ON-CHIP COMPUTATION OF EULER NUMBER OF A BINARY IMAGE FOR EFFICIENT DATABASE SEARCH[†]

Arijit Bishnu , Bhargab B. Bhattacharya[‡] *, Malay K. Kundu, C. A. Murthy*
{bishnu_t, bhargab, malay, murthy}@isical.ac.in
Indian Statistical Institute, 203 B. T. Road, Calcutta - 700035
and
Tinku Acharya
tinku.acharya@intel.com
Intel Corporation, Chandler, AZ - 85226, USA

ABSTRACT

Euler number is a fundamental topological feature of an image, which remains invariant under translation, rotation, scaling, and rubber-sheet transformation of the image. In this work, a novel algorithm for computing the Euler number of a binary image is proposed which is based on the properties of runs of 0's and 1's present in the pixel matrix. The algorithm outperforms significantly the existing techniques in terms of both the number of pixel accesses and CPU time. It can be easily parallelized, and a simple on-chip implementation is reported here. Results on a database consisting of 1039 logo images reveal that Euler number has a strong discriminatory power, and hence can be used for efficient database searching or matching of binary images. The proposed algorithm is very fast and easy to implement, and has potential of wide applicability in image processing.

1. INTRODUCTION

Topological properties remain invariant under any arbitrary *rubber-sheet* transformation and hence, are very useful in image characterization for matching shapes, recognizing objects, image database retrieval, and many other image processing and vision applications. An important topological feature of an image is the *Euler number* (or genus), which is the difference between the number of connected components (objects), and the number of holes [3, 5]. Many critical image processing applications involve large amount of data, and at the same time demand quick real-time response. Euler number provides a simple and fast method of screening in such cases. Euler number of cell images can be used in medical diagnosis in a variety of ways.

The classical algorithm for computing the Euler number of a binary image is based on counting certain (2×2) pixel patterns called bit quads [4]. Dyer [2] proposed an algorithm to compute the Euler number of an image represented by a quadtree. Samet and Tamminen [6] improved the algorithm further by using a new staircase type of data structure to represent the blocks that have already been processed. However, it is not suitable for VLSI implementation. Recently, Dey et al. [1] have reported a divide-and-conquer algorithm for computing the Euler number of a binary image, and its VLSI implementation.

In this work, a new algorithm is proposed based on certain properties of 0-1 runs of the pixel matrix, that computes the Euler number of a binary image very fast. Although the worst-case complexity of the algorithm is still $O(N^2)$ for an $(N \times N)$ image, its average-case behavior outperforms significantly all the earlier algorithms [1, 2, 4, 5]. Next, a parallel version of the algorithm is described that can be executed in $O(N)$ time using $O(N)$ simple processing elements. The proposed algorithm can easily be implemented with a special purpose VLSI chip to expedite computation. The algorithm was run on a database of 1039 logo images for computing the Euler number of each of them. Experimental results are found to be very encouraging, and demonstrate the superiority of this method to earlier approaches.

2. PROPOSED SEQUENTIAL ALGORITHM

2.1. Theme

Let the binary image be represented by a 0-1 pixel matrix of size $(N \times M), N \geq M$, in which an object (background) pixel is denoted as 1 (0). In a binary image, a *connected component* is a set of object pixels such that any object pixel in the set is in the 8 (or 4) neighborhood of at least one object pixel of the same set. A *hole* is a set of background pixels such that any background pixel in the set is in the 4 (or 8) neighborhood of at least one background pixel of the same set and this entire set of background pixels is enclosed by a connected component. A *run* in any column (or row)

[†] This work is funded by a grant from Intel Corp., USA (PO #CAC042717000); US Patent pending, Nov. 2000.

[‡] Author for correspondence.

of the pixel matrix is defined to be a maximal sequence of consecutive 1's in that column (or row). Let *R(i)* denote the number of such runs in the i-th column (row).

Fact 1 *If the image I consists of a single row or a single column i, the Euler number $E(I) = R(i)$.*

Fact 2 *Euler number satisfies the additive set property. Given two images I_1 and I_2 with Euler numbers $E(I_1)$ and $E(I_2)$ respectively, the Euler number of the image $I = I_1 \cup I_2$ is given by: $E(I) = E(I_1 \cup I_2) = E(I_1) + E(I_2) - E(I_1 \cap I_2)$ (see [4]).*

