Parallel Hough Transform based on Object Dual and Pymp Library

Abdoulaye SERE
ER-SIC, LAMDI
Université Nazi BONI,
RECIF, https://recifaso.org
Burkina Faso

Moïse OUEDRAOGO
ER-SIC, LAMDI
Université Nazi BONI,
RECIF, https://recifaso.org
Burkina Faso

Armand Kodjo ATIAMPO
UREN
Université Virtuelle de Côte d'Ivoire
Côte d'Ivoire

***Abstract*—Geometric shape detection in an image is a classical problem that leads to many applications, in cartography to highlight roads in a noisy image, in medical imaging to localize disease in a region and in agronomy to fight against weeds with pesticides. The Hough Transform method contributes effectively to the recognition of digital objects, as straight lines, circles and arbitrary objects. This paper deals with the theoretical comparisons of object dual based on the definition of Standard Hough Transform. It also focuses on parallelism of Hough Transform. A generic pseudo-code algorithm, using the Openmp library for the parallel computing of object dual is proposed in order to improve the execution time. In simulation, a triangular mesh superimposed on the image is implemented with the pymp library in python, in considering threads as inputs to read the image and to update the accumulator. The parallel computing presents reduction of the execution time accordingly to the rate of lit pixels in each virtual object and the number of threads. In perspectives, it will contribute to strenghen the developement of a toolkit for the Hough Transform method.**

***Keywords*—Hough transform; parallel computing; pattern recognition**

I. Introduction

Digital object recognition in an image is a classical problem that arises many applications in cartography to highlight noisy roads, in medical imaging for disease detection, in agronomy for weed detection and in cyber-security, as Facial Recognition using deep learning through convolutive neural network application. That supposes the definition of a model for digital objects and establishement of its adequate recognition method. In this sense, Bresenham's line in [18], Reveilles in [9], Andres in [10] have established several digital models, particularly for analytical straight lines. Parallel algorithms such as parallelization Bresenham Line and circle in [19] have been introduced to improve the execution time to draw digital object.

The Hough Transform method has been initially introduced by Paul Hough in [3] in 1962 and applied to the detection of straight lines in an image : It transforms a point in an image space to a straight line in a parameter space. It transposes the problem of straight line detection in the image space to an intersection of straight lines in the parameter space. But this method is limited in the detection of vertical straight lines.

Another variant of the Hough transform method named the standard Hough Transform, associates continuous points in an image space to sine curves in a parameter space. It allows to overcome the problems of vertical straight line recognition that occurs in the case of the classical Hough Transform, which transforms a continuous point in the image space to a straight line in the parameter space.

Forward, the Hough Transform method has been extended to the detection of others shapes such as circles, ellipses. Duda and others in [4] have also proposed the recognition of arbitrary shapes in an image.

In 1985, Henri Maître in [1] has also proposed a survey on the Hough Transform method with an unified definition of Hough transform. Several works have studied the application of Hough Transform, for instance to mouth recognition in [11] and to action detection in [12].

Moreover, extensions of Hough transform for straight line recognition have been proposed by scientists. For instance, in 2006 Martine Dexet in [2], [6] introduced a new method based on the initial definition of Hough Transform, computing the Hough transform of continous points in a square. Then, the dual of a square is a set of continuous straight lines in the parameter space.

By analogy to the works of Dexet in [2], SERE and others in [5] extends the standard Hough transform, to define the dual of a square and the dual of a triangle. Others extensions followed these works in [5] with the dual of a rectangle in [14], the dual of an hexagon in [16] and the dual of an octagon in [15]. All these works will lead to build a toolkit for the Hough Transform methods.

The serial execution of Hough transform consists of the serial execution of each real pixel or each virtual cell defined by the mesh generation, layed on an image.

Many works have carried out improvements of Hough transform to reduce the execution time. Acceleration of Hough transform has been studied by Jošth and others in [21] to allows real time line detection.

Parallelization of the Hough Transform method is also an alternative in order to improve the execution time. For instance, SERE and others in [7] have demonstrated the application of Hough Transform with the map-reduce algorithm for straight line recognition. These works have been extended by Mateus Coelho and others in [8] to deal with circle recognition.

Through recent innovations precisely in deep learning on Convolutive Neural Network, scientists have worked to combine the Hough Transform method with Convolutive Neural

Network to improve algorithms working in object detection, as studied by Spratling in [20].

In Traffic Management on roads, various techniques of Hough Transform have been proposed in Detection of Traffic Saturation, Traffic Light and Traffic Sign. For instance, SERE and others in [16] have introduced an application of the Hough Transform method in Traffic Saturation using GPS.

