Hindawi
Mathematical Problems in Engineering
Volume 2018, Article ID 5714638, 17 pages
https://doi.org/10.1155/2018/5714638

Research Article

A Parallel Algorithm for the Counting of Ellipses Present in Conglomerates Using GPU

Reyes Yam-Uicab,[1] José López-Martínez,[1] Erika Llanes-Castro,[1] Lizzie Narvaez-Díaz,[1] and Joel Trejo-Sánchez[2]

[1]*Facultad de Matemáticas, Universidad Autónoma de Yucatán, Mérida, YUC, Mexico*
[2]*Centro de Investigación en Matemáticas, Conacyt, Mérida, YUC, Mexico*

Correspondence should be addressed to José López-Martínez; jose.lopez@correo.uady.mx

Received 7 November 2017; Revised 2 February 2018; Accepted 8 March 2018; Published 18 April 2018

Academic Editor: Benjamin Ivorra

Detecting and counting elliptical objects are an interesting problem in digital image processing. There are real-world applications of this problem in various disciplines. Solving this problem is harder when there is occlusion among the elliptical objects, since in general these objects are considered as part of the bigger object (conglomerate). The solution to this problem focusses on the detection and segmentation of the precise number of occluded elliptical objects, while omitting all noninteresting objects. There are a variety of computational approximations that focus on this problem; however, such approximations are not accurate when there is occlusion. This paper presents an algorithm designed to solve this problem, specifically, to detect, segment, and count elliptical objects of a specific size when these are in occlusion with other objects within the conglomerate. Our algorithm deals with a time-consuming combinatorial process. To optimize the execution time of our algorithm, we implemented a parallel GPU version with CUDA-C, which experimentally improved the detection of occluded objects, as well as lowering processing times compared to the sequential version of the method. Comparative test results with another method featured in literature showed improved detection of objects in occlusion when using the proposed parallel method.

1. Introduction

The use of a computer to automate certain tasks originally done by hand has led to, among other clear advantages, increased reliability, reduced cost and time investment, and more precise results. Additionally, the workload performed by users can be reduced and the physical strain that would normally be incurred can be avoided. This study presents the parallelization of a detection method for ellipses that are in occlusion with other objects. This method is very helpful for solving problems and increasing efficiency in a wide range of areas. For example, in microbiology and biomedicine it is common to analyze blood samples by counting cells that are semielliptical in shape [1, 2]. This is a time-consuming manual task that can lead to eyesight problems and inaccurate results over a long period [3]. Several approximations have been presented which try to tackle these specific issues; for example, in [2] a cell detection method is proposed in which initial preprocessing is used to differentiate among objects (in this case, cells) and the image background for which it is necessary to combine an initial group of images using multifocal image fusion techniques. In addition to this first step, a better adjusted ellipse is obtained for each object using a method defined as robust ellipse fitting using a Heteroscedastic Errors-In-Variables (HEIV) regression algorithm. Other applications can be found in computing areas such as intelligent systems to recognize elliptical and semielliptical objects in real contexts, as well as many others. There are numerous methods currently being outlined in literature which aim to solve this general problem using different approximations. For example, in 2015 Bera [4] presented a method which facilitated the detection and counting of circular objects in partial occlusion. The author proposed a methodology based on finding the centroid of the conglomerate and, from this

point, calculating the necessary strategic points on the outline of conglomerate to start the circle detection process via geometric properties. However, the author indicated that the method did not work if the circular objects were in occlusion with other noncircular objects. Also, the method used to obtain the strategic points on the outline can eventually miss certain values. In [5] a similar method was presented for use in the analysis of phytoplankton images. In this case the objective was to count said species (circular in shape) using high resolution images based on a phase of circular object detection, stochastic optimization of the outline of object, and object classification. Similar methods have been proposed in [3, 6], in which the process is carried out by obtaining certain values and applying equations that provide an estimate of the total objects in the conglomerate. Conversely, in [7] a method is presented for the detection of ellipses which is based on sorted merging strategy. The proposed methodology consists of receiving an image of outlines, linking the outlines into a single group of points, dividing this group into small segments, and then combining them to generate possible elliptical arcs or straight lines and adjust all possible ellipses. For the adjustment of ellipses, a least squares method is used to correctly process situations that the proposed methodology generates. The proposed method improves the accuracy of the elliptical object count compared to a similar method presented in the literature [4]. The parallelization of algorithms that involve high computational complexities due to combinatorial processes as is the case of the proposed method can be reduced in time depending on the granularity of parallelism.

In this work, we propose a method to count elliptical objects in a conglomerate. Using two input images (base ellipse and the conglomerate), objects counting is carried out by analysis and adjustment of arcs combination (outline points of conglomerate) to the base ellipse. Since the proposed method possesses a high computational complexity, a parallel implementation is developed using NVIDIA GPU (Graphics Processing Unit) and CUDA (Compute Unified Device Architecture). Computer simulation and experimental results obtained with the proposed method are analyzed in terms of accuracy and speedup. We showed that the proposed parallel method is capable of counting elliptical objects with a good accuracy in real time.

This paper is organized as follows: Section 2 gives a detailed description of the sequential method for counting ellipses in occlusion, as well as a description of its initial order of complexity, Section 3 describes the use of parallel computing with CUDA-C and the GPU, Section 4 describes the parallelization of the sequential method, Section 5 shows the experimental results, and Section 6 presents conclusions.

2. The Sequential Algorithm for Counting Ellipses with Occlusions

In general, the algorithm receives two input images; the first contains an independent ellipse with no occlusion representing the ellipse that the method aims to detect within the conglomerate (this image will be referred to as the base ellipse), and the second image contains the objects in occlusion with each other, that is, the conglomerate. Preprocessing is done on both images and following this (using the information obtained from the preprocessing) a combinatorial process of points and ellipse adjustments is performed to obtain ellipses which could be candidates for detection. We selected, using a defined minimum error, the ellipses found to be similar to the base ellipse. Figure 1 illustrates the methodology of the algorithm for the counting of objects in the presence of occlusion.

The algorithm is divided into seven steps, which are described below.

(1) Obtain the Image. We obtained two images, corresponding, respectively, to the base ellipse and the conglomerate to be analyzed. The algorithm works on the images in black and white; hence the two input images are binarized (without loss of generality we use binarized images as input of our algorithm). To obtain the binary images when we perform real experiments, we use Otsu's method; however, it is possible to use other image binarization techniques [8–11] according to initial conditions of the input real image (e.g., low contrast, low resolution, high luminosity, and shading). Figure 2 shows an example of input to the algorithm. Figure 2(a) shows the base ellipse and Figure 2(b) shows the conglomerate to be analyzed. The base ellipse will be useful in step (4) to determine the possible ellipses present in the conglomerate.

(2) Obtain Singular Concavity Points (SCPs). A *singular concavity point* is a point of intersection or cut-off point between two or more overlapped objects [3]; Figure 3 shows a conglomerate of two objects in which arrows mark the SCPs present. The aim of this step is to obtain such data, and to do so we employed a method proposed in [12] which uses trigonometric properties to detect abrupt changes in the curvature of the outline of conglomerate. The result of this step is the detection of the geometrical positions of the SCPs within the conglomerate. The total number of SCPs will be represented by n (singular concavity points), for future reference.

(3) Arc Generation. In this step, we first detected the outline of the conglomerate using a basic process of mathematical morphology [8]. Next, having established the positions of the SCPs in the conglomerate, we obtained the arcs. An arc is a pixel path that goes from SCP_j to SCP_{j+1} in such a way that it crosses the outline of the conglomerate. The number of arcs will be equal to n, that is, the number of detected SCPs. Figure 4 shows the outline of a conglomerate with arrows showing the arcs present.

(4) Arc Combination. The following is a general description of this step which starts with a combinatorial process among all the obtained arcs, giving in total of $2^n - 1$ necessary combinations, where n is the number of SCPs. Each combination is formed by a group of k points (the sum of the points in each arc which includes the combination). Using the exhaustive method of ellipse adjustment with minimum

