Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Systems Architecture 51 (2005) 470–487
JOURNAL OF SYSTEMS ARCHITECTURE
www.elsevier.com/locate/sysarc

A pipeline architecture for computing the Euler number of a binary image ☆

Arijit Bishnu [a], Bhargab B. Bhattacharya [b,*], Malay K. Kundu [b], C.A. Murthy [b], Tinku Acharya [c]

[a] *Japan Advanced Institute of Science and Technology, 1-1, Asahidai, Tatsunokuchi, Ishikawa 9231292, Japan*
[b] *Center for Soft Computing Research, Indian Statistical Institute, 203 B.T. Road, Kolkata 700108, India*
[c] *Avisere Inc., Tucson, AZ 85711, United States*

Received 22 September 2003; received in revised form 1 March 2004; accepted 4 December 2004
Available online 11 February 2005

Abstract

Euler number of a binary image is a fundamental topological feature that remains invariant under translation, rotation, scaling, and rubber-sheet transformation of the image. In this work, a run-based method for computing Euler number is formulated and a new hardware implementation is described. Analysis of time complexity and performance measure is provided to demonstrate the efficiency of the method. The sequential version of the proposed algorithm requires significantly fewer number of pixel accesses compared to the existing methods and tools based on bit-quad counting or quad-tree, both for the worst case and the average case. A pipelined architecture is designed with a single adder tree to implement the algorithm on-chip by exploiting its inherent parallelism. The architecture uses $O(N)$ 2-input gates and requires $O(N \log N)$ time to compute the Euler number of an $N \times N$ image. The same hardware, with minor modification, can be used to handle arbitrarily large pixel matrices. A standard cell based VLSI implementation of the architecture is also reported. As Euler number is a widely used parameter, the proposed design can be readily used to save computation time in many image processing applications.

Keywords: Euler number; Image processing; Pipeline architecture; VLSI implementation

☆ A preliminary version of this paper appeared in the Proceedings of the *International Conference on Image Processing* (ICIP), vol. III, Greece, 2001, pp. 310–313. This work was funded in part by a grant from the Intel Corporation, USA.

* Corresponding author. Tel.: +91 33 2575 3003; fax: +91 33 2577 3035.
E-mail addresses: arijit@jaist.ac.jp (A. Bishnu), bhargab@isical.ac.in (B.B. Bhattacharya), malay@isical.ac.in (M.K. Kundu), murthy@isical.ac.in (C.A. Murthy), tinku.acharya@avisere.com (T. Acharya).

1383-7621/$ - see front matter
doi:10.1016/j.sysarc.2004.12.001

1. Introduction

Topological properties that remain invariant under various transformations are useful in image characterization for matching shapes, recognizing objects, image retrieval from a database, and in many other image processing and computer vision applications. An important topological feature of a binary image is the *Euler number* (or genus), which is defined as the difference of the number of connected components (objects), and the number of holes [1,2]. Euler number remains invariant under translation, rotation, scaling, and rubber-sheet transformation of the image. 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 is widely used in medical diagnosis [3]. It has recently been observed that Euler number is the most clinically useful feature that discriminates many cervical disorders [4]. Being a fundamental topological feature, it has numerous applications in image processing, e.g., optical character recognition, document image processing [5], reflectance-based object recognition [6], analysis of sandstone for geological applications [7], shadow detection [8]. Other topological properties using the convex deficiency and convex hull of the shape are used in conjunction with Euler number for classifying typewritten letters or binary silhouettes [9]. Recently, based on Euler number, the concept of Euler vector is introduced to characterize a gray-tone image [10].

The classical algorithm for computing the Euler number of a binary image is based on counting certain (2×2) pixel patterns called bit-quads [2,11] over the entire image. Gray [11] used the fact that the Euler number of a region of space is locally countable [12]. The classical graph-theoretic definition of Euler number relating vertices, edges and faces is applied to an image followed by triangulation and then its Euler number is computed as the difference of the number of connected components and that of holes. Alternatively, Euler number is shown to be the difference of left-facing convexities and concavities in the image. Computation of these parameters needs counting of specific types of bit-quads in the image namely,

$$Q_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \quad Q_2 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \quad \text{and} \quad Q_D = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and their rotational equivalents. Let v, t, and d be the patterns Q_1, Q_2, and Q_D respectively in an image. Then the Euler number can be expressed as $(\sum v - \sum t - 2 \times \sum d)/4$ under the 8-connectivity definition. The commercial image processing toolbox MATLAB uses the above quad counting algorithm for Euler number computation [13]. Dyer [14] proposed an algorithm to compute the Euler number of an image represented by a quad-tree. Samet and Tamminen [15] improved the algorithm further by using a new staircase type of data structure to represent the blocks that have already been processed. Juan et al. [16] considered a skeletonized version of the binary image and computed Euler number in terms of the number of terminal points T_p, and the number of three-edge points TE_p, as $(T_p - TE_p)/2$. The terminal and three-edge points are defined from pixel neighborhood relations. Chen and Yen [3] developed a parallel localized algorithm using square graphs to calculate the Euler number of a given binary image on a square grid. Chiavetta and Gesú [17] used connectivity graph representation of a binary image and developed an algorithm for computing the Euler number. The connectivity graph was derived from the discrete version of the cylindrical algebraic decomposition of the digital plane. The authors also described a parallel implementation of the same algorithm on a linear array network topology.

Rosenfeld and Kak [18] observed that Euler number can be computed from the run length representation of an image. If for each run r, $k(r)$ be the number of runs on the preceding row to which r is adjacent, then Euler number can be expressed as: $E = \sum_r (1 - k(r))$. Further results on 2D and 3D images are reported in [19]. Di Zenzo et al. [20] suggested a planar graph representation of an image from run descriptions. Next, the number of connected components C in the image is computed by applying a standard graph algorithm. The number of holes H is then computed from the Euler's

formula of a planar graph as $H = 1 + m - v$, where m = number of edges and v = number of nodes in the graph. Finally, Euler number is calculated as $E = C - H$. Dey et al. [21] have reported a divide-and-conquer algorithm that can be parallelized for computing the Euler number of a binary image.

In this work, we revisit the run-based expression of Rosenfeld and Kak [18], and present a novel hardware design for computing Euler number. We show that certain properties of runs and neighboring runs and their distributions in the pixel matrix can be exploited to compute the Euler number of a binary image very efficiently. Performance analysis of the algorithm indicates that the proposed technique outperforms significantly in speed, the existing bit-quad counting and quadtree based methods [2,11,13–15]. Experimental results on a logo database show very favorable results. This run-based algorithm, with its inherent parallelism, provides the basis of building a pipeline architecture for on-chip computation of Euler number. Given a pixel matrix of size $(N \times N)$ the upper and lower bounds on the value of the Euler number are also derived. This result allows us to design the architecture correctly. We further derive analytical expressions for performance measures of the proposed architecture to establish its efficiency. It is also shown that the circuit, with minor modifications, can handle a pixel matrix of arbitrary size. A standard cell based VLSI implementation of the architecture on a real technology is described and relevant data on circuit area and delay are reported. To the best of our knowledge, on-chip design of any run-based technique does not seem to be available to date.

The rest of the paper is organized as follows. Section 2 presents the formulation for computing Euler number based on runs, theoretical analysis of complexities, and experimental results on run distributions. Next in Section 3, a simple parallel version of the algorithm, the proposed pipeline architecture, and results on its performance analysis are reported. Section 4 describes VLSI implementation of the design for a 256×256 image. Section 5 concludes the paper.

2. Proposed algorithm

2.1. Theme

Let the binary image be represented by a 0–1 pixel matrix of size $(N \times M)$, 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. The sets referred to in the definition are sets contained in the image. A *run* in any row (or column) of the pixel matrix is defined to be a maximal sequence of consecutive 1's in that row (or column). Let $R(i)$ denote the number of such runs in the ith row (column). $R(i)$ can be counted as the number of $0 \rightarrow 1$ transitions in that row with a 0 padded at the start of the row (the 0 is to be padded to handle the case where there is a 1 at the start of the row). See Figs. 1 and 2 for illustration. There can be no holes in a single row, and the number of connected components in that row is same as the number of runs in that 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* local additive 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* [11,12]).