Our motivation is the study of parallel processing for the dual of geometry shapes related to squares, triangles, rectangles, hexagons and octagons with pymp library in python, in considering a set of threads as input parameters to compute different duals and to update the accumulator data in order to reduce the execution time.

Here, the parallel execution is outside the framework map-reduce in [7]. It takes into account previous works in a serial execution on :

- improvements of standard Hough transform in [13],
- straight line recognition in a rectangular grid in [14],
- straight line recognition in a triangualar grid in [22]

.

But it considers the parallel execution of object dual for any grid to shorten the execution time.

This paper is organized as follows : the Section II recalls the definitions of geometric shape duals and analytical straight lines. The Section III focuses on parallelization of object dual. Experimental results in Section IV use the pymp library to implement parallelism in python on real images.

II. Preliminaries

As previously introduced in Section I, Standard Hough Transform (SHT) associates a point in the image space to a sine curve in the parameter space, to achieve straight line recognition, as defined by :

Definition 1: (Standard Hough Transform)

The dual S(x,y) of the point (x,y) in a two dimensional space where (x, y) $\in R^2$, is the Standard Hough Transform defined by the set of points :

$$\{(\theta,r)\in[0,\pi]\times[-\sqrt{l^2+h^2},\sqrt{l^2+h^2}]/\ r=x\cos\theta+y\sin\theta\} \quad (1)$$

Duality is described as establishment of the relation between a digital object in the image space and the set of standard Hough transform of its internal points in the parameter space. For instance, let O be a digital object such as O= $\cup_{i=1}^{n} P_i$ where $P_i \in$ O. P_i could be a discrete point or a set of discrete points. The dual of O is the union of the dual of each element (P_i $\in$ O), accordindly to the standard Hough transform of P_i in definition 1.

That means formally :

$$Dual(O)=\cup_{i=1}^{n} dual(P_i) \quad (2)$$

The relation 2 has been specialized precisely to obtain the definitions for the dual of geometric shapes, such as rectangles, triangles, as defined by SERE and others in [5], [13], [15] as follows :

Definition 2: (Rectangle dual) the dual of a square (or a rectangle) is the area localized between the curves corresponding to the duals of its two internal diagonal segments in the parameter space.

Definition 3: (Triangle dual) the dual of a triangle is the area localized between the curves corresponding to the duals of its two adjacent sides in the parameter space.

An octagon is the union of four internal rectangles. While a hexagon is the union of three internal rectangles. Definition 2 leads to establish hexagon dual and octagon dual as follows :

Definition 4: (Hexagon dual) the dual of an hexagon is the area localized between between the curves corresponding to the duals of its three internal diagonal segments in the parameter space.

Definition 5: (Octagon dual) the dual of an octagon is the area localized between the curves corresponding to the duals of its four internal diagonal segments in the parameter space.

The dual contributes to analytical straight line recognition, particularly those proposed by Reveilles in [9] related to standard analytical straight line and extended by Andres in [10] to have the supercover model.

Definition 6: analytical digital straight line ([9], [10]) with parameters (a,b,μ) and thickness w is defined by the set of integer points (x,y) verifying : $\mu \leq ax+by < \mu + w$, $(a,b,\mu,w)\in Z^4,\ pgcd(a,b)=1$.

Thus, Analytical digital straight line is then :

- thin, if $w < \max(|a|,|b|)$
- naive, if $w = \max(|a|,|b|)$
- thick, if $w > (|a|+|b|)$
- standard, if $w = (|a|+|b|)$

Parallel computing of object dual will be described in Section III and applied to analytical straight line recognition in Section IV.

III. Method Description

This section is focusing on parallel computing of object dual, previously defined in Section II. It also proposes pseudo-algorithms using openmp library to compute object dual in the parallel context.

As an object dual is defined previously by the relation :

$$Dual(O)=\cup_{i=1}^{n} dual(P_i) \quad (3)$$

Where O is extracted from an image and $P_i \in$ O. Then, parallel computing of Dual (O) consists firstly of splitting an object to several components and secondly to apply parallel computing to each elementary components P_i. Threads will assume easily these elementary tasks. Let n and v be respectively the number of components in an initial object O and the number of threads. Each thread will compute normally almost $[\frac{n}{v}]$ elementary components. That means if the number of threads is superior to the number of elementary components, some threads will not be launched.

Moreover, instead of parallel computing for the dual of all the components P_i, a contraint is introduced by the rate of lit components in O : If an object O has the rate of lit components, superior to the rate α, the dual of this object will be computed in the accumulator data.

Let O_l be a subset of O, constituted only of lit components. Suppose that lc is the number of lit components. We have lc=card(O_l) and $\alpha \leq [\frac{lc}{n}]$. Then, the dual(O) will be approximately the dual(O_l).