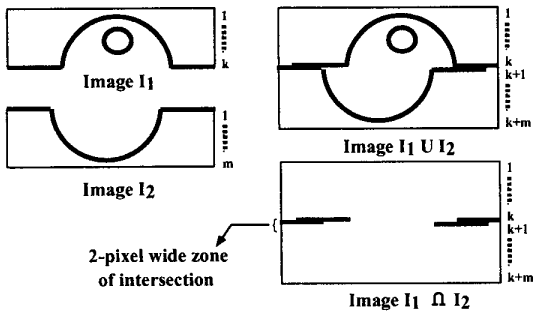

Fig. 1. Union and intersection of images

The *union* ($\cup$) of two images is defined as simple juxtaposition of I_1 and I_2 either vertically or horizontally, without any overlap. The *Intersection* ($\cap$) of I_1 and I_2 is the image formed by the last row (or column) of I_1, and the first row (or column) of I_2, if the images I_1 and I_2 are joined horizontally (or vertically). The intersection image is always two pixel row (or column) wide. Without any loss of generality, let I_1 and I_2 be joined horizontally. Thus, the last row of I_1 will lie above the first row of I_2. Two runs appearing in two adjacent rows each, are said to be *neighboring* if at least one pixel of a run is in the 8 (or 4) neighborhood of a pixel of the other run; we follow 8 neighborhood convention throughout. Clearly, $(I_1 \cap I_2)$ will denote the image containing the last row of I_1 and the first row of I_2, and as no holes can be present in a two-row wide image, we have the following observation:

Fact 3 $E(I_1 \cap I_2)$ = *the number of neighboring runs between I_1 and I_2.*

We now use Facts 1,2 and 3 iteratively to compute the Euler number of the entire image, as follows.

Let I_{i-1} be the partial image consisting of rows $1, 2, \ldots, (i-1)$ of the pixel matrix. Let $E(I_{i-1})$ be the Euler number of I_{i-1}. The Euler number of the image consisting of only row $i = R(i)$ (by Fact 1). The row i is now added to I_{i-1} to form the union image I_i. The intersection image is formed by the $(i-1)^{th}$ and the i^{th} row. Let the number of neighboring runs between them be O_i. Hence,

$E(I_1) = R(1)$

$E(I_2) = E(I_1) + E(2) - O_2 = R(1) + R(2) - O_2$

$E(I_3) = E(I_2) + E(3) - O_3 = R(1) + R(2) + R(3) - (O_2 + O_3)$

...

$E(I_N) = E(I_{N-1}) + E(N) - O_i$

$= (R(1) + R(2) + R(3) + \ldots + R(N)) - (O_2 + O_3 + \ldots + O_N)$

$= \sum_{i=1}^{N} R(i) - \sum_{i=2}^{N} O_i$

where, I_N denotes the entire image.

Thus, the following key result is proved which provides the basis of our algorithm.

Theorem 1 *The Euler number of a given binary image is the difference between the sum of the number of runs for all rows (or columns), and the sum of the neighbouring runs between all consecutive pairs of rows (or columns).*

2.2. Algorithm

Method

Input: An $(N \times M)$ binary pixel matrix of an image I;
Output: Euler number $E(I)$.
Compute_Euler_Sequential
compute the number of runs $R(1)$ present in the first row;
$E(I) = R(1)$;
for (i = row number 2 to N)
calculate the number of runs $R(i)$ in the i^{th} row;
calculate the number of neighboring runs O_i between $(i-1)^{th}$ and i^{th} rows;
$E(I) = E(I) + R(i) - O_i$;
endfor
return E(I) as the Euler number.

2.3. Space and time complexity

For M columns, the maximum number of runs in a row is $\lceil (M/2) \rceil$. We calculate the number of neighborhood runs between two consecutive rows at a time. Since a run can be designated by its two ends, the total space required = $(N \times M) + 4 \times \lceil (M/2) \rceil \approx O(N \times M)$.

The time complexity can be measured in terms of the number of pixel accesses. Computation of runs for all the rows needs $(N \times M)$ pixel accesses. The number of pixel accesses required to find the two end points of all runs present in the matrix $= 2 \times \sum_{i=1}^{N} R(i) \leq (N \times M)$. Checking whether a run in row $(i-1)$ is in the neighborhood of a run of row i, requires 2 pixel accesses. Determination of neighboring runs for all consecutive pairs of rows needs $= 2 \times \sum_{i=2}^{N} O_i \leq (N \times M)$ pixel accesses. Therefore, the total number of pixel accesses needed is, $P = (N \times M) + 2 \times \{\sum_{i=1}^{N} R(i) - \sum_{i=2}^{N} O_i\}$ which is $O(N \times M)$ in the worst case. Thus, it has the same order as that of the earlier method [4]. However, almost for all the cases, the actual performance of our algorithm is found to be superior. For a given image, the number of pixel accesses becomes equal to $2 \times \{(N \times M) + M\}$, if the bit quad counting algorithm [4] is used to compute Euler number. In the proposed method, the number of pixel accesses depends on the distribution of the runs in the pixel matrix. It has been observed that for most of the images, $R(i) \ll M$ and also

$O_i \ll M$, and typically both $R(i)$ and O_i have a value around 4. Thus, on the average, the number of pixel accesses will be much less compared to those of the earlier methods [1, 2, 4, 5]. Empirical evidence justifies the rationale behind savings. We have considered a database of 1039 logo images, and normalized each of them to the same size. From the experimental results, the expected value of the number of runs $R(i)$ present in a column is observed to be 4.252741 and that of the neighboring runs between two consecutive columns, O_i is found to be 4.266170.