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 gener-

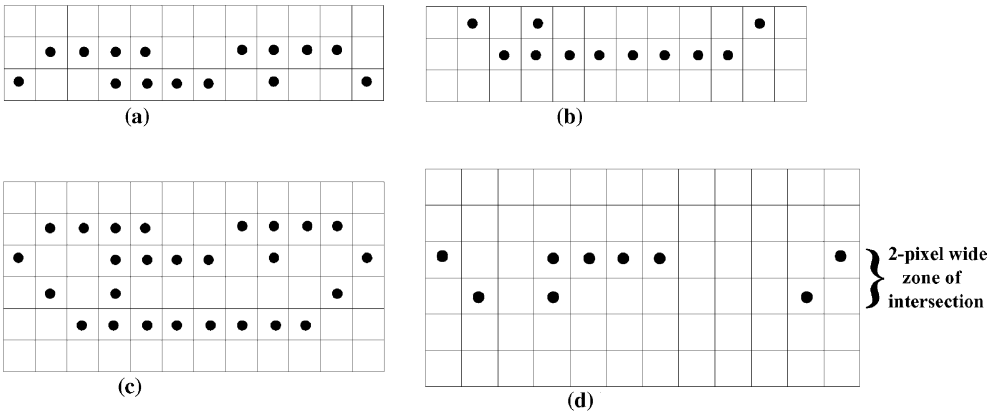


Fig. 1. Union and intersection of images. (a) Image I_1, (b) image I_2, (c) image $I_1 \cup I_2$, and (d) image $I_1 \cap I_2$.

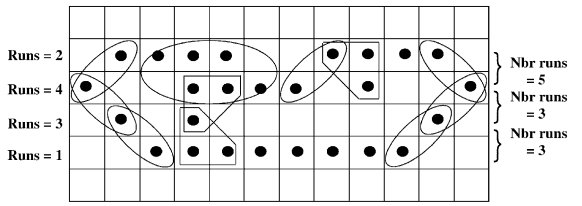


Fig. 2. Illustration of runs and neighboring runs.

ality, 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. See Fig. 1 for an example. 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 is a two-row wide image. See Fig. 1 for an example. No holes can be present in a two-row wide image, and also the number of connected components in a two-row wide image is the number of neighboring runs. So, 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 1–3 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 ith row. Let the number of neighboring runs between them be O_i. Hence,

$$\begin{aligned}
E(I_1) &= R(1),\\
E(I_2) &= E(I_1) + E(2) - O_2\\
&= R(1) + R(2) - O_2,\\
E(I_3) &= E(I_2) + E(3) - O_3\\
&= R(1) + R(2) + R(3) - (O_2 + O_3),\\
&\ldots\\
E(I_N) &= E(I_{N-1}) + E(N) - O_i\\
&= (R(1) + R(2) + R(3) + \cdots + R(N))\\
&\quad - (O_2 + O_3 + \cdots + O_N),\\
&= \sum_{i=1}^{N} R(i) - \sum_{i=2}^{N} O_i,
\end{aligned}$$

where, I_N denotes the entire image.

The above analysis proves the following known result [18], and provides the basis of our hardware implementation.

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 neighboring runs between all consecutive pairs of rows* (*or columns*)*.*

2.2. Computation of run and neighboring run

A run of a row is a sequence of consecutive 1-pixels in that row. See Fig. 2 for an example.

Lemma 1. *The number of runs in any row is counted as the number of* $0 \rightarrow 1$ *transitions in the row with an additional* 0 *padded at the start.*

Proof. The proof is trivial. □

Two runs in two adjacent rows are neighboring if at least one pixel of a run in a row is in 8-neighborhood of a pixel of a run in the other row. To count the number of neighboring runs, the occurrence of the start of a neighboring run is to be determined. A neighboring run between two rows can start only when there is an occurrence of a run ($0 \rightarrow 1$ transition) in at least one of the two rows, and there is at least one 1-pixel in 8-neighborhood of the location where the run starts.

Lemma 2. *A neighboring run between two rows occurs if and only if there is a* $0 \rightarrow 1$ *transition in at least one of the two rows and there is a* 1 *pixel in the* 8*-neighborhood of that location.*

Proof. A neighboring run cannot occur without a run in any of the two rows. Two such runs should have at least one 1 pixel in their neighborhood following 8-neighborhood relations. To count the number of neighboring runs between the $(i-1)$th and ith rows, we have to do the following. A run starts in any jth column of the ith row, if there is a 0 in the $(j-1)$th column and 1 in the jth column of the ith row. A run ends in any jth column of the ith row, if there is a 1 in the $(j-1)$th column and 0 in the jth column of the ith row. The run start location s_j and run end location e_j are stored while counting the number of runs in the ith row. When a run occurs in any kth location in $(i+1)$th row, if the pixel $(i+1,k)$ is in the 8 neighborhood of any of the pixels $(i,s_j),\ldots,(i,e_j)$, then a neighboring run count is increased by one. □

See Fig. 2 for an illustration of the above lemma.

2.3. 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 ith row;
    if (there are runs between the $(i-1)$th and ith rows)
      (comments: see Lemma 2)
      calculate the number of neighboring runs O_i between $(i-1)$th and ith rows;
    endif
    (see Section 2.2)
    $E(I) = E(I) + R(i) - O_i$;
endfor
return $E(I)$ as the Euler number.

2.4. Space and time complexity

The lower bound of computation of Euler number is $\Omega(N^2)$ by definition of the Euler number. In that sense, most of the algorithms are asymptotically optimal in that they compute Euler number in $O(N^2)$. But in image processing applications where huge data is involved and real time application is needed, the constant so often hidden in the O-notation becomes important. This has motivated several algorithms for computation of Euler number. So, we find out the exact number of image pixel matrix accesses for an $N \times N$ matrix under the sequential *RAM* model for the previous algorithms and compare it to the run-based method.

The bit-quad counting technique [2,11] checks for bit-quads (i.e. 4 pixels) for each entry and also has to check for the convexity and concavity along the borders. This takes $4N^2 + 4N$ accesses. Note that, irrespective of the matrix entries, these number of pixel accesses are required. So, the average case access would also be the same.

The Euler number computation based on the thinned version of the image [16] checks for two types of terminal points. For that, 8 neighbors of each pixel is to be accessed. That takes $8N^2$, add to that the time thinning takes and obviously this algorithm takes more pixel accesses than the bit-quad counting technique [2,11].

The Euler number computation based on the quad-tree [14,15] involves computing the quad-tree from the pixel entries followed by a traversal on the quad-tree. The time complexity $T(n)$ for formation of the quad-tree involves the following recurrence:

$$T(n) = \begin{cases} 1 & \text{if } n = 1; \\ 4T\left(\frac{n}{4}\right) + n & \text{if } n \geqslant 2. \end{cases}$$

Solving this recurrence, $T(n) = n\log_4 n + n$. In our case $n = N^2$, so the number of accesses is $N^2\log_2 N + N^2$. Now, traversal of a quad-tree for computing the Euler number takes linear time in the number of leaf nodes in the quad-tree [14,15]. The number of leaf nodes can be N^2. So, Euler number computation takes $N^2\log_2 N + N^2 + cN^2$. This is obviously greater than the accesses required by the bit-quad counting technique.

The above analysis shows that the bit-quad counting algorithm is the best in terms of less number of pixel accesses. That might be the reason of the commercial image processing toolbox MATLAB using this algorithm [13].

Next, we compute the worst case and average case analysis of the proposed run-based technique to show its efficacy.

We require $N \times N$ space to store all the pixels of the given image. To calculate the number of neighboring runs between two consecutive rows, we need to store for each run, the column numbers where a run starts or terminates. The worst case arises in a checkerboard situation when a row has the maximum number of runs, i.e., when every alternating pixel is an object pixel. For N columns, the maximum number of runs in a row is $N/2$. 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 = $(N \times N) + 4 \times (N/2) \approx O(N^2)$, i.e. linear in the number of pixels.

Computation of runs for all the rows needs $2 \times (N \times N)$ pixel accesses. The number of runs $R(i)$ in a row in the worst case is $N/2$. The number of pixel accesses required to check and store the two end points of all runs present in the matrix = $P_R = 2 \times \sum_{i=1}^{N} R(i) = N^2$. Checking whether a run in row $(i-1)$ is in the neighborhood of a run of row i, requires 2 accesses. Also, a neighboring run is to be checked only after the occurrence of a run. So, determination of neighboring runs for all consecutive pairs of rows needs = $2 \times \sum_{i=2}^{N}\sum_{i=1}^{R(i)} O_i = N \times (N-1) = (N^2)/2 - (N/2)$ accesses. Note that, the upper limit on the summation of O_i is $R(i)$ (due to Lemma 2) and not N. Therefore, the total number of accesses is, $2N^2 + N^2 + (N^2 - N) = 4N^2 - N$.