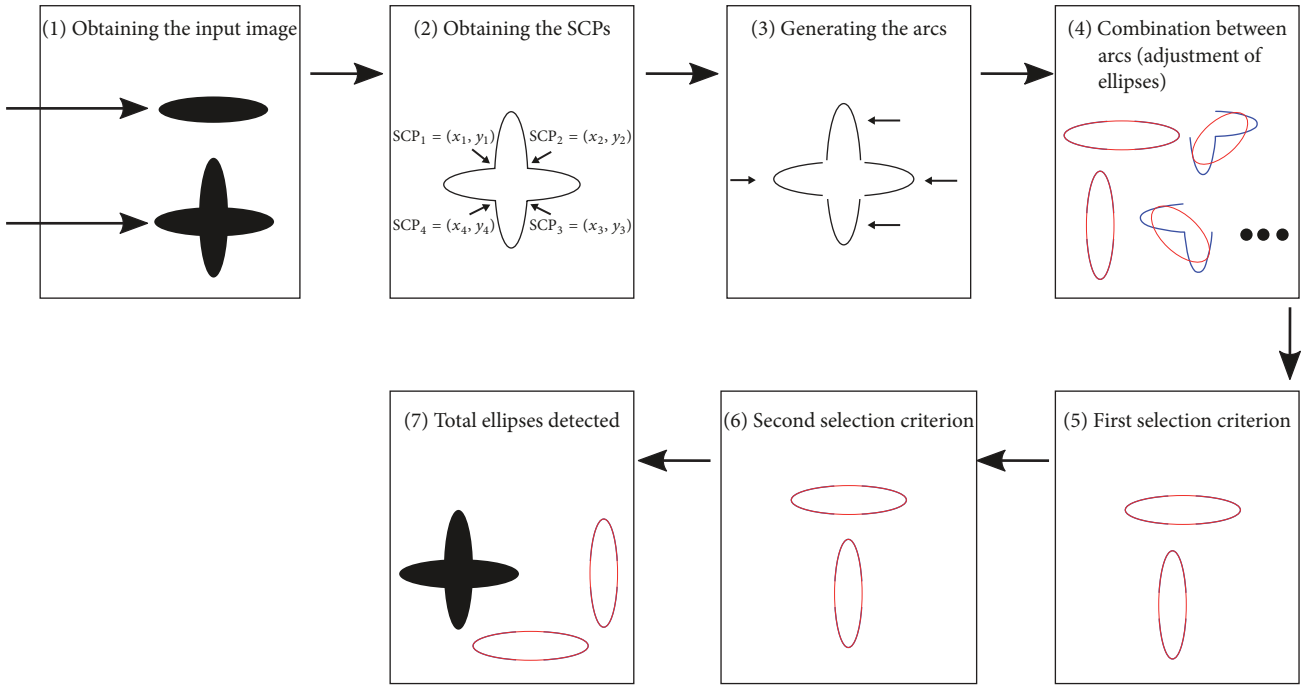


FIGURE 1: Methodology used in the sequential algorithm for the counting of elliptical objects in the presence of occlusion. A conglomerate of two elliptical objects is used.

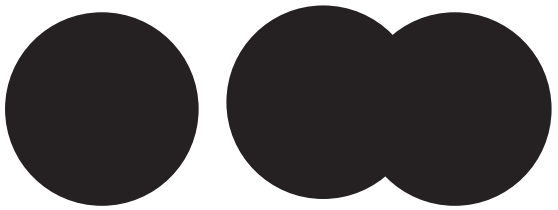


FIGURE 2: Example of input to the algorithm for the counting of ellipses in occlusion. (a) Image of base ellipse. (b) Image of the conglomerate containing two objects for detection.

squares proposed in [13], we obtained the equation for the ellipse best adjusted to the points of the ith combination with $1 \leq i < 2^n$. Each equation is composed of the coordinates of its center (Cx_i, Cy_i), the length of its radii (Rx_i, Ry_i), and the angle θ_i of rotation of the ellipse. Notice that our technique is feasible when the number of conglomerated objects is low. Otherwise, our technique is not feasible, because of the execution time of our algorithm. For such a case, metaheuristics like evolutionary computation are traditionally used to obtain a good solution (not necessarily the optimal) in low execution time. To the best of our knowledge, no evolutionary strategy has been used to objects counting (particularly in arc combination step), when occlusion occurs.

Performing the step described above proves more time-consuming than any other in the whole algorithm, given that it is necessary to iterate $2^n - 1$ times, and for each iteration first the corresponding combination of arcs is generated and then the ellipse best adjusted to the points conforming to the combination is generated. This means it is possible to estimate the total runtime of the algorithm, which is defined specifically as $(2^n - 1) + k(2^n - 1)$, where the first part of the expression $(2^n - 1)$ represents the time required to generate the combination, while $k(2^n - 1)$ is the time required to obtain the corresponding equations for each combination. The value k refers to the time needed for the algorithm in [13] to obtain an equation from the total number of input points. The maximum value possible for k is given by the total number of points in the outline of the conglomerate and corresponds to the combination that includes all the arcs.

The evident disadvantage of this process is the large number of combinations requiring analysis. Upon considering the different options for streamlining the process, we devised the

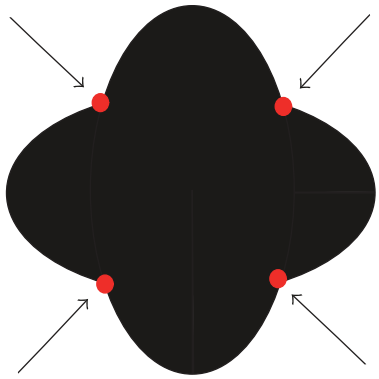


Figure 3: Conglomerate of two elliptical objects. Trigonometric properties generated four SCPs.

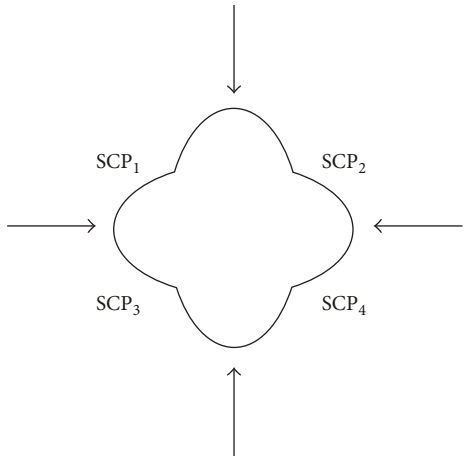


Figure 4: Arcs in a conglomerate of two elliptical objects. The number of arcs is equal to the number of SCPs.

following restriction using information from the base ellipse to restrict the amount of required combination processing (this relates to whether or not the ith combination should have an ellipse adjusted to it):

> *If the total points in the ith combination are fewer than or the same as the total points in the base outline of ellipse, said combination will be analyzed, and if not, the combination will be discarded.*

With this restriction added, it is possible to reduce the total number of combinations requiring analysis and determine whether they represent ellipses to be counted or not, while at the same time reducing the total runtime without the use of parallel computing. It is important to mention that this restriction does not affect the result of the algorithm, because by discarding combinations with numbers of points greater than those on the base outline of ellipse, all potentially incongruous ellipses are essentially being discarded.

The order of time complexity of an algorithm allows comparison of efficiency and performance with other existing methods from a theoretical point of view [14]. Therefore, the total runtime of the algorithm with the base ellipse restriction can be described with the expression $(mn^4) + (2^n - 1)$, in which m represents the total number of points in the ith combination whose maximum value (the worst case) is the perimeter of the base ellipse (the added restriction limits the size of each combination to the total number of points in the base outline of ellipse), which is much lower than k ($m \ll k$). The value n^4 represents the new number of combinations that must be analyzed with the ellipse adjustment method (n^4 was obtained from a polynomial that describes the total increase in combinations complying with the base ellipse restriction, through the generalized method of adjustment of minimum squares for polynomials). The expression $(2^n - 1)$ represents the part of the algorithm responsible for generating the combinations.

The reduction in processing occurs in the section of the algorithm responsible for analyzing and making the ellipse adjustments for the selected combinations; however, it is still necessary to generate the $2^n - 1$ combinations to verify which ones comply with the base ellipse restriction. This, despite being lower than $k(2^n - 1)$, now becomes the most time-consuming step of the algorithm, and accordingly the order of time complexity of the algorithm can be defined as $O(2^n)$.

Notice that theoretically there seems to be no significant time reduction gained by adding the restriction because the order of complexity is still defined as an exponential function; however, once implemented this restriction does provide significant gains in terms of the total run time.

(5) First Selection Criterion. This first step takes the group of points in the outline and then using the method proposed in [13] determines the equation for the ellipse best adjusted to the base ellipse.

For the ellipses obtained in the previous step, their radii will be compared with the radii in the base ellipse equation and if the difference is less than a predefined threshold of T, the equation is preselected for counting. All other equations are discarded.

(6) Second Selection Criterion. The second selection criterion aims to filter out the equations selected in the previous step which represent the same object in the image. When an equation is selected, we need to mark the arcs forming the combination from which the equation comes as used. The equation is discarded if the next equation is preselected and the original combination is formed by an already used arc.

(7) Number of Semicircular or Elliptical Objects. In this step, only those equations that survived the previous step are counted. They are superimposed onto the original input image of the conglomerate and the result is shown to the user.

Table 1: Types of memory available in a GPU with CUDA and the speeds they support. Data obtained from reference in [11].

	Registers	Shared memory	Texture memory	Constant memory	Global memory
Bandwidth	~8 TB/s	~1.5 TB/s	~200 MB/s	~200 MB/s	~200 MB/s
Latency	1 cycle	1 to 32 cycles	~400 to 600	~400 to 600	~400 to 600

Figure 4 illustrates the methodology of the algorithm for the counting of objects in the presence of occlusion. The programming codes and the images used can be consulted in the following link https://github.com/joselopmart/CountEllipses.git.

3. Programming GPUs with CUDA

The Graphics Processing Unit (GPU) is a hardware device that has become a very cost-effective way to obtain high capacity computing power. A GPU does not differ greatly from a cluster in terms of its architecture; essentially it is comprised of a number of Streaming Multiprocessors (SMs). Each SM has its own L1 cache memory as well as an L2 cache memory for communication with other SMs. An SM is generally made up of eight or more Streaming Processors (SPs), also known as CUDA cores [15]. Generally, a GPU is made up of multiple SMs, and as each SM can support concurrent executions of thousands of processing threads, this gives the GPU considerable value as a device for intensive parallel computing [16].

CUDA is a platform for the development of parallel algorithms in C language created by the NVIDIA company, to allow users, exploiting the computing capacity of their GPUs and meeting the need for a computing language capable of using their graphics units for general purposes (GPGPU) [15]. CUDA uses a variant of the SIMD (Single Instruction, Multiple Data) parallel systems model based on the use of threads, warps, blocks, grids, and kernels [15, 17].

