern and C.M. Dai, "Design of VLSI Parallel Processors for Hough Transform-based Line Detection", Journal of The Chinese Institute of Electrical Engineering, Vol.7, no.1, 2000, pp.41-52.