For the average case analysis, we assume a random input of 0 and 1 with a probability of $\frac{1}{2}$ each. As the number of accesses depend on the number of runs, we compute the expected number of runs in a row of width N. Let X_t be the number of 1 runs in the first t entries and Y_t be the number of 0 or 1 runs in the first t entries. Obviously, $E(X_t) = E(Y_t)/2$, where $E(X)$ denotes the expectation of X. Now, the conditional expectation of Y_{t+1} when $Y_t = c$ is $E(Y_{t+1}|Y_t = c) = c + \frac{1}{2}$, because a 0 or 1 can occur with probability $\frac{1}{2}$ in the $(t+1)$th entry. Now, $E(Y_{t+1}) = \sum_c \left(\frac{1}{2}cP(Y_t = c) + \frac{1}{2}(c+1)P(Y_t = c)\right) = \sum_c cP(Y_t = c) + \frac{1}{2}\sum_c P(Y_t = c) = E(Y_t) + \frac{1}{2}$. Now, surely $E(Y_1) = 1$ because either there is a 0 or 1. Therefore, according to the recurrence just deduced $E(Y_N) = \frac{N+1}{2}$. So, $E(X_N) = \frac{E(Y_N)}{2} = \frac{N+1}{4}$. As an example, for $N = 4$, there are $2^4 = 16$ possible entries of 0 and 1 combination. The frequency of a single run is 10 and the frequency of two runs is 5. So, the expected value of 1 runs is $\frac{10\times1+2\times5}{16} = \frac{5}{4}$. So, the number of accesses in the expected case is $2N^2 + 2N\left(\frac{N+1}{4}\right) + 2(N-1)\left(\frac{N+1}{4}\right) = 3N^2 + \frac{N}{2} - \frac{1}{2}$. Thus, it can be seen that the proposed method of Euler number computation requires fewer number of accesses than the bit-quad counting method.

In the proposed method, the number of pixel accesses depends on the distribution of the runs in the pixel matrix in contrast to the method in

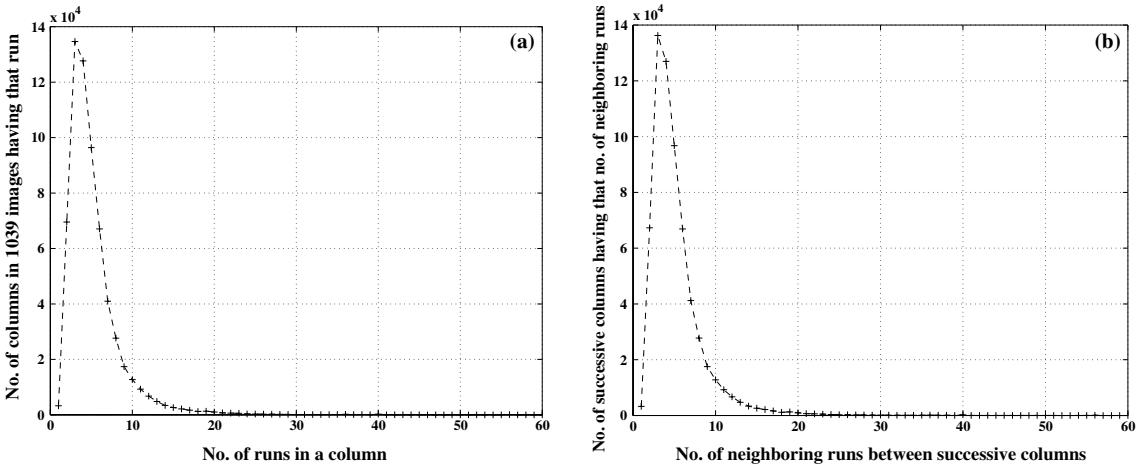


Fig. 3. Graphs showing (a) run and (b) neighboring run statistics.

[11]. 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 other methods. Empirical evidence justifies the rationale. 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 (see Fig. 3(a) and (b)). Thus, the expected number of runs in actual images is nearly constant and much less than the expected case of $\frac{N+1}{4}$ runs. Therefore, the number of accesses would be much less than that of the average case value. The above analytical expressions and experimental validation suggest that the proposed method of computing Euler number will outperform the existing techniques significantly.

We observed that the average CPU time required for computing the Euler number of a 256×256 logo image by running the proposed algorithm is around 11×10^3 μs on a Sun Ultra-5_10 Sparc workstation (233 MHz) with SunOS Release 5.7 Generic OS. On the same platform, the bit-quad based algorithm [2,11] takes around 19.5×10^3 μs.

3. Hardware implementation

Euler number satisfies the local additive property or additive set property [11]. This property allows us to split the image into smaller subimages, and calculate the overall Euler number by combining the Euler numbers of the smaller subimages. An essential feature common to most of these methods is the addition and subtraction of the Euler numbers or some other parameters of the subimages [3,11,16–18,21]. Thus for fast computation of Euler number, the design of the adder is most vital. It may be noted that once the local property counting has been completed involving a set of pixels, the same pixels should not be required again to reduce CPU time. This indicates that in terms of precedence relations the addition process follows the local property counting and it can be overlapped with the next set of local property counting involving another set of pixels. This observation suggests that addition can be pipelined with the local property counting. In this section, the hardware design is reported. We also derive the upper and lower bounds of Euler number, which lead to estimation of the time complexity as well as the overall gate count. The design proposed here is then compared to other methods. The run-based sequential algorithm can be easily parallelized

soas to make it suitable for hardware implementation.

3.1. Parallel algorithm

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 ith row;
(by Fact 1, run of each row can be computed independently)
calculate the number of neighboring runs O_i between $(i-1)$th and ith rows; (by Fact 3)
endfor
$E(I) = \sum_{i=1}^{N} R(i) - \sum_{i=2}^{N} O_i$; (by Theorem 1)
return $E(I)$ as the Euler number.

3.2. Hardware architecture

To implement the proposed algorithm in hardware, two types of processing elements (PE) are required: P_1 for identifying the start of a run in a row; P_2 to signal the start of a neighboring run.

Using Lemma 1, the PE P_1 (shown in Fig. 4(a)) is used to detect the transition from 0 to 1; the number of runs is equal to the number of such detections. A delay (D) flip-flop is initialized to 0 at the start 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 ith pixel and $(i-1)$th pixel of a row are checked for a $0 \rightarrow 1$ transition. The maximum number of runs in a row having M columns is $\lceil (M/2) \rceil$.

In Fig. 4(b), P_1 is shown within a box, and the remaining portion of the circuit constitutes the processing element P_2. It is used to detect the start of a neighboring run between two consecutive rows following Lemma 2. The PE P_2 checks for the condition when a 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, ith and $(i-1)$th pixels of two consecutive rows are checked for a neighboring run. The maximum number of neighboring runs is $2 \times \lceil (M/2) \rceil$. To process an $(N \times M)$ image in parallel, we require N pieces of P_1, and $(N-1)$ pieces of P_2.

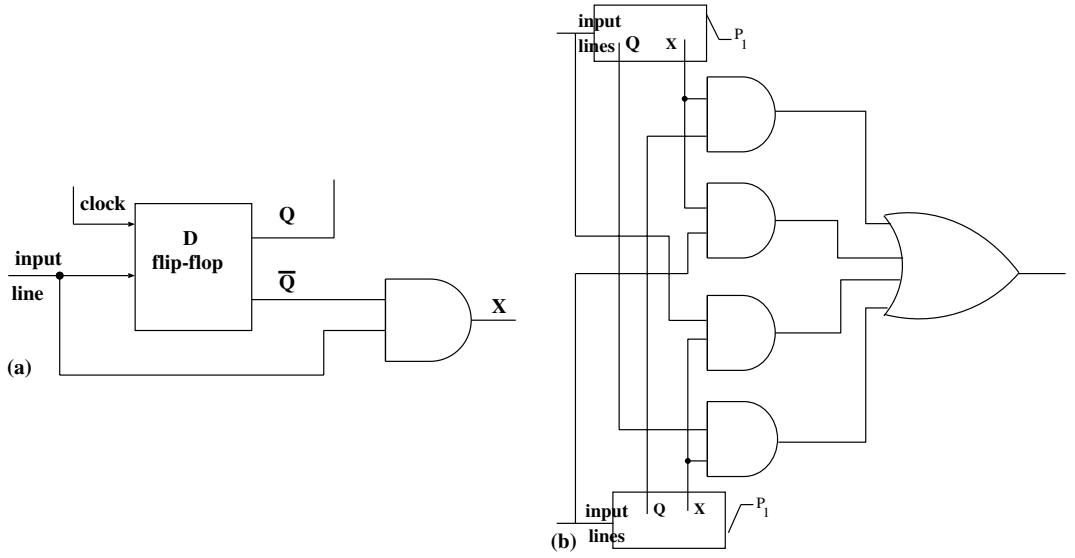


Fig. 4. Processing elements P_1 (a) and P_2 (b).