Generally, in CUDA the CPU is referred to as the host, while the GPU is named as the device [17]. For the device to execute a given task in parallel, first it is divided into subtasks, these are assigned to concurrent processing threads, which are then grouped into blocks. In turn, the blocks are grouped into warps, and finally each warp is assigned to an SM to be executed [17]. Finally, to execute a parallel process in the device, first the process must be initiated from the host via the definition and call of a kernel. We then provide the total number of blocks and threads per block to be used. Depending on the nature of the parallelized task, the number of blocks and threads can be grouped into grids of specific dimensions.

The CUDA platform offers several types of memory to the programmer, which vary in usefulness according to the specific characteristics of the algorithm in question. Table 1 shows the types of memory, bandwidth, and latency supported by CUDA, with [15] used as a reference.

In practice, memory use varies according to the characteristics of the method and it is necessary to take into account certain restrictions; for example, the use of registers is limited to a reduced memory space but it offers the best possible speeds, while global memory offers the greatest space possible but the lowest speeds. Often, using the wrong memory type can cause the parallelized method to generate increased synchronization, copying, and writing times compared to the algorithm methodology itself.

4. The Parallel Algorithm for the Counting of Ellipses

Analysis and adjustment for each combination happen independently of the other combinations; this means it is possible to employ parallelism during their execution. However, owing to the base ellipse restriction, a load imbalance would be generated [18] as some processors would not be carrying out tasks when assigned combinations that did not pass the defined restriction.

The best alternative for parallelization would be to first ascertain the group of combinations that comply with the base ellipse restriction. Once this group is established, processing is carried out using parallelism to obtain the corresponding ellipse and complete the corresponding process. The application of parallelism to the algorithm has been refined into three steps described below.

(1) Definition of Parallelism Level. This step begins after preprocessing (binarization, establishing the SPs, establishing the arcs, etc.). First it is necessary to define a constant positive integer named ***MAX_THREADS***, which will represent the maximum number of processes to be executed in parallel. For the CUDA programming platform, this value is associated with the number of processing threads being used. The value should be a multiple of 2.

(2) Selection of the Combinations That Comply with the Base Ellipse Restriction. The combinations that have complied with the base ellipse restriction are obtained. This process is done by checking whether the total number of points that form the jth combination complies with the base ellipse restriction, where $0 < j < 2^n$ (n being the number of SCPs). This means that the algorithm must generate the $2^n - 1$ combinations to determine which ones comply with the aforementioned restriction. Notice that there is no attempt to change this part of the methodology (generate the $2^n - 1$ combinations) because it represents a fundamental process for the counting algorithm.

Parallelism is applied to this step as follows:

(2.1) The first *MAX_THREADS* combinations are selected from the $2^n - 1$ total.

```
Input: Ieb, Ic
Output: Ieg, TE
Begin
(1)  SCPs = get_scps (Ic)
(2)  ARCS = arc_generation (Ic, SCPs)
(3)  base_eq = base_equation_ellipse (Ieb)
(4)  n = SCPs.length
(5)  MAX_THREADS = The maximum number of processes to be executed in parallel.
(6)  iter = 1
(7)  while iter < 2^n:
(8)      total_combvalids = 0
(9)      while total_combvalids < MAX_THREADS:
(10)       COMBVAL_TMP = kernel_comb_valids <<<MAX_THREADS processes>>> (iter, ARCS, base_eq)
(11)       total_combvalids + = COMBVAL_TMP.length
(12)       COMBVAL.add (COMBVAL_TMP)
(13)       iter + = MAX_THREADS
(14)     end
(15)     EQ_ELLIPSES_1CS_TMP = kernel_adjust_ellipses <<<MAX_THREADS processes>>> (COMBVAL, base_eq)
(16)     EQ_ELLIPSES_1CS.add (EQELLIPSES_1CS_TMP)
(17) end
(18) EQ_ELLIPSES_SEL = verify_2cs (EQ_ELIPSES_1CS)
(19) TE = EQ_ELLIPSES_SEL.length
(20) Ieg = graph_equations (Ic, EQ_ELLIPSES_SEL)
End
```

Pseudocode 1: Parallel algorithm for the counting of elliptical objects in occlusion.

(2.2) In parallel, each of the *MAX_THREADS* combinations is analyzed by a processor to check whether it complies with the base ellipse restriction. Those that comply are marked as valid.

(2.3) Combinations marked as valid are saved contiguously to obtain their corresponding ellipse equation. Combinations marked as nonvalid are discarded.

(2.4) Once the previous parallel process ends, this procedure is followed: if the total number of saved combinations from step (2.3) does not surpass the *MAX_THREADS* combinations, the following group of *MAX_THREADS* combinations from the $2^n - 1$ total combinations is analyzed (repeated from step (2.1)).

(2.5) Once the maximum number of *MAX_THREADS* combinations complying with the restriction is obtained, or there are no more combinations to be evaluated with the restriction, the next step will be step (3). If combinations from the $2^n - 1$ total remain unverified according to the restriction, then once step (3) is finalized, the missing combinations are reexamined and the process is repeated.

Ideally, for this step to be executed only once, *MAX_THREADS* should be higher than the value for the total combinations complying with the base restriction.

(3) Obtaining the Equations for the Ellipses. Once the previous step is finalized, we have a group of combinations complying with the base ellipse restriction whose maximum size is *MAX_THREADS*.

In the next step we obtain the equation for the ellipse best adjusted to each combination of the generated group, an action which, in the original sequential method, represented the costliest stage in terms of resources. In this step each of the obtained combinations is analyzed in parallel as described below.

(3.1) For each combination, using the method from [13], the equation for the best adjusted ellipse is obtained comprising the following five values ($Cx_i, Cy_i, Rx_i, Ry_i,$ and θ_i), where i represents the ith combination of the combination group obtained in step (2), with $0 < i < MAX\_THREADS$.

(3.2) The first selection criterion (as described in Section 2) is applied to each of the obtained equations. Those equations complying with this criterion are marked as valid and preselected to be potentially used in the postprocessing steps.

Upon finishing the parallel processing, those ellipse equations marked as valid are recovered and saved to continue with the subsequent steps in the ellipse-counting method. If combinations still require processing from step (2), then the process described in step (2.5) will be performed.

Once the parallel processing described in steps (1), (2), and (3) is complete, the original ellipse-counting method is continued, beginning with the execution of step (6) from the original method (described in Section 2) in which the second selection criterion is applied to the preselected equations. Finally, step (7) of the global method is performed to obtain the final result (as described in Section 2).

Shown in Pseudocode 1 are the general steps for the parallel algorithm. There are two input images **Ieb** and **Ic**

corresponding to the base ellipse image and the conglomerate image, respectively. First, the singular concavity points are obtained in the **SCPs** vector, for which image **Ic** (line (1)) is required. Then, in the **ARCS** vector, the arcs generated from conglomerate **Ic** are stored, likewise the singular concavity points in **SCPs**.

In line (3), **base_eq** represents the base ellipse equation that is obtained from the input base ellipse image **Ieb**. The value **n** represents the total SCPs obtained (line (4)). **MAX_THREADS** represents the maximum total number of processes that is possible to execute in parallel.

The **iter** variable allows control over the number of combinations that have been processed from the $2^n - 1$ total (lines (6) and (7)). From line (8) to line (14) we define the procedure required to obtain the first **MAX_THREADS** combinations complying with the base ellipse restriction. For this, **total_combvalids** allows control to be kept when obtaining a maximum number of valid **MAX_THREADS** combinations (lines (8) and (9)).

The **kernel_comb_valids** function is executed in parallel with **MAX_THREADS** processes which are specified between the symbols "⋘" and "⋙" in order to verify whether the combinations $iter, iter + 1, iter + 2, \ldots, iter + MAX\_THREADS - 1$ comply with the base ellipse restriction. To generate the combinations and verify this restriction it is necessary to provide the following data parameters: the number of the first combinations to be verified by **iter**, the **ARCS** to generate the corresponding combinations, and the base ellipse equation **base_eq** to verify the base ellipse restriction (line (10)). The combinations that complied with the restriction are saved in **COMBVAL_TMP** which can vary in size for each iteration of the cycle in line (9). In line (11), the total number of combinations is increased.

For the combinations marked as valid not to get lost in the next iteration of line (9), they are saved in **COMBVAL** (line (12)). In line (13), the total number of analyzed combinations is increased so that line (7) cycle can be completed.

In line (15) the **kernel_adjust_ellipses** function is called using **MAX_THREADS** parallel processes and with the valid combinations stored in **COMBVAL**, along with the base ellipse equation **base_eq**, all the corresponding base ellipse equations are adjusted and the first selection criterion is checked for each equation. The returned result is a group of ellipse equations meeting the first selection criterion, and this is saved as **EQ_ELLIPSES_1CS_TMP**.

In case of a possible second iteration of the algorithm in line (7), the obtained equations are stored in **EQ_ELLIPSES_1CS** (line (16)).

Generally for line (7) cycle to be repeated only once, **MAX_THREADS** should be sufficiently larger than the total number of combinations that comply with the base ellipse equation.

In line (18), the equations stored in **EQ_ELLIPSES_1CS** are verified for compliance with the second selection criterion, and those that meet this criterion are stored in **EQ_ELLIPSES_SEL**, which will represent the final equations to be counted and illustrated correspondingly in the **TE** variable and **Ieg** output image.

Order of Complexity of the Parallel Method. When applying parallelism, $T(n)$ defines the total runtime of the proposed algorithm.

$$T(n) = m + \frac{2^n - 1}{MAX\_THREADS}. \tag{1}$$

The first term given as m represents the new total runtime needed to analyze and adjust ellipses to each of the selected combinations (remember that the maximum value for m is the perimeter of the base ellipse). This time reduction represents the main instance of parallelism applied to the algorithm.