2.4. Results

We have coded our algorithm in C, and studied its performance in terms of the number of pixel accesses as well as CPU time compared to an earlier sequential algorithm [4], for each image in the logo database. We have observed significant improvement in all these cases. Because of space limitation, only a few of them are shown in Table 1. The CPU time is for a Sun Ultra-5_10, Sparc; the OS is SunOS Release 5.7 Generic. The values of Euler number of the images in the database vary widely from -3796 to 2425. There are 85 images having Euler numbers in the range -3796 to -10; 829 images in the range, -9 to 10; and 125 images in the range, 11 to 2425. There are 76 images each with a distinct Euler number; there are 18 cases where only two images have the same Euler number. This observation justifies that Euler number can be used as a potential tool for image discrimination, search, and retrieval.

3. PARALLEL ALGORITHM AND VLSI IMPLEMENTATION

3.1. Parallel algorithm

The above algorithm can be easily parallelized so as to make it suitable for on-chip implementation.

Method

Input: An $(N \times M)$ binary pixel matrix of an image I;
Output: Euler number $E(I)$.
Compute_Euler_Parallel

for (i = row number 1 till end do in parallel)
calculate the number of runs $R(i)$ in the i^{th} row;
calculate the number of neighboring runs O_i between $(i-1)^{th}$ and i^{th} rows;
endfor
$E(I) = \sum_{i=1}^{N} R(i) - \sum_{i=2}^{N} O_i$
return E(I) as the Euler number.

3.2. VLSI architecture

We require two types of processing elements (PE), P_1 for computing the number of runs in a row, and P_2 for computing the number of neighboring runs.

The PE P_1 (shown in Figure 2) is used to compute the number of transitions from 0-to-1 which is equal to the number of runs. The D flip-flop is initialized to 0 in the beginning of processing each row. It holds the value of the previous pixel for the purpose of checking a transition. The pixels in a row are pipelined into P_1. At any instant of time t_i, the i^{th} and $(i-1)^{th}$ pixel of a row are checked for a 0-to-1 transition. Since, the maximum number of runs possible in a row is $\lceil (M/2) \rceil$, we need a counter of size modulo-$\lceil (M/2) \rceil$.

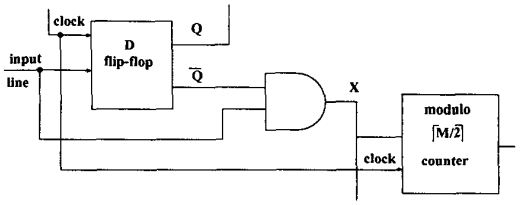

Fig. 2. Processing element P_1

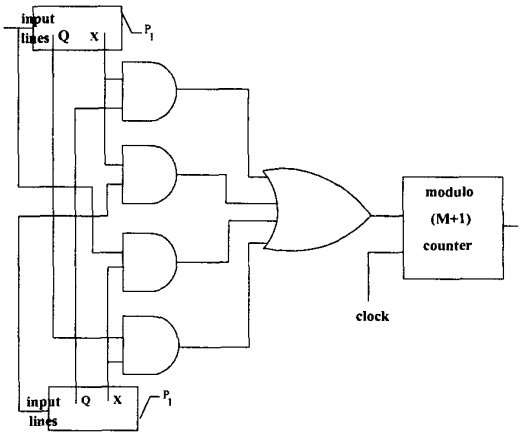

Fig. 3. Processing element P_2

In Figure 3, P_1 is shown within a box, and the remaining portion of the circuit constitutes the processing element P_2. It is used to compute the number of neighboring runs between two consecutive rows. The PE P_2 checks for the condition when a particular run in a row begins, and whether it is in the neighborhood of another run in its adjacent row. The pixels corresponding to the columns of two adjacent rows are fed to P_2 in a pipeline. In addition, it receives data from the outputs of the D flip-flops of these two rows. At any instant of time t_i, i^{th} and $(i-1)^{th}$ pixels of two consecutive rows are checked for a neighboring run. The maximum number of neighboring runs is $2 \times \lceil (M/2) \rceil$. Thus, the counter used in P_2 should be of size modulo-$(M+1)$. To process an $(N \times M)$ image in parallel, we require N pieces of P_1, and $(N-1)$ pieces of P_2. The outputs of P_1 and P_2 are fed to adders A_1 and A_2 respectively. The module A_1 is an N-input adder and A_2 is an $(N-1)$-input adder. The circuits for A_1 and A_2 should be designed in such a way that they should be able to handle a sum of $\{N \times \lceil (M/2) \rceil\}$ and $\{2 \times (N-1) \times \lceil (M/2) \rceil\}$ respec-

Table 1.