Table 1
Truth table

	P_1	P_2	p_1	p_2	$(P_1 + p_1) - (P_2 + p_2)$	Sign-bit (s)	Data-bit (d)
1	0	0	0	0	0	0	0
2	0	0	0	1	−1	1	1
3	0	0	1	0	1	0	1
4	0	0	1	1	0	0	0
5	0	1	0	0	×	×	×
6	0	1	0	1	×	×	×
7	0	1	1	0	0	0	0
8	0	1	1	1	−1	1	1
9	1	0	0	0	1	0	1
10	1	0	0	1	×	×	×
11	1	0	1	0	×	×	×
12	1	0	1	1	×	×	×
13	1	1	0	0	0	0	0
14	1	1	0	1	−1	1	1
15	1	1	1	0	1	0	1
16	1	1	1	1	0	0	0

To compute Euler number of the whole image, we add the number of runs (calculated by summing up all $0 \rightarrow 1$ transitions in all rows) and deduct from it the number of neighboring runs (calculated by adding up the number of all neighboring runs between consecutive rows). At any instant of time t_i, the output from P_1 or P_2 goes high or low indicating the start of a run or neighboring run respectively. A neighboring run implies the presence of a run in either or both of the corresponding row(s). Thus, a high (1) output from P_2 is accompanied by a high output from either one or both of the P_1's connected to it. The interrelation of runs and neighboring runs is depicted as a truth table in Table 1, where × represents a don't care entry.

3.3. Truth table and combinational circuit

In order to perform the addition process efficiently, we combine the local results in such a fashion so that the final subtraction can be avoided and a single adder tree can be designed to output the final result.

We take the outputs from the two modules generating P_1 and two modules producing P_2 for consecutive rows. To distinguish them, we use upper and lower case symbols (P_1, p_1, P_2, p_2) in Table 1. The sum of the two P_1 outputs is computed, and the sum of the two P_2 outputs is then subtracted from it. It is easy to prove that the result is always either −1, or 0, or 1. This follows from the properties of runs and neighboring runs as mentioned in the previous subsection. The entries corresponding to rows 5 and 6 in the truth table are not feasible as $P_2 = 1$ implies that either or both of P_1 and p_1 must be 1. The case as in row 10 does not appear as $p_2 = 1$ and $p_1 = 0$ implies the PE P'_1 associated with p_2 (and not included in this group of 4 P_1's and P_2's, and as such not shown in the truth table) should be high. If P'_1 and $p_2 = 1$, it implies the presence of a continuing run (not a run start) in the row associated with p_1; otherwise p_2 cannot be-

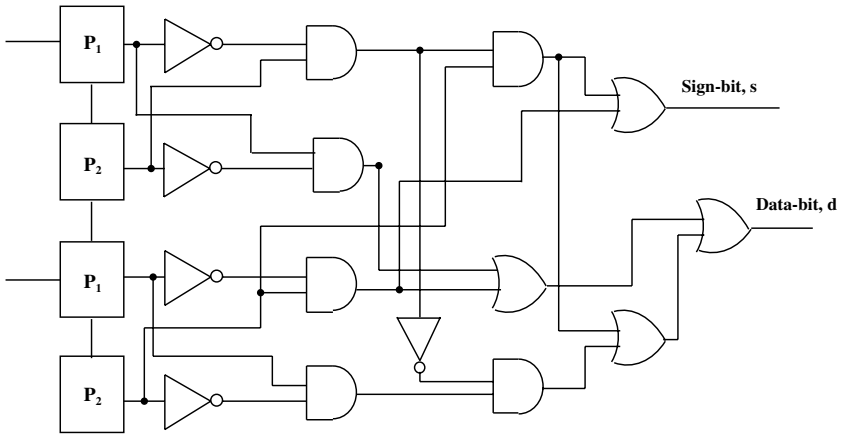


Fig. 5. Combinational circuit for finding the sign and data bit.

come high. If there exists a continuing run associated with p_1 and also $P_1 = 1$, then P_2 should be high, but that does not happen. Hence, this input combination never arises. Rows 11 and 12 are also inadmissible as in both the cases, $P_1 = 1$ and $p_1 = 1$ imply that $P_2 = 1$, which is not true. To represent -1, 0, 1 in 2′s complement, we require 2 bits (the sign bit and data bit). A combinational circuit C as shown in Fig. 5, is designed following the truth table (Table 1) to produce the sign and data bits from different input combinations of the two P'_1s and P'_2s.

3.4. Adder design and time complexity

The sum of the combination of the outputs from the two modules generating P_1 and two modules generating P_2 for consecutive rows lies in the range $[-1, 1]$. To design the adder circuit for Euler number computation, it is essential to determine the range of values Euler number can assume for an $(N \times M)$ image, so that the width of the adders can be properly determined.

Lemma 3. *The Euler number of an* $(N \times M)$ *binary image lies in the range* $[-M/2 \times \lceil (N/2) \rceil, M/2 \times \lceil (N/2) \rceil]$.

Proof. For an $N \times M$ image, we require $\lceil (N/2) \rceil$ pieces of combinational module C. Each C-module produces an output in the range $[-1, 1]$. Thus, each column can have a maximum sum of $\lceil (N/2) \rceil$ and a minimum sum of $-\lceil (N/2) \rceil$. Since a run and a neighboring run for a row cannot begin in two consecutive columns, the Euler number should lie in the range $[-M/2 \times \lceil (N/2) \rceil, M/2 \times \lceil (N/2) \rceil]$. The adders should be so designed that they are able to handle this range of numbers. Alternatively, it can also be proved from the truth table entries as given in Table 1. The truth table entries that give rise to a sum of 1 are 3, 9 and 15 of Table 1; the entries that give rise to a sum of -1 are 2, 8 and 14 of Table 1. To prove that the range of Euler number for an $(N \times M)$ image lies in $[-M/2 \times \lceil (N/2) \rceil, M/2 \times \lceil (N/2) \rceil]$, it suffices to show that if any one of the entries 3, 9 and 15 of Table 1 occurs at any clock instant, the other two entries cannot occur at the next clock instant. Similar cases arise for the entries 2, 8 and 14 of Table 1. If otherwise, then the sum would not lie in the range $[-M/2 \times \lceil (N/2) \rceil, M/2 \times \lceil (N/2) \rceil]$. If the entry 3 occurred at any clock instant t_i, then for p_1 (see Table 1) to have a high (1) output, a 1 was fired at time instance t_i and a 0 at t_{i-1}, for a run ($0 \rightarrow 1$ transition) to be detected. At time instance t_{i+1}, the 9th or 15th entry of the Table 1 cannot occur. If the 9th entry occurred, there should be an occurrence of a neighboring run, because with p_1 going high at time instance t_i and P_1 going high at time instance t_{i+1}, P_2 should also go high at time instance t_{i+1}. So, the 9th entry should have been as $P_1 = 1$, $P_2 = 1$, $p_1 = 0$ and $p_2 = 0$. So, there is a contradiction, and as such the 9th entry cannot occur after the 3rd entry. Similarly, it can be shown that the pairs (3, 15), (9, 15), (2, 8), (2, 14) and (8, 14) cannot occur at consecutive time instances. □

The complete hardware design for a (16×16) image is shown in Fig. 7. The output of C represents a 2-bit 2′s complement number generated by two P'_1s and two P'_2s. An adder circuit is needed to sum up the outputs of all these C-modules to obtain the final result. We use a binary adder tree in pipeline [22] to accelerate the addition process. Addition of two 2′s complement numbers may produce a 3-bit number. To implement such a scheme, we use at the leaf level of the adder tree, a set of 3-bit adders each with a sign bit extension. For each subsequent level in the tree, the adder size (width) is increased by one bit. The depth of the adder tree would be $\lceil \log_2 N \rceil - 1$. As the width of the adder in the tree increases by 1 at each level, the width of the adder A_r at the root, would be $= (3 - 1) + \lceil \log_2 N \rceil - 1 = \lceil \log_2 N \rceil + 1$. Finally, a sequential full adder FA is used to accumulate the sum. The range of numbers FA should be able to handle is $[-M/2 \times \lceil (N/2) \rceil, M/2 \times \lceil (N/2) \rceil]$ (see Lemma 3) Thus, the number of bits T required for the adder FA would be $= \lceil \log_2 \{M \times \lceil (N/2) \rceil + 1\} \rceil$. With the assumption that $M \approx N$, T is $O(\log_2 N)$. The number of the stages of the pipeline is two more (one for the PEs P_1, P_2 and C and the other for the adder FA) than the depth of the adder tree. Therefore, the stages of the pipeline equals $(\lceil \log_2 N \rceil - 1) + 2 =$

$\lceil \log_2 N \rceil + 1$. The clock period of the linear pipeline is determined by the sum of the longest pipeline stage and the delay of the latches, and hence is equal to $T + \delta$, where T is the time taken by the adder FA and δ is the time delay of the latches. The number of clock cycles required by the linear pipeline to perform the entire addition is $(\lceil \log_2 N \rceil + 1) + (M - 1) = \lceil \log_2 N \rceil + M$. Therefore, total time needed to produce the final output in 2′s complement form is $(T + \delta) \times \{M + \lceil \log_2 N \rceil\}$ and M being approximately equal to N, the time taken is $O(N \log N + 2 \times \log_2^2 N) \approx O(N \log N)$. The performance measures e.g., speed up, efficiency and throughput [22] for the linear pipeline designed here are given below.