The second term represents the time needed to obtain combinations that comply with the base ellipse restriction. From $T(n)$ we can ascertain the most resource-hungry part of the parallel method and thus determine the order of complexity.

$$O\left(\frac{2^n - 1}{MAX\_THREADS}\right). \tag{2}$$

5. Experimental Results

We implemented the algorithm using the parallel programming platform CUDA-C, and, for both the preprocessing [19] and the generation of the output image with illustrated results, we used OpenCV 2.4 library.

For the experiments, we used a computer with the following characteristics:

(i) Intel Xeon processor with 12 cores, each with 1.6 GHz velocity

(ii) 32 GB of RAM

(iii) NVIDIA GPU QUADRO K6000 with 12 GB of memory and 2880 CUDA CORES.

For the parallel algorithm a value of $2^{18} = 262{,}144$ was used for the *MAX_THREADS* constant.

For the experiments, five groups of images were used, the first four were composed of images designed artificially to measure the performance of the algorithm. The fifth group was composed of real images to check the functionality of the algorithm in real situations.

5.1. First Group of Experiments to Measure the Increase in Performance. The objective of this first experiment was to analyze the increase in performance of the parallel algorithm in comparison with the best sequential version. We created 15 artificial conglomerate images with dimensions of 2000 × 2000 pixels, each one with $3, 4, 5, 6, \ldots, 17$ ellipses in occlusion, respectively. The total of SCPs in each conglomerate was increased in increments of two, starting with three SCPs in the first conglomerate. The base ellipse used for generating the conglomerates had radii of 105 px × 140 px. The conglomerates used are illustrated in Figure 5 (1–15).

Table 2: Comparison of the runtimes of the parallel algorithm with the best sequential algorithm for the counting of elliptical objects in the presence of conglomerates using images from the first group to measure the increase in performance of the parallel method.

Conglomerate	Parallel algorithm time (seconds)	Sequential algorithm time (seconds)	Ellipses detected	PSs detected
1	0.445	0.013	3	3
2	0.443	0.01	4	5
3	0.447	0.011	5	7
4	0.443	0.014	6	9
5	0.453	0.016	7	11
6	0.453	0.023	8	13
7	0.453	0.046	9	15
8	0.437	0.099	10	17
9	0.469	0.218	11	19
10	0.453	0.582	12	21
11	0.485	2.1	13	23
12	0.594	7.938	14	25
13	1.047	31.807	15	27
14	3.02	130.77	16	29
15	11.266	527.306	17	31

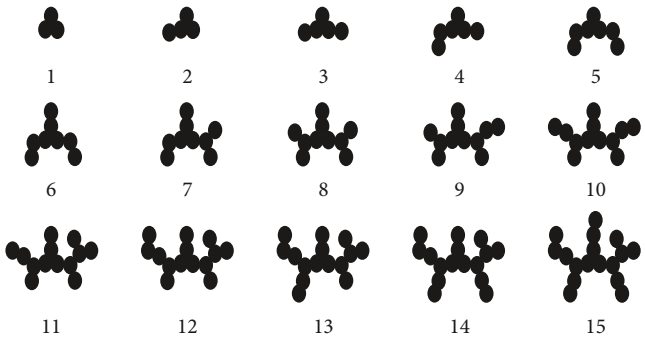


Figure 5: Images belonging to the first group of experiments.

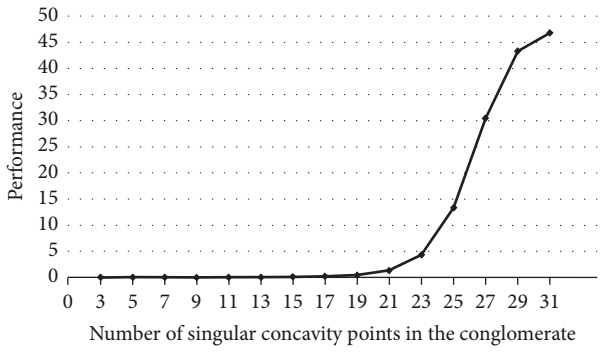


Figure 6: *Speedup* of the parallel algorithm compared with the sequential algorithm.

The experiments consisted of applying the parallel method and the sequential method to each conglomerate, comparing the obtained results and calculating the increase in *speedup*.

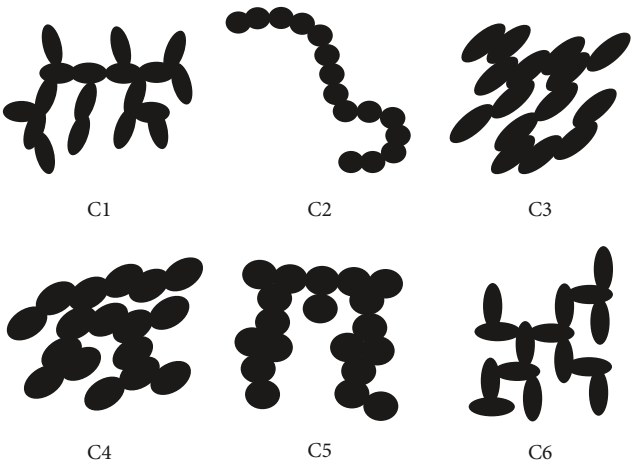


Figure 7: Second group of conglomerates used for the experiments. (C1) Conglomerate with 18 ellipses, with 100 px × 50 px radii. (C2) Conglomerate with 15 circles with 75 px radius. (C3) Conglomerate with 15 ellipses with 122 px × 50 px radii. (C4) Conglomerate with 17 ellipses with 17 px × 115 px radii. (C5) Conglomerate with 19 circles, with 150 px radius. (C6) Conglomerate with 15 ellipses with 185 px × 75 px radii.

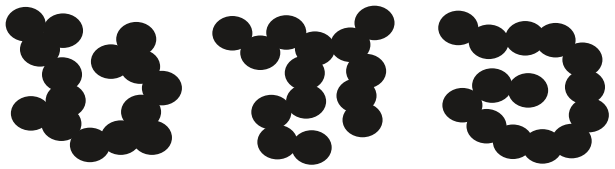
The *speedup* refers to how much faster an algorithm performs using p processors, compared with the same algorithm in its best sequential version [20]. This value can be obtained with the following formula:

$$S_p = \frac{T(n)}{T_p(n)}, \tag{3}$$

where $T(n)$ is the time taken for the best sequential algorithm to resolve a problem of size n and $T_p(n)$ is the time taken for the parallel algorithm with p processors to resolve the same

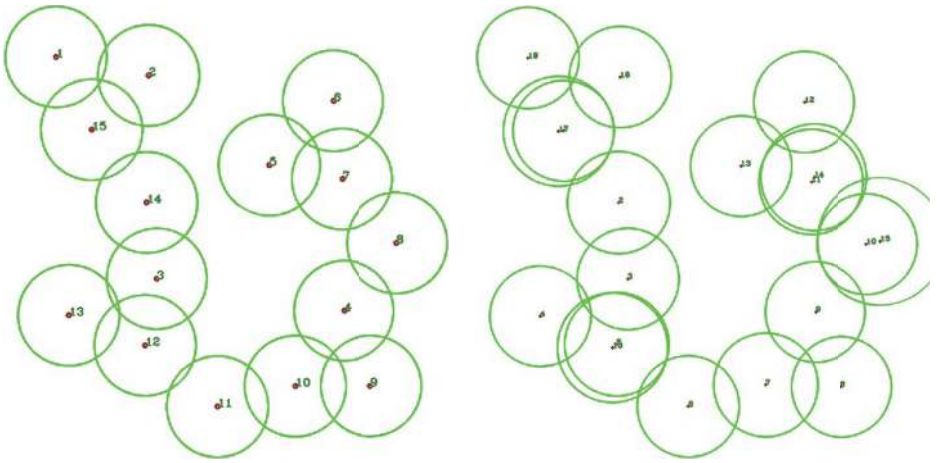
Table 3: Comparison of runtimes between parallel and sequential methods to estimate the maximum performance that the parallel algorithm could achieve.

Conglomerate	Parallel algorithm time (seconds)	Sequential algorithm time (seconds)	Performance
C1	5.329	258.597	48.5263652
C2	1.5	60.545	40.3633333
C3	22.126	1089.46	46.7920801
C4	10.721	524.592	46.0451154
C5	4.962	255.459	46.8604185
C6	1.14	60.831	39.3219134



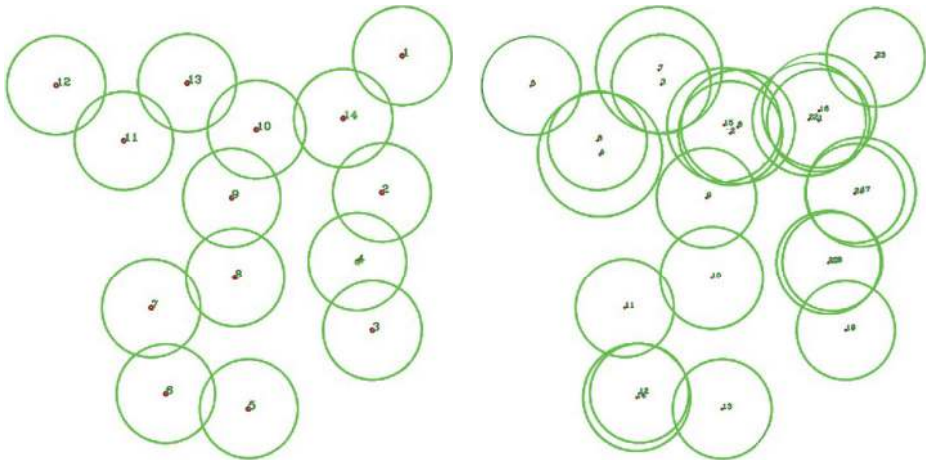
CA CB CC

Figure 8: Conglomerates used to carry out comparative experiments between the proposed parallel method and the method from Bera [4]. The base circular object has a radius of 140 px. All the images had dimensions of 1400 px × 1400 px. (CA) Conglomerate with 15 circles. (CB) Conglomerate with 14 objects. (CC) Conglomerate with 14 objects.