Image number	Image size	Euler number	Pixel access			CPU time (in microseconds)		
			Proposed method	Gray's method	% saving	Proposed method	Gray's method	% saving
1	274x274	-1	80800	150700	46.4	12251	23175	47.1
2	226x226	7	56108	102604	45.3	8737	17849	51.1
3	220x220	-4	54422	97240	44.0	9046	17112	47.1
4	176x176	-3	39644	62304	36.4	7191	9324	22.9
5	128x128	-2	26490	33024	19.8	4397	4994	12.0
6	256x256	0	68086	131584	48.3	10890	19638	44.5
7	475x280	1	144596	266950	45.8	22766	39373	42.2
8	256x256	-3	67426	131584	48.8	9810	19526	49.8
9	459x456	0	214512	419526	48.9	33606	65073	48.4
10	512x256	-2	140828	263168	46.5	23449	39264	40.3

tively. The subtractor S should handle values in the range $[(-)2 \times (N-1) \times \lceil (M/2) \rceil$ to $N \times \lceil (M/2) \rceil]$.

3.3. Time Complexity

We process a column of the pixel matrix $(N \times M)$ in parallel at every clock pulse. Thus, $M+1$ clock cycles are needed to process the entire pixel matrix including initialization. The adders need $O(\lceil \log N \rceil)$ time assuming that unit time is required to add two words of size $\lceil \log(M+1) \rceil$. An additional time unit is assumed to be required to complete the final subtraction. Thus, the time complexity of the algorithm is $O(N)$. The complexity of the best sequential algorithm for finding the Euler number is $O(N^2)$. Using $O(N)$ PE's and adders, we here achieve an $O(N)$ parallel time of execution. Thus, speed up is $O(N)$.

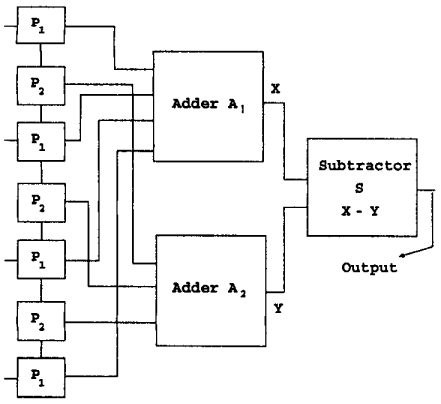

Fig. 4. A sample circuit for calculating the Euler number of a (4×5) image

3.4. Circuit cost

We require the following components to implement the VLSI architecture for processing an $(N \times M)$ binary image:
1) N pieces of edge triggered D-flip-flops;
2) $(5 \times N - 4)$ pieces of 2-input AND gates;
3) $(N-1)$ pieces 4-input OR gates;
4) N pieces of modulo-$\lceil (M/2) \rceil$ counters;
5) $(N-1)$ pieces modulo-$(M+1)$ counters;
6) $(2N-2)$ pieces of $\lceil \log(M+1) \rceil$-bit word size two-input adder/subtractor.
A complete circuit for a (4×5) image is shown in Fig. 4.

4. CONCLUSIONS

A fast algorithm for computing the Euler number of a binary image is presented. The algorithm is based on certain combinatorial properties of runs present in the pixel matrix of the image. A parallel version of the algorithm can be executed in $O(N)$ time using $O(N)$ PE's. A simple on-chip VLSI implementation has also been reported.

Acknowledgments: We would like to thank Prof. Anil K. Jain and Aditya Vailya of the Michigan State Univ., USA, for sending us the logo trademark database.

5. REFERENCES

[1] Dey, S., Bhattacharya, B. B., Kundu, M. K., Acharya, T., "A Fast Algorithm for Computing the Euler Number of an Image and its VLSI Implementation", *Proc. 13th Intl. Conf. on VLSI Design*, 2000, pp. 330-335.

[2] Dyer, C. R., "Computing the Euler Number of an Image from its Quadtree", *Computer Graphics Image Processing*, vol. 13, no. 3, pp. 270-276, July 1980.

[3] Gonzalez, R.C., and Woods, R.E., *Digital Image Processing*. Addison-Wesley, Reading, Massachusetts, 1993.

[4] Gray, S. B., "Local Properties of Binary Images in Two Dimensions", *IEEE Trans. Computers*, no. 5, pp. 551-561, May 1971.

[5] Pratt, W. K., *Digital Image Processing*. John Wiley & Sons., 1978.

[6] Samet, H. and Tamminen, H., "Computing Geometric Properties of Images Represented by Linear Quadtrees", *IEEE Trans. PAMI*, vol. PAMI-7, no. 2, March 1985.