Speed-up, S_k: The speed-up is $nk/k + (n - 1)$, where n is the number of tasks and k is the number of stages of the pipeline. So, in our case for an $(N \times N)$ image,

$$S_k = \frac{N(\lceil \log_2 N \rceil + 1)}{(\lceil \log_2 N \rceil + 1) + (N - 1)} = \frac{1 + \lceil \log_2 N \rceil}{1 + \frac{\lceil \log_2 N \rceil}{N}}.$$

Efficiency, η: The efficiency is the ratio of speed-up and the number of stages. So, here the efficiency is $\frac{1}{1+\frac{\lceil \log_2 N \rceil}{N}}$.

Throughput, w: The throughput is defined as the number of tasks that the pipeline can complete per unit time and is the ratio of the efficiency and the clock period of the pipeline. Thus,

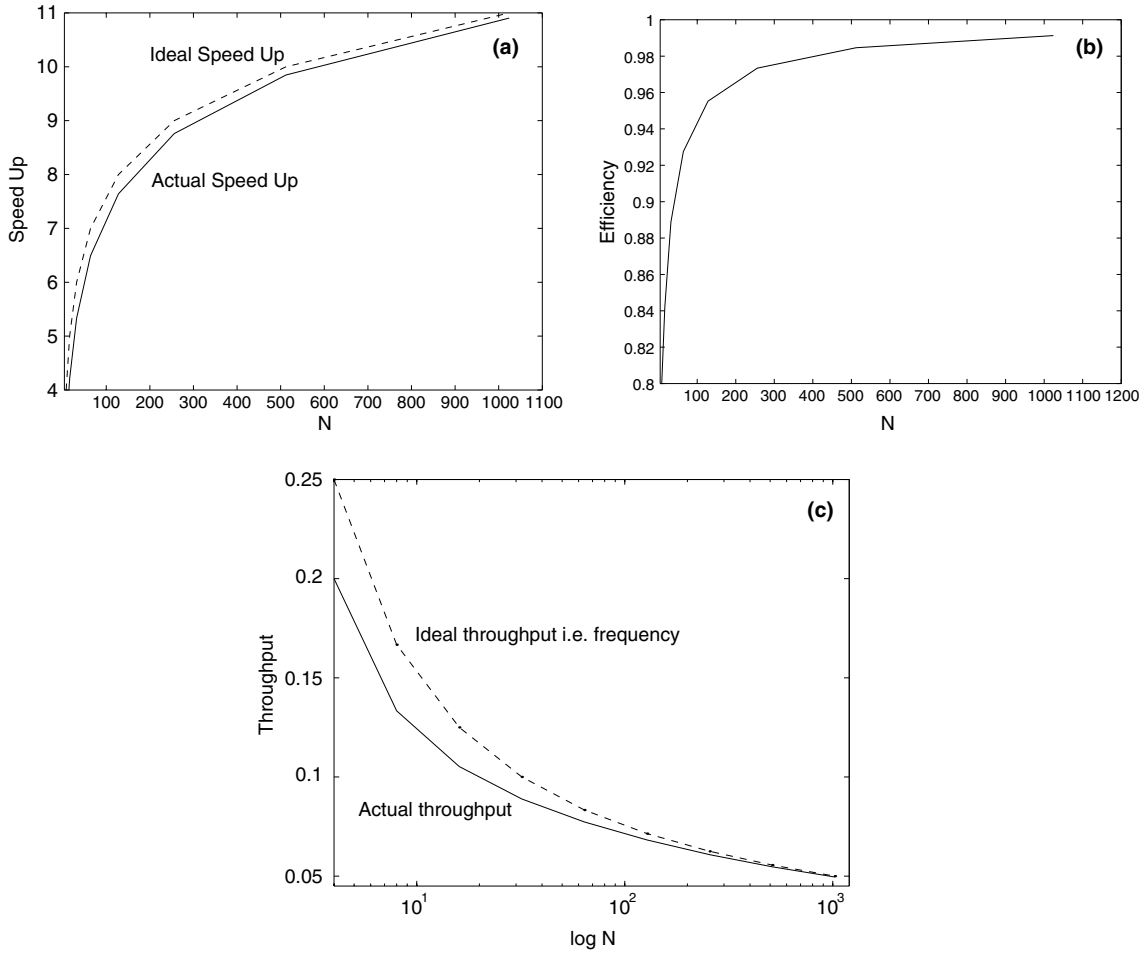


Fig. 6. Performance measures of the pipeline. (a) Speed-up of the pipeline, (b) efficiency of the pipeline, and (c) throughput of the pipeline.

$$w = \frac{\eta}{T+\delta} = \frac{1}{(\lceil \log_2\{N \times \lceil (N/2) \rceil + 1\} \rceil)\left(1 + \frac{\lceil \log_2 N \rceil}{N}\right)},$$

neglecting δ.

The graphs in Fig. 6(a)–(c) show speed-up, efficiency, and throughput, with the variations in the image size N respectively.

If the final adder FA performs a carry look-ahead addition [23] taking $O(\log T)$ time, then the clock period of the pipeline can further be reduced and would be determined by the delay of A_r.