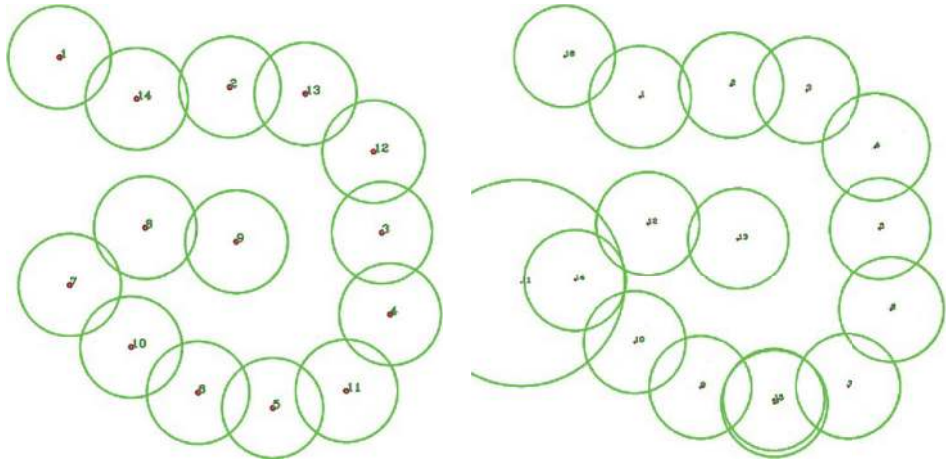
(a) (b)

Figure 9: Results for conglomerate CA. (a) Result of parallel algorithm on conglomerate CA. (b) Result of algorithm in [4] on conglomerate CA.



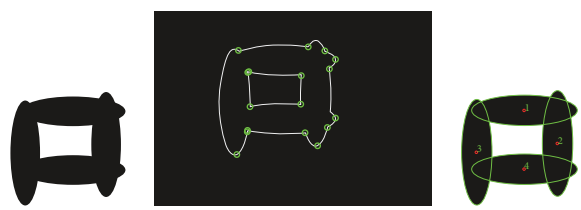
(a) (b)

Figure 10: Results for conglomerate CB. (a) Result of the parallel algorithm for conglomerate CB. (b) Result of the algorithm in [4] for conglomerate CB.



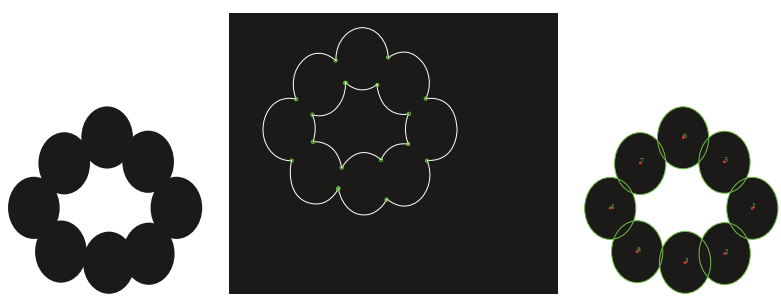
(a) (b)

Figure 11: Results for conglomerate CC. (a) Result of the parallel algorithm for conglomerate CC. (b) Result of the algorithm in [4] for conglomerate CC.



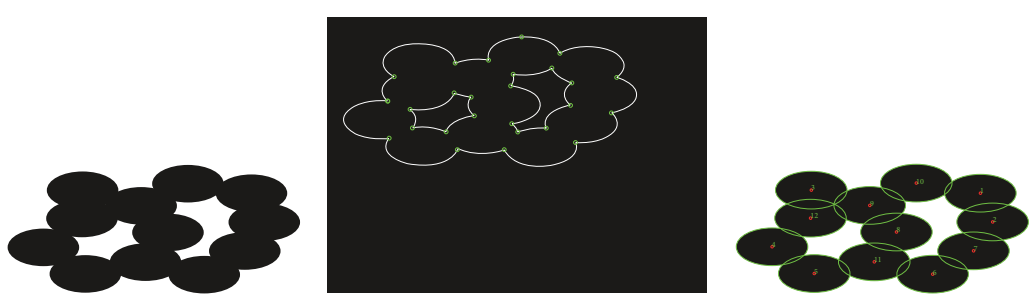
(a) (b) (c)

Figure 12: Result of the parallel algorithm on R1 with a hole. (a) Experiment R1 with 4 objects. (b) Detection of 17 SCPs. (c) Result of our parallel method: 4 objects detected in 0.597 seconds.



(a) (b)

Figure 13: Result of the parallel algorithm on experiment R2. (a) Image of experiment R2. (b) Detection of 18 SCPs. (c) Result of the detection of eight objects with our parallel algorithm.



(a) (b) (c)

Figure 14: Result of the parallel algorithm on experiment R3. (a) Image of experiment R3. (b) Detection of 27 SCPs. (c) Result of the detection of 12 objects with our parallel algorithm.

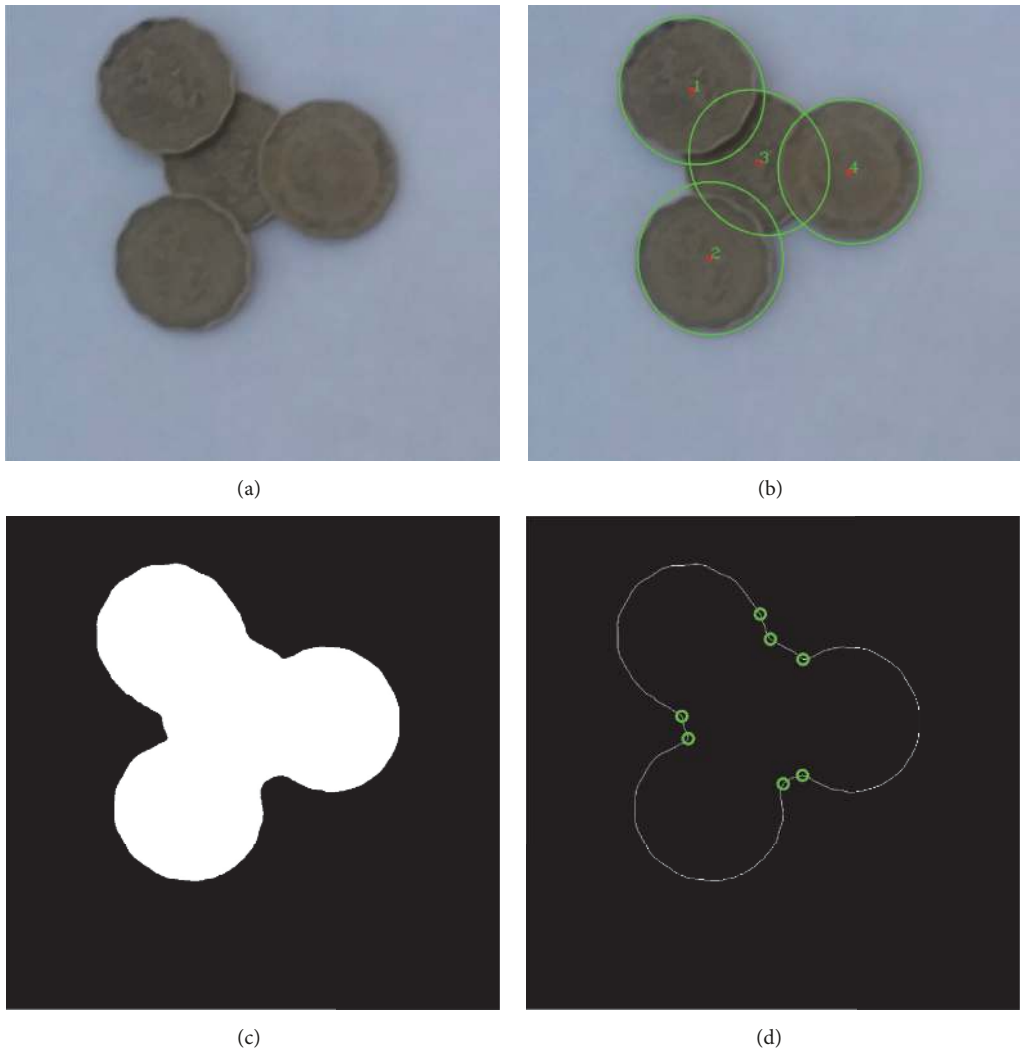


Figure 15: Result of the parallel algorithm on the first real image. (a) Image of the conglomerate with 4 objects. With dimensions of 929 px × 929 px. (b) Result of parallel algorithm, 4 objects detected. (c) Binarization of the conglomerate. (d) Detection of 7 SCPs.

size n problem [21]. Table 2 shows the results obtained from the experiments with the first group of images.

Notice that, for the first experiments from Table 2, the times for the parallel method were approximately 0.4 seconds. This is due to processes of initialization and data transfer to the GPU's memory and cannot be controlled in the implementation.

Using the formula to ascertain *speedup*, we plotted the graph in Figure 6 to represent the enhancement in algorithm performance as the number of SCPs in the conglomerate increased, showing performance up to 40 times better. This means that the parallel method was 40 times faster than the sequential method.