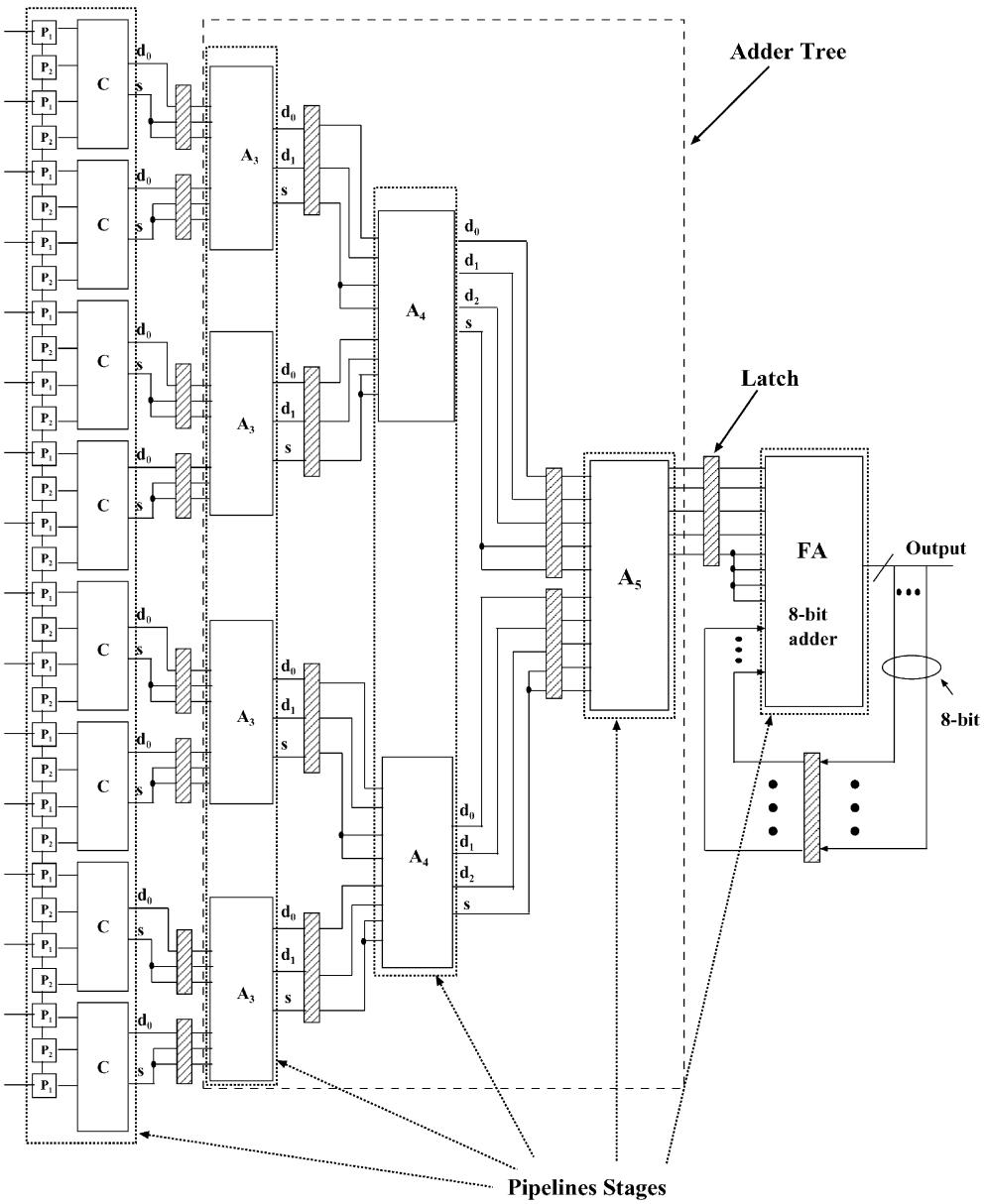


Fig. 7. A sample circuit for calculating the Euler number of a (16×16) image.

3.5. Circuit cost and gate count

We require the following components to implement the hardware architecture for processing an $(N \times M)$ binary image:

(1) N pieces of processing element P_1, each having one edge triggered D-flip-flop and one 2-input AND gate. The gate count is $O(N)$, with constant number of gates required for each P_1 and D-flip-flop.
(2) $N-1$ pieces of processing element P_2, each having four 2-input AND gates and one 4-input OR gate. Here, also the gate count is $O(N)$.
(3) $\lceil (N/2) \rceil$ pieces of the combinational circuit C, each having four inverters, six 2-input AND gates and five 2-input OR gates. As each piece of combinational circuit C requires constant number of gates, the gate requirement is $O(N)$.
(4) $\lceil (N/4) \rceil$ pieces of 3-bit full adder, $\lceil (N/8) \rceil$ pieces of 4-bit full adder, ..., one piece of $\lceil \log_2 N \rceil$-bit full adder. The full adders require constant number of gates for each full bit adder. So, the gate count is of the order of the following:

$$\begin{aligned} &3\lceil (N/4) \rceil + 4\lceil (N/8) \rceil + \cdots + \log_2 N \\ &\quad \approx \frac{3N}{2^2} + \frac{4N}{2^3} + \cdots + \log_2 N \frac{N}{2^{\log_2 N}} \\ &\quad \approx \frac{13N-4}{16} + \frac{\log_2 N}{2} = O(N). \end{aligned}$$

(5) One piece of $\lceil \log_2 \{ M \times \lceil (N/2) \rceil + 1 \} \rceil$-bit full adder. The gate count is $O(N)$.
(6) Latches between the two adjacent stages of the pipeline. The total gate count = $\frac{6N}{4} + \frac{8N}{8} + \cdots + \lceil \log_2 N \rceil + 1 + T \approx O(N)$.

Hence, the total gate count is $O(N)$.

The complete circuit for a (16×16) image with appropriate adder blocks is shown in Fig. 7.

3.6. Comparisons with other methods

The various formulations of computing Euler number require additions of two or more properties and finally their subtraction. The formulation in [11] given as $= (\sum v - \sum t - 2 \times \sum d)/4$ requires addition and subtraction of three properties viz. v, t and d, pertaining to bit-quads Q_1, Q_2 and Q_D. Three types of processing elements would be required to detect bit-quads Q_1, Q_2 and Q_D. Also, adder complexity would be higher as more variables are to be dealt with in comparison to the run based method. The formulation has a divide-by-4 operation. So, it is obvious that the width of the adder would be more and shift registers will be needed for the division operator. This bit-quad formulation for computing the Euler number has also another drawback regarding border pixels. Any implementation of this bit-quad algorithm has to deal with the convexity on the border by specially employing separate processing elements for the border pixels other than the ones required for bit-quad checking. The parallel implementation based on Euler number computation of the thinned version [16] does not hold promise as parallel thinning in itself is a non-trivial task [24]. The hardware implementation of quad-tree based algorithms [14,15] are complicated as the sizes of the blocks represented by the leaf nodes may be unequal. Further, the number of leaf nodes may vary widely for different image samples. The best known parallel algorithm has been proposed in [17] that uses the connectivity graph (CG) derived from the cylindrical algebraic decomposition of the Euclidean plane. The authors describe a parallel implementation of their algorithm on a linear array network topology that uses the CG as the image data structure and performs a parallel searching of the sub graphs in CG. The estimated time complexity is $O((\lceil M/5 \rceil/16)(1 + \log_2 M))$, where M is the number of arcs in the CG. A careful analysis reveals that the number of arcs M in CG can be linear in the number of pixel entries (which is N^2) i.e., M can be $O(N^2)$ and so does the time complexity. Our proposed algorithm with $O(N \log N)$ time complexity and $O(N)$ gates thus compares favorably against existing algorithms.

3.7. Handling large images

Given an architecture for computing the Euler number of an image matrix of size $N \times N$, the

Euler number of a larger image of size $K \times K$, where $K = x \times N$ can be easily determined. The matrix is partitioned into several $N \times K$ sub-blocks as $B_1, B_2, \ldots, B_x$, where each B_i is an $N \times K$ matrix. The Euler number of each such block of size $N \times K$ is computed using our proposed architecture for an $N \times N$ block by changing the size of the adder FA. Let E_i be the Euler number of a block B_i.

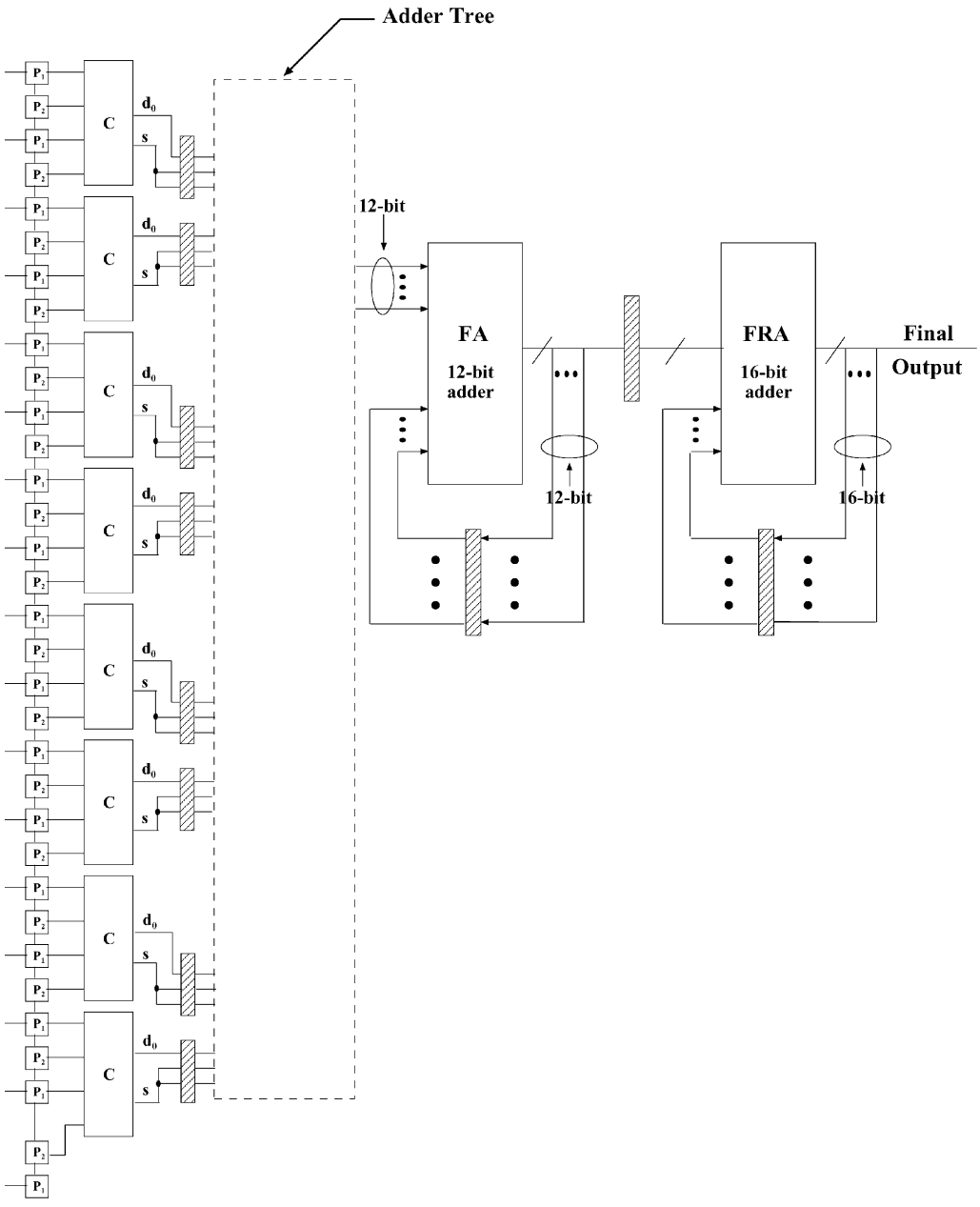


Fig. 8. A scalable circuit for calculating the Euler number of a (256×256) image using the circuit for a (16×16) image.

The Euler number of the overall image can now be easily determined by using the following algorithm based on Facts 2 and 3 as mentioned in Section 2.1.

Method
Input: A $(K \times K)$ binary pixel matrix of an image I, where $K = x \times N$;
Output: Euler number $E(I)$.
Compute_Euler_General
$E(I) = 0$;
$O_i = 0$;
for (i = block number 1 to $x - 1$)
calculate the Euler number, $E(B_i)$ of the block B_i using *Compute_Euler_Parallel*;
calculate the number of neighboring runs O_i between last row of block B_i and first row of block B_{i+1}; (by Fact 3)
$E(I) = E(I) + E(B_i) - O_i$; (by Fact 2)
endfor
compute the Euler number, $E(B_x)$ of the last block B_x using *Compute_Euler_Parallel*;
$E(I) = E(I) + E(B_x)$; (by Fact 2)
return $E(I)$ as the Euler number.

The adder tree can handle an image matrix with N rows, as it deals with only one column at a time. The adder FA adds up values from all columns. When the number of columns changes from N to K, the width W of the adder FA should be made equal to $\lceil \log_2\{K \times \lceil (N/2) \rceil + 1\} \rceil$. To calculate the number of neighboring runs (O_i) between the last row of block B_i and the first row of block B_{i+1}, we require processing elements P_1 and P_2. The output of P_2 is fed as an input of the last C-module in the column (Fig. 8). The outputs of FA corresponding to each block of size $N \times K$ is pipelined to the final sequential adder FRA (Final Root Adder) (see Fig. 8). Using Lemma 3, it can be deduced that the number of bits T required for the adder FRA would be $\lceil \log_2\{K \times \lceil (K/2) \rceil + 1\} \rceil$. The clock period of the linear pipeline is obviously $T + \delta$, where δ is the delay of the latches. The number of stages S_N of the linear pipeline in the scalable circuit is one more than the earlier case and is $(\lceil \log_2 N + 2 \rceil)$. Therefore, the total time needed to compute the Euler number scalably is $(T + \delta) \times (S_N) \approx O(Nx \log_2(Nx))$. The speed-up, efficiency, throughput are as follows:

Speed-up S_k: The number of tasks is obviously now $Kx = \frac{K^2}{N}$ and the number of stages is S_N. So, the speed-up is

$$\frac{\frac{K^2}{N} \times (\log_2 N + 2)}{\lceil \log_2 N \rceil + \frac{K^2}{N} + 1} = \frac{Nx^2(\lceil \log_2 N \rceil + 2)}{\lceil \log_2 N \rceil + Nx^2 + 1}.$$

The ideal speed-up is obviously $\lceil \log_2 N + 2 \rceil$.

Efficiency η: The efficiency is the ratio of speed-up and the number of stages and is $\frac{Nx^2}{\lceil \log_2 N \rceil + Nx^2 + 1}$.

Throughput w: The throughput is

$$\frac{Nx^2}{(\lceil \log_2 N \rceil + Nx^2 + 1)\left(\lceil \log_2\left(Nx \times \left\lceil \frac{Nx}{2} \right\rceil + 1\right) \right\rceil\right)}.$$

It can be seen that for a fixed N, speed-up and efficiency of the pipeline increases with x and throughput decreases with x. This is a desirable feature of the pipeline.

The circuit for computing the Euler number of an image of size (256×256) using the circuit module for a (16×16) image, is shown in Fig. 8. We have assumed sequential full adders for our complexity analysis for both the normal and scalable cases. It can be observed that the time complexity can further be improved by using carry-save addition [23].

4. VLSI implementation of the architecture

The proposed architecture has been designed on-chip using Mentor Graphics Leonardo Spectrum, Modelsim, and IC Station run on a SUN Blade 2000 Workstation. The design is coded with VHDL, and after simulating and verifying it using Modelsim, all the VHDL modules are synthesized to generate the Verilog netlist using 0.18 *micron* technology. The Verilog netlist file is then fed to the tool IC Station, which produces the final chip layout using standard cell design style. All the geometric information of the layout is produced in Graphic Design System-II (GDS-II) code that is needed by foundry for fabrication of the chip. The synthesized netlist report is presented in Table 2 for a 256×256 image. The internal zone area is 400,980.9 μm^2. The overall chip X-dimension is 607.9 μm and the Y-dimension is 671.2 μm. The critical delay in the circuit is 2.29 ns. The on-chip

Table 2
Synthesis report of the circuit for processing a (256×256) image

Number of ports	274
Number of nets	5362
Total number of gates	138,594

time to compute Euler number of a 256×256 binary logo image turns out to be 0.6 μs.

5. Conclusions and discussions

A run-based algorithm for computing the Euler number of a binary image is formulated and its performance is analyzed. The algorithm is based on certain combinatorial and statistical properties of runs present in the pixel matrix of the image. Analytical and experimental studies on a logo database show that the proposed algorithm outperforms existing methods based on bit-quad or quad-tree significantly. A new hardware implementation using pipeline architecture for fast on-chip computation of Euler number is also reported. The hardware design uses $O(N)$ gates to compute the Euler number of an $N \times N$ image in $O(N \log N)$ time. This improves on the best known parallel implementation of $O(N^2)$ on a linear array network topology. The basic module can be used to handle arbitrarily large-sized pixel matrices. The architecture has been implemented in VLSI and relevant chip data on area and speed are reported. It has been observed that the on-chip computation is extremely fast and hence, the design will be useful to many real-time applications. Design of algorithms for applicability to higher dimensions needs further investigation.

Acknowledgment

We would like to thank Prof. Anil K. Jain and Dr. Aditya Vailaya for sending us the logo trademark database. Thanks are also due to Mr. Souradip Sarkar and Mr. Souvik Maity of the Sikkim Manipal Institute of Technology, for their help in implementing the VLSI architecture. The authors wish to thank Dr. Mitsuo Motoki for helpful discussions. We also take this opportunity to thank the reviewers for their constructive and critical comments that helped us to improve the paper significantly.