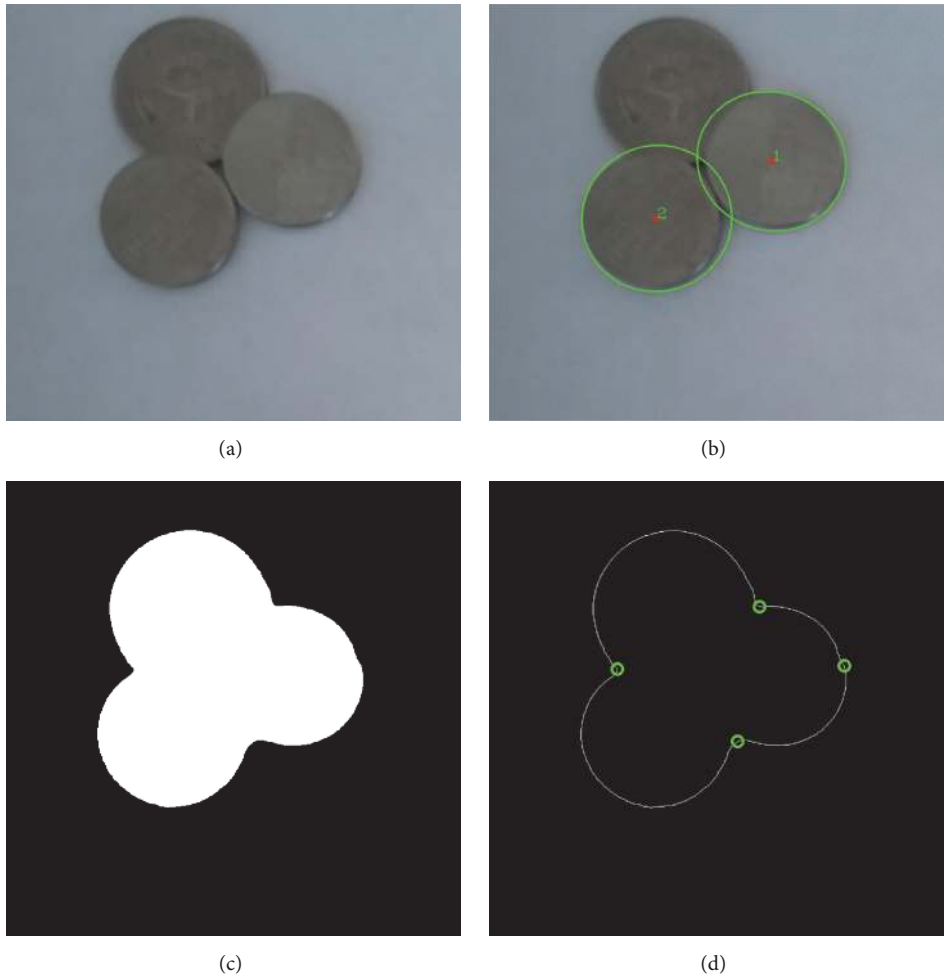
5.2. Second Group of Experiments to Obtain an Estimate of the Maximum Performance of the Parallel Algorithm. With the objective estimating the maximum performance that the parallel algorithm is capable of, six experiments were performed on artificial images created specifically to require considerable processing. Figure 7 shows the six conglomerates used in this experiment where each one has been labelled for simplicity (C1, C2, C3, and C4).

The execution of the experiments consisted of running the parallel method and the sequential method on each conglomerate from Figure 7 and then calculating the performance achieved in each case.

Table 3 shows the times obtained for these experiments as well as the individual performances achieved.

The shortest runtime was that of conglomerate C2, while the longest was that of conglomerate C3. It is worth noting that in general the times varied greatly between the six conglomerates, a factor which should be taken into account when analyzing performance.

With this data it is possible to estimate the maximum performance that the parallel method can achieve. A good value would be the average of the performances achieved with



(a) (b) (c) (d)

Figure 16: Result of the parallel algorithm on the second real image. (a) Image of the conglomerate with 3 objects. With dimensions of 833 px × 833 px. (b) Result of parallel algorithm, 2 objects detected. (c) Binarization of the conglomerate. (d) Detection of 4 SCPs.

images requiring high computing intensity. It follows that a reliable estimate of *speedup* would be 44.6565, meaning that the parallel method could be up to 44 times faster than the sequential method.

5.3. Third Group of Experiments to Compare the Parallel Method with the Method Present in Literature for the Counting of Elliptical Objects. In 2015, Bera [4] presented a method for detecting circular objects in occlusion. In his method Bera used a procedure based on finding the centroid of the conglomerate and from there finding the local minimum values in the outline of the conglomerate in order to plot the arcs. Later, using geometric properties for each of the arcs, both the radius and center of the circle passing through the arc can be ascertained.

The following experiments make comparisons with the method in [4] with regard to the times and results obtained. These experiments consisted of running the parallel method and the method in [4] on the three images of conglomerates shown in Figure 8, which have been labelled for simplicity (CA, CB, and CC).

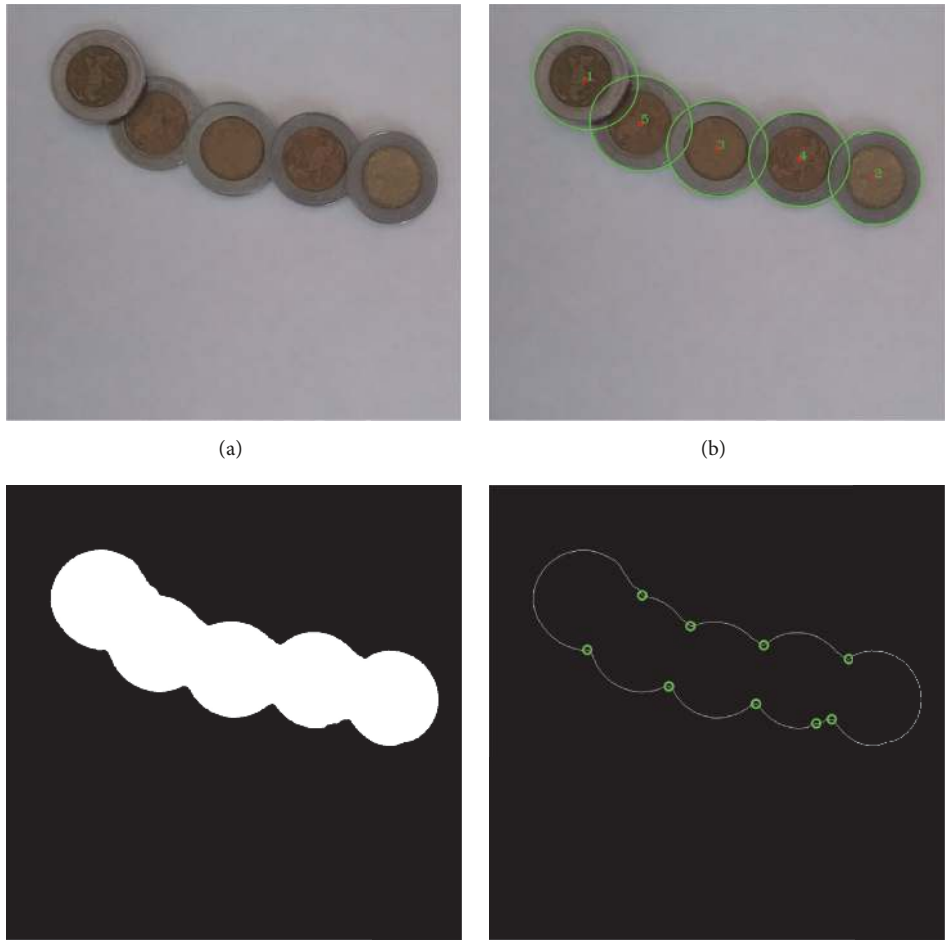
The following is a description of the results obtained for each experiment.

In Figure 9 the result obtained for conglomerate CA is shown. The proposed parallel method detected 15 ellipses in 0.515 seconds (Figure 9(a)), while the method proposed in [4] detected 19 ellipses in 0.001 seconds (Figure 9(b)). The correct result for this conglomerate should be 15 ellipses detected.

Figure 10 shows the result obtained for conglomerate CB. The parallel method detected 14 objects in 0.766 seconds (Figure 10(a)), while the method in [4] detected 23 objects in 0.001 seconds (Figure 10(b)). The correct result should be 14 objects detected.

In the case of conglomerate CC, Figure 11 shows the results obtained, where the parallel method detected 14 objects in 0.719 seconds (Figure 11(a)), while the method in [4] detected 16 objects in 0.004 seconds (Figure 11(b)).

Notably these results confirm that the parallel method is a good option for dealing with the problem being studied. Despite the method in [4] obtaining better times, in both cases times were below one second, which represents a



(a) (b) (c) (d)

Figure 17: Result of the parallel algorithm on the third real image. (a) Image of the conglomerate with 5 objects. With dimensions of 1097 px × 1097 px. (b) Result of parallel algorithm, 5 objects detected. (c) Binarization of the conglomerate. (d) Detection of 9 SCPs.

very short waiting time for the user. The parallel method has the benefit of very good processing times coupled with very good results in the detection of objects in occlusion.

5.4. Fourth Group of Experiments with Images Generating Holes When Occluding. The aim of the following experiments is to analyze the performance of our algorithm when occluded elliptical objects generate holes. We create three artificial images (R1, R2, and R3) with these features. We use the parallel method to reduce the execution time of our algorithm.

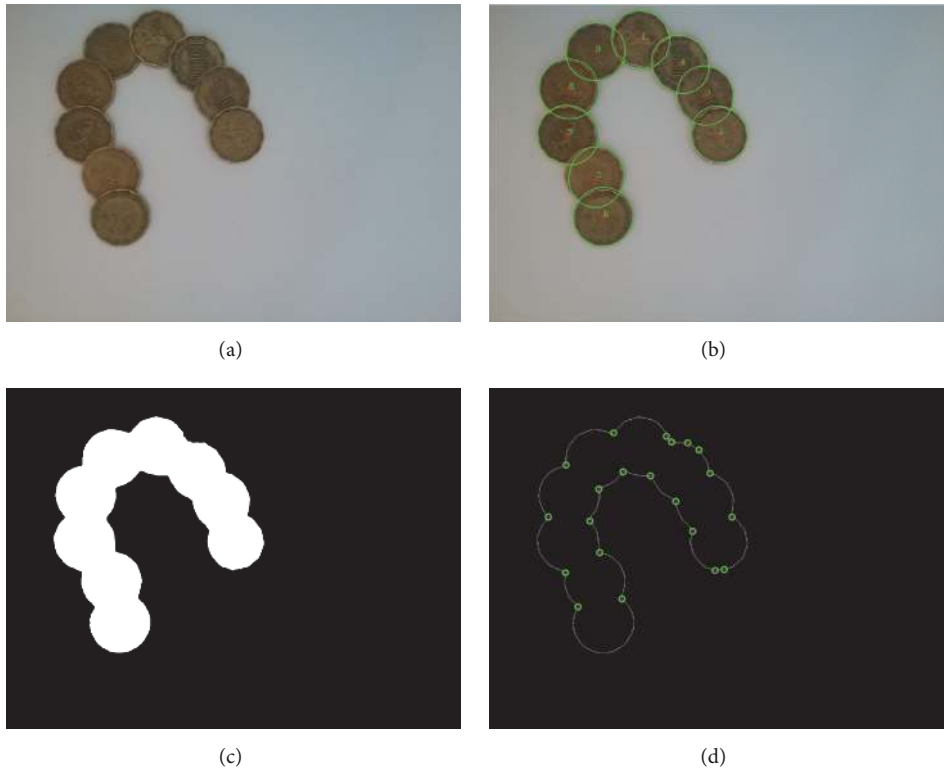
The experiment in R1 (see Figure 12(a)) consists of four elliptical occluded objects forming a hole; the size of each one of these ellipses is 56 px × 195 px radii. Notice that our algorithm detects the 17 SCPs (see Figure 12(b)). Finally, we observe the result of the parallel method in Figure 12(c), which takes 0.597 seconds in detecting the ellipses.

Now, we present the results of experiment R2. This experiment consists of eight ellipses of size 162 px × 194 px radii generating a hole in the center. Our algorithm detects the 18 SCPs and the eight ellipses (see Figure 13). The execution time of our algorithm is 0.578 seconds.

Finally, we present the results of experiment R3 (Figure 14(a)). This experiment presents 12 occluded elliptical objects forming two holes in the conglomerate. The elliptical objects are of size 194 px × 102 px radii. Our algorithm detects 27 SCPs (Figure 14(b)). Figure 14(c) illustrates the 12 detected elliptical objects using the parallel method. The execution time for experiment R3 is 1.394 seconds.

Results of experiments R1, R2, and R3 show that our algorithm is precise when there exist holes generated by occluded objects. Additionally, the execution time of our algorithm is low even in such a case.

5.5. Fifth Group of Experiments Involving Real Images. These experiments were conducted using real images with the aim



(a) (b) (c) (d)

Figure 18: Result of the parallel algorithm on the fourth real image. (a) Image of the conglomerate with 9 objects. With dimensions of 1600 px × 1200 px. (b) Result of parallel algorithm, 9 objects detected. (c) Binarization of the conglomerate. (d) Detection of 21 SCPs.

of checking and measuring the correct functionality of the parallel method in real situations.

To perform the experiments, six conglomerate images were captured using a Samsung Galaxy Tab tablet with a 5 MP camera. The conglomerates were created from coins in positions of occlusion. Local binarization and edge-smoothing processes were added in the preprocessing stage.

Next, each conglomerate with its corresponding results is presented in Figures 15–20. In this case, for each result the obtained binarized image is included together with the SCP detection along the outline of the corresponding conglomerate.

Figure 15 shows the image of the first conglomerate with four coins as detected in the results obtained with the parallel method (Figure 15(b)) in a time of 0.484 seconds. The sequential method obtained the same result (Figure 15(b)) in 0.014 seconds.

Figure 16(a) shows the image of the second conglomerate with 3 coins in occlusion, in this case, one coin is larger in size and, due to the fact that the base ellipse in the experiment was entered as similar in size to the smaller coins in the image, the results (Figure 16(b)) revealed that only the two smaller (similarly sized) coins had been detected. This result was obtained with the parallel method in a time of 0.473 seconds. The sequential method obtained the same result in 0.014 seconds.

Figure 17 shows the result of the parallel algorithm when applied to the third image of a real conglomerate (Figures 17(a) and 17(b)). This result was obtained in 0.461 seconds. It is important to note that for the method to be successful a satisfactory binarization is necessary in order to obtain only the relevant information. As can be seen in Figure 17(a), there are shadows around the conglomerate that could cause problems when processing the data. For the sequential method the same results were obtained in 0.017 seconds.

In Figure 18(a) we can see the fourth real image of a conglomerate, this time showing 9 coins. The figure also shows the result from the parallel method (Figure 18(b)) which successfully detected all 9 objects as expected in a time of 0.498 seconds. The sequential method obtained the same result (Figure 18(b)) in a time of 1.1641 seconds. For this experiment, the parallel performance was 3.2958 times faster.

The fifth real image is shown in Figure 19(a) and it is comprised of 8 coins in occlusion. The result from the algorithm (Figure 19(b)) was available in a total time of 0.563. It is important to highlight that the method detected 9 objects, although 8 was the correct number, and this can be attributed to two factors; the first occurs because of possible unnecessary SCP detection in the binary image (Figures 19(c) and 19(d)), and the second is due to the fact that the

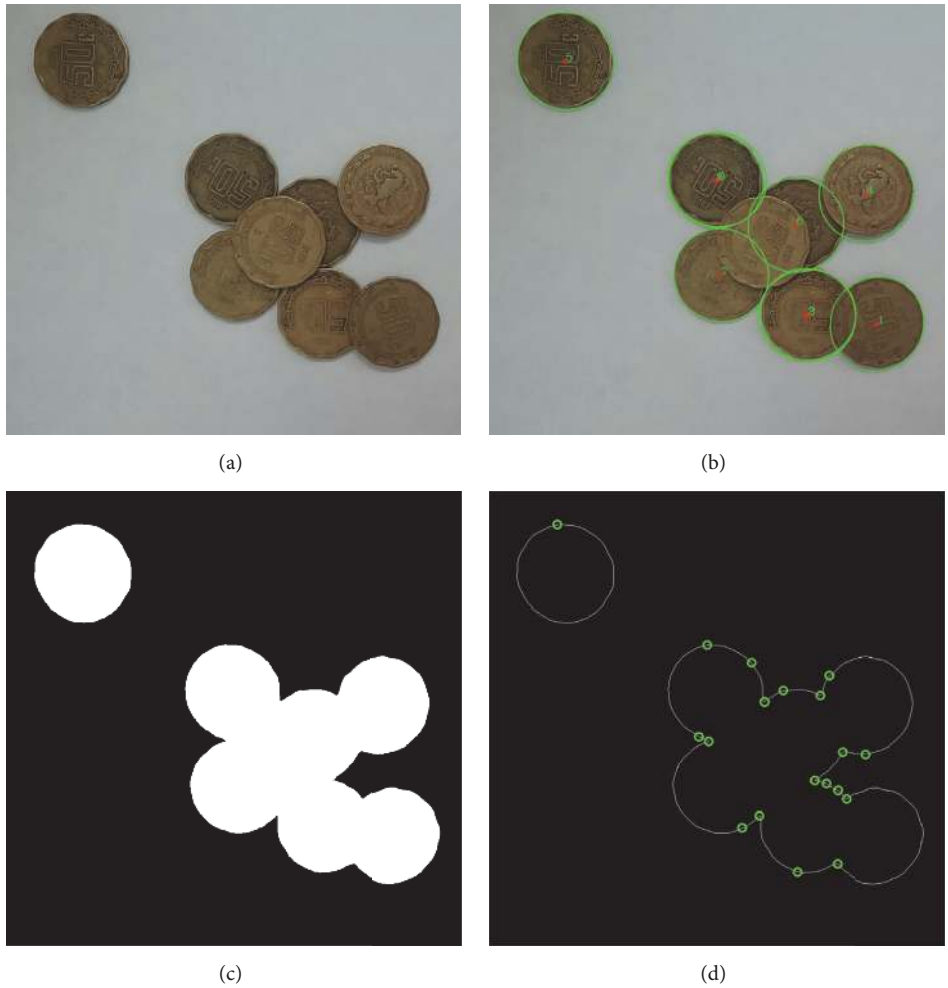


FIGURE 19: Result of the parallel algorithm on the fifth real image. (a) Image of the conglomerate with 8 objects. With dimensions of 1162 px × 1162 px. (b) Result of parallel algorithm, 9 objects detected. (c) Binarization of the conglomerate. (d) Detection of 19 SCPs.