References

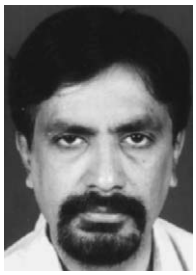
[1] R.C. Gonzalez, R.E. Woods, Digital Image Processing, Addison-Wesley, Reading, MA, 1993.
[2] W.K. Pratt, Digital Image Processing, John Wiley & Sons, 1978.
[3] M.-H. Chen, P.-F. Yan, A fast algorithm to calculate the Euler number for binary image, Pattern Recognition Letters 8 (5) (1988) 295–297.
[4] B.W. Pogue, M.A. Mycek, D. Harper, Image analysis for discrimination of cervical neoplasia, Journal of Biomedical Optics 5 (1) (2000) 72–82.
[5] S.N. Srihari, Document image understanding, in: Proc. ACM/IEEE Joint Fall Computer Conference, 1986.
[6] S.K. Nayar, R.M. Bolle, Reflectance-based object recognition, International Journal of Computer Vision 17 (3) (1996) 219–240.
[7] A.B. Venkatarangan, Geometric and statistical analysis of porous media, Ph.D. Dissertation, Department of Applied Mathematics and Statistics, SUNY at Stony Brook, NY, USA, 2000.
[8] P.L. Rosin, T. Ellis, Image difference threshold strategies and shadow detection, in: Proc. British Machine Vision Conference, 1995, pp. 347–356.
[9] A. Stavrianopoulou, V. Anastassopoulos, The Euler feature vector, in: Proc. Intl. Conf. on Pattern Recognition (ICPR), September, vol. III, IEEE CS Press, Barcelona, Spain, 2000, pp. 7034–7036.
[10] A. Bishnu, B.B. Bhattacharya, M.K. Kundu, C.A. Murthy, T. Acharya, Euler vector: a combinatorial signature for gray-tone images, in: Proc. 3rd Intl. Conf. on Information Technology: Coding and Computing (ITCC), April, IEEE CS Press, Las Vegas, 2002, pp. 121–126.
[11] S.B. Gray, Local properties of binary images in two dimensions, IEEE Transactions on Computers 5 (1971) 551–561.
[12] M. Minsky, S. Papert, Perceptrons, MIT Press, Cambridge, USA, 1968.
[13] <http://www.mathworks.com/access/helpdesk/help/toolbox/images/bweuler.shtml>.
[14] C.R. Dyer, Computing the Euler number of an image from its quadtree, Computer Graphics and Image Processing 13 (3) (1980) 270–276.
[15] H. Samet, H. Tamminen, Computing geometric properties of images represented by linear quadtrees, IEEE Transactions on PAMI PAMI-7 (2) (1985).
[16] L.D.S. Juan, H.S. Juan, On the computation of the Euler number of a binary object, Pattern Recognition 29 (3) (1996) 471–476.
[17] F. Chiavetta, V.D. Gesú, Parallel computation of the Euler number via connectivity graph, Pattern Recognition Letters 14 (11) (1993) 849–859.

[18] A. Rosenfeld, A.C. Kak, Digital Picture Processing, Academic Press Inc., New York, 1982.
[19] C.N. Lee, T. Poston, A. Rosenfeld, Winding and Euler numbers for 2D and 3D digital images, Computer Vision, Graphics and Image Processing 53 (1991) 522–537.
[20] S. Di Zenzo, L. Cinque, S. Levialdi, Run-based algorithms for binary image analysis and processing, IEEE Transactions on PAMI PAMI-18 (1) (1996) 83–89.
[21] S. Dey, B.B. Bhattacharya, M.K. Kundu, T. Acharya, A fast algorithm for computing the euler number of an image and its VLSI implementation, in: Proc. 13th Intl. Conf. on VLSI Design, 2000, 330–335.
[22] K. Hwang, F.A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill International Edition, Singapore, 1985.
[23] F.T. Leighton, Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes, Morgan Kaufmann Publishers, San Mateo, CA, 1992.
[24] F.Y. Shih, C.T. King, C.C. Pu, Pipeline Architectures for Recursive Morphological Operations, IEEE Transactions on Image Processing 4 (1) (1995) 11–18.

Arijit Bishnu received the B.E. degree in electrical engineering from the Burdwan University, India in 1995, M.Tech. degree in computer science and the Ph.D. degree both from the Indian Statistical Institute, Kolkata, India in 1998 and 2003 respectively. Currently he is an Associate in the School of Information Sciences, Japan Advanced Institute of Science and Technology, Japan.

Bhargab B. Bhattacharya received the B.Sc. degree in physics from the Presidency College, Calcutta, the B.Tech. and M.Tech. degrees in radiophysics and electronics, and the Ph.D. degree in computer science all from the University of Calcutta, India. Since 1982, he has been on the faculty of the Indian Statistical Institute, Calcutta, where he is full professor. He visited the Department of Computer Science and Engineering, University of Nebraska—Lincoln, USA, during 1985–1987, and 2001–2002, and the Fault-Tolerant Computing Group, Institute of Informatics, at the University of Potsdam, Germany during 1998–2000. His research interest includes logic synthesis and testing of VLSI circuits, physical design, graph algorithms, and image processing architecture. He has published more than 150 papers in archival journals and refereed conference proceedings, and holds four United States Patents. Currently, he is collaborating with Intel Corporation, USA, and IRISA, France, for development of image processing hardware and reconfigurable parallel computing tools.

Dr. Bhattacharya is a Fellow of the Indian National Academy of Engineering. He served on the conference committees of the International Test Conference (ITC), the Asian Test Symposium (ATS), the VLSI Design and Test Workshop (VDAT), the International Conference on Advanced Computing (ADCOMP), and the International Conference on High-Performance Computing (HiPC). For the International Conference on VLSI Design, he served as Tutorial Co-Chair (1994), Program Co-Chair (1997), General Co-Chair (2000), and as a member of the Steering Committee since 2001. He is on the editorial board of the Journal of Circuits, Systems, and Computers (World Scientific, Singapore), and the Journal of Electronic Testing Theory and Applications (JETTA).



Malay K. Kundu received his B.Tech., M.Tech. and Ph.D. (Tech.) degrees all in radiophysics and electronics from the University of Calcutta. In 1982, he joined the Indian Statistical Institute, Calcutta, as a faculty member. He had been the Head of the Machine Intelligence Unit of the Institute during September 1993 to November 1995, and currently he is full professor at the same unit. His current research interest includes image processing and analysis, image compression, digital watermarking, wavelets, fractals, VLSI design for digital imaging and soft Computing. He received the prestigious VASVIK award for industrial research in Electronic Sciences and Technology for the year 1999 and Sir J.C. Bose memorial award in the year 1986.

He has contributed about 80 research papers in well known and prestigious archival journals, international refereed conferences and as chapters in monographs and edited volumes. He is the holder of four US Patents. He is co-author of the book titled *Soft Computing for Image Processing* published from Physica-Verlag, Heidelberg. He is a fellow of the National Academy of Sciences, India and a fellow of the Institute of Electronics and Telecommunication Engineers, India.

C.A. Murthy was born in Ongole, Andhra Pradesh in 1958. He obtained B.Stat. (Hons.), M.Stat. and Ph.D. degrees from the Indian Statistical Institute (ISI). He visited the Michigan State University, East Lansing in 1991–1992, and the Pennsylvania State University, University Park in 1996–1997. Currently he is a Professor at the Machine Intelligence Unit of ISI. His fields of research interest include pattern recognition, image processing, machine learning, neural networks, fractals, genetic algorithms, wavelets and data min-

ing. He received the best paper award in 1996 in Computer Science from the Institute of Engineers, India. He received the Vasvik award for Electronic Sciences and Technology for the year 1999 along with his two colleagues. He is a fellow of the Indian National Academy of Engineering.

Tinku Acharya is currently Senior Executive Vice President and Chief Science Officer of Avisere Inc., Tucson, Arizona. He is also Adjunct Professor in the Department of Electrical Engineering, Arizona State University, Tempe, Arizona since 1997. He received his B.Sc. (Honours) in physics, B.Tech. and M.Tech. in computer science from the University of Calcutta, India and Ph.D. in computer science from the University of Central Florida, USA.

Dr. Acharya was a Principal Engineer in Intel Corporation, Arizona (1996–2002), a consulting engineer at AT&T Bell Laboratories (1995–1996) in New Jersey, a research faculty member at the Institute of Systems Research, University of Maryland at College Park (1994–1995), and held visiting faculty positions at Indian Institute of Technology (IIT), Kharagpur (on several occasions during 1998–2003). He also served as Systems Analyst in National Informatics Center, Planning Commission, Government of India (1988–1990). He held many other positions in industry and research laboratories.

Dr. Acharya is an inventor of 80 US patents and 15 European patents in diverse areas of data compression, multimedia computing, electronic imaging, VLSI architectures, and more than 50 patents are currently pending in the US Patent Office. He has been awarded the ''Most Prolific Inventor'' in Intel Corporation Worldwide in 1999 and ''Most Prolific Inventor'' in Intel Corporation Arizona site for five consecutive years (1997–2001). He contributed to over 70 refereed technical papers. He is author two books *Data Mining: Multimedia, Soft Computing and Bioinformatics* (*2003*), and *JPEG2000 Standard for Image Compression: Concepts, Algorithms, and VLSI Architectures* (*2004*), published by John Wiley & Sons, New Jersey. He also co-edited the book *Information Technology: Principles and Applications*, published by Prentice-Hall India, New Delhi, 2004.

Dr. Acharya is a Life Fellow of the Institution of Electronics and Telecommunication Engineers (FIETE), and Senior Member of IEEE. He served on the US National Body of JPEG2000 standards committee (1998–2002). He served in program committees of several international conferences and many other professional bodies in academia and industry. His current research interests are in computer vision for enterprise applications, biometrics, multimedia computing, and VLSI architectures and algorithms.