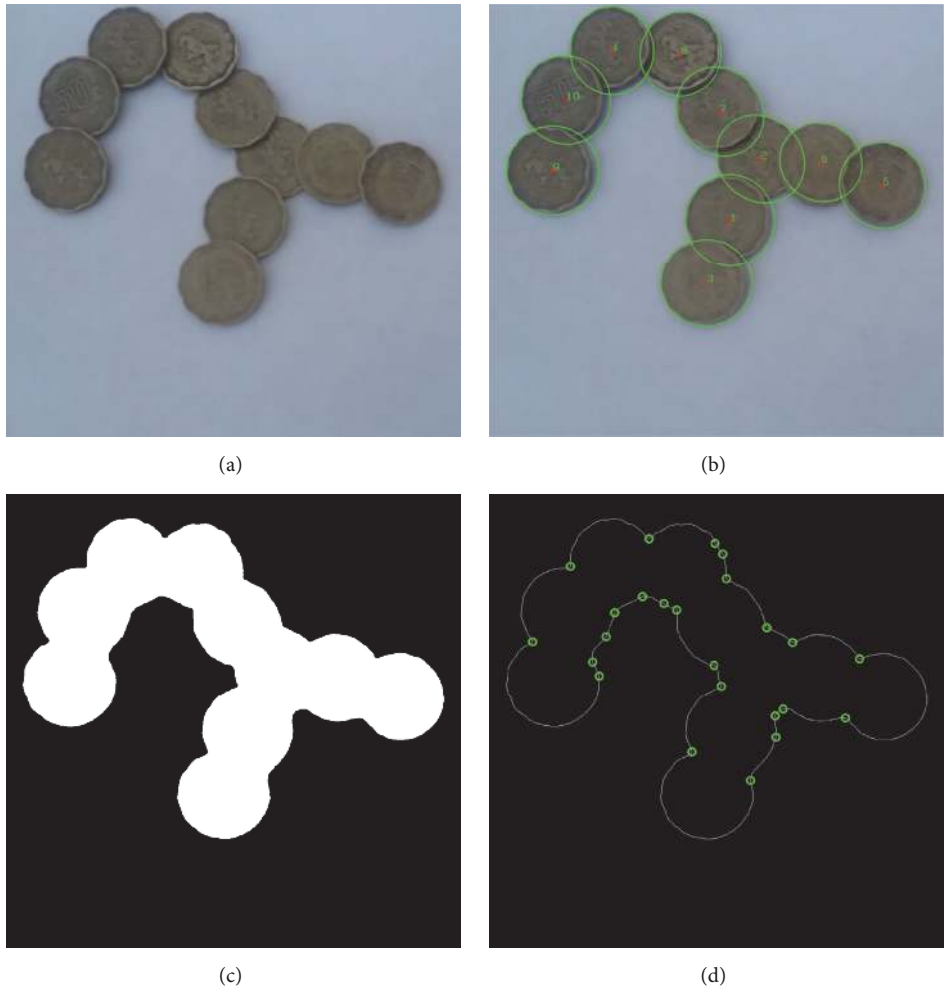
parallelized ellipse-counting method only works correctly on binary images and therefore when two or more objects in the image are completely occluded, applying binarization will mean that the data for the underlying object(s) is lost. The sequential method obtained the same result (Figure 19(b)) in 1.636 seconds. For this experiment, the performance was 2.9058 times better.

The sixth and final real image was a conglomerate (Figure 20(a)) formed by 10 coins in occlusion. Running the parallel method on this image gave results detecting all 10 of the requested objects (Figure 20(b)) in a time of 0.975 seconds. The sequential method obtained the same result in 12.835 seconds, which gives a performance 13.16 times faster for the parallel method over the sequential method.

6. Conclusions

In this study, we presented the parallelization of an algorithm for the counting of elliptical objects within conglomerates using a GPU and the CUDA-C platform. Several experiments were conducted to ascertain the performance of the parallel method compared with the best sequential version, at times achieving performances up to 40 times faster. Comparison of results was also made with an existing study in literature, in which our experiments showed the strong performance and results obtained by the parallel algorithm. Finally, experiments were performed using real images of conglomerates created with various occluded coin configurations.

One weakness that remains with the parallel method and which exists outside the purview of this study is the difficulty in dealing with conglomerates with a high percentage of occlusion, resulting in a very high number of SCPs (up to 100) being detected. Initially the problem seems to be caused by the implicit difficulty of dealing with processes that require iterations of over $2^{100} - 1$ combinations, and for this reason future studies will aim to identify possible ways to omit this action or at least to reduce the processing load. As future work, we would implement evolutionary strategies to obtain good approximations to the ellipse adjustment. Another option could be to divide the image into subimages



(a) (b) (c) (d)

Figure 20: Result of the parallel algorithm on the sixth real image. (a) Image of the conglomerate with 10 objects. With dimensions of 1255 px × 1255 px. (b) Result of parallel algorithm, 10 objects detected. (c) Binarization of the conglomerate. (d) Detection of 24 SCPs.

and thus, assuming a relatively even distribution of SCPs, reduce the total number of combination iterations required.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to express their gratitude to UADY-PADECCA 2015 for supporting this project.

References

[1] M.-J. Su, Z.-B. Wang, H.-J. Zhang, and Y.-D. Ma, "A new method for blood cell image segmentation and counting based on PCNN and autowave," in *Proceedings of the 2008 3rd International Symposium on Communications, Control, and Signal Processing*, pp. 6–9, St Julians, Malta, March 2008.

[2] H. Su, F. Xing, J. D. Lee, C. A. Peterson, and L. Yang, "Automatic myonuclear detection in isolatedsingle muscle fibers using robust ellipse fitting and sparse representation," *IEEE Transactions on Computational Biology and Bioinformatics*, vol. 11, no. 4, pp. 714–726, 2014.

[3] J. Sossa Azuela, G. Guzman Lugo, and R. Sotelo Rangel, "Counting the number of blobs in an image," in *Proceedings of the 2001 International Conference on Image Processing*, vol. 1, Cat. No. 01CH37205, pp. 1086–1089, Thessaloniki, Greece.

[4] S. Bera, "Partially occluded object detection and counting," in *Proceedings of the 2015 3rd International Conference on Computer, Communication, Control and Information Technology (C3IT '15)*, pp. 1–6, Hooghly, India, February 2015.

[5] A. Verikas, A. Gelzinis, M. Bacauskiene, I. Olenina, S. Olenin, and E. Vaiciukynas, "Phase congruency-based detection of circular objects applied to analysis of phytoplankton images," *Pattern Recognition*, vol. 45, no. 4, pp. 1659–1670, 2012.

[6] H. Sossa, G. Guzmán, O. Pogrebnyak, and F. Cuevas, "Object counting without conglomerate separation," in *Proceedings of the 4th Mexican International Conference on Computer Science (ENC '03)*, pp. 216–220, September 2003.

[7] G. Wang, G. Ren, Z. Wu, Y. Zhao, and L. Jiang, "A fast and robust ellipse-detection method based on sorted merging," *The Scientific World Journal*, vol. 2014, Article ID 481312, 15 pages, 2014.

[8] R. Gonzalez and R. Woods, *Digital Image Processing*, Dorling Kindersley, New Delhi, India, 2014.

[9] R.-M. Chao, H.-C. Wu, and Z.-C. Chen, "Image segmentation by automatic histogram thresholding," in *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (ICIS '09)*, pp. 136–141, ACM, Seoul, Republic of Korea, November 2009.

[10] A. Bovik, *Handbook of Image and Video Processing*, Academic Press, 2nd edition, 2010.

[11] L. Grady and O. Lezoray, *Image Processing and Analysis with Graphs*, CRC Press, 2017.

[12] M. Sarfraz, A. Masood, and M. R. Asim, "A new approach to corner detection," in *Computer Vision and Graphics: International Conference, ICCVG 2004, Warsaw, Poland, September 2004, Proceedings*, K. Wojciechowski, B. Smolka, H. Palus, R. S. Kozera, W. Skarbek, and L. Noakes, Eds., pp. 528–533, Springer, Dordrecht, The Netherlands, 2006.

[13] A. W. Fitzgibbon, M. Pilu, and R. B. Fisher, "Direct least squares fitting of ellipses," in *Proceedings of the 13th International Conference on Pattern Recognition (ICPR '96)*, vol. 1, pp. 253–257, Vienna, Austria, August 1996.

[14] T. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, The MIT Press, Cambridge, Mass, USA, 2009.

[15] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, Applications of GPU Computing Series, Morgan Kaufmann Publishers, 2013.

[16] J. Cheng, M. Grossman, and T. McKercher, *Professional CUDA C Programming*, John Wiley & Sons, Indianapolis, Ind, USA, 2014.

[17] D. Storti and M. Yurtoglu, *CUDA for Engineers*, Addison-Wesley, New York, NY, USA, 2016.

[18] G. Golub and J. M. Ortega, *Scientific Computing: An Introduction with Parallel Computing*, Academic Press, Boston, Mass, USA, 1997.

[19] G. Garci, *Learning Image Processing with OpenCV: Exploit the Amazing Features of OpenCV to Create Powerful Image Processing Applications through Easy-to-Follow Examples*, Packt Publishing, Birmingham, UK, 2015.

[20] R. Yam-Uicab, J. L. Lopez-Martinez, J. A. Trejo-Sanchez, H. Hidalgo-Silva, and S. Gonzalez-Segura, "A fast Hough Transform algorithm for straight lines detection in an image using GPU parallel computing with CUDA-C," *The Journal of Supercomputing*, vol. 73, no. 11, pp. 4823–4842, 2017.

[21] J. JaJa, *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Company, Reading, Mass, USA, 1992.