Moreover, suppose that $\alpha = [\frac{m}{n}]$. We constitute the subsets of O_l that contain exactly m elements of O_l, denoted P(O_l). There are C_{lc}^{m} subsets (as a combinaison) in O_l that verify the constraint $\alpha = [\frac{m}{n}]$. As $O_m \in$ P(O_l). Now, the dual(O) also becomes approximately the dual(O_m).

Finally :

$$Dual(O_m) = \cup_{i=1}^{n} dual(P_i) \tag{4}$$

Where O_m is extracted from O_l (O_l is a subset of O) and parallelization will concern with the $dual(P_i)$, where $P_i \in O_m$.

Let us explain the dual of an object with more details through different algorithms in the following sections.

A. Therorical Comparisons of Computing Object Dual

Definitions 2, 3, 4, 5 extend Standard Hough Transform in two ways :

- the first one is to achieve object detection in virtual grid based on geometry shapes in computing its duals in the accumulator : digital objects pass through this virtual grid. Thus, parallel computing is applied to elementary shapes issued by this virtual grid in order to improve the execution time.
- the second one is to analyze the internal structure of points inside these duals in the accumulator data to highlight properties or characteristics for square, rectangle, triangle, hexagon, octagon recognition in the image. The future works will analyzed in details the internal structure of these dual.

Let us focus on the first case. Computing the dual of an hexagon and the dual of an octagon, could take more time than the dual of a square or the dual of a triangle. Because the number of internal diagonal segments in an hexagon or an octagon, building the area between different segment dual is important. For instance, according to SERE and others in [15], the dual of an hexagon and the dual of an octagon depend respectively on the dual of four internal segments and the dual of three internal segments. The dual of internal segments in objects can be realized through threads computing simultaneously the dual of each segment.

Our purpose is to use the dual of geometric objects in the parallel context to improve the execution time of Hough transform.

Hough Transform uses two spaces, namely an image space and a parameter space (or accumulator data). Proposed parallel Hough Transform takes into account these two spaces : that means the parallel reading of data in the image space and the parallel updating of the accumulator data

B. Analysis of Parallel Algorithm to Read the Data in the Image Space

Mesh generation establishes a virtual grid, superimposed on an image to bring out internal elementary virtual objects. For instance, An elementary object could be a rectangle, a triangle, an hexagon or an octagon. Fig. 1 presents an example of a virtual grid with triangles, created by Ouedraogo and and others in [22], an extension of the method proposed by Cheick and others in [14] to generate rectangles, where the triangles are its components.

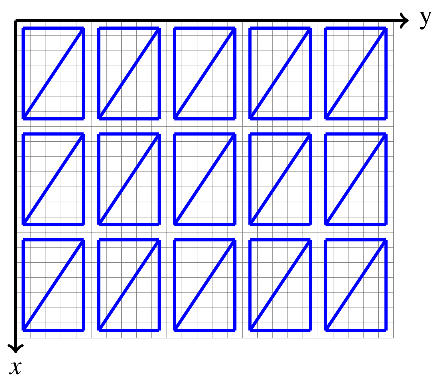

Fig. 1. An Example of the Triangular Mesh, Used by Ouedraogo and Others in [22] to Compute the Dual of a Triangle

Consider an image with (w x h) pixels. Algorithm 1 presents a generic serial execution. It uses a virtual mesh to perform an image, where each elementary object is referenced by the references Image(i, j). For instance, it takes particularly the dual of rectangles into account, as defined by SERE and others in [5] (see in preliminaries) where dual(Object x) corresponds precisely to dual(Rectangle x).

In this way, generalization is done by substitution of an object by any geometric shape .

Algorithm 1: Computing the Dual of Objects

```
Result: the dual of objects
image: Matrix ;
ob:Object ;
i, j : Integer ;
Image=pretreated(load(URL)) ;
for (i=1; i ≤ w; i++) do
    for (j=1; j ≤ h; j++) do
        ob=(Object) extract(Image(i, j));
        if light(ob) then
            dual(ob);
        end
    end
end
```

Let α and β be parameters corresponding to the number of threads to read data in the initial image. Data parallelism is introduced in algorithm 1 by OPENMP primitives as " # pragma omp parallel for" to define the parallel section as illustrated precisely in algorithm 2.

Algorithm 2: Data Parallelism with Openmp Primitives in a Pseudo-Code Algorithm

```
Result: the dual of objects
image: Matrix ;
ob:Object ;
i, j : Integer ;
Image=pretreated(load(URL)) ;
omp_set_num_threads($\alpha$) ;
# pragma omp parallel for ;
for (i=1; i $\leq$ w; i++) do
    omp_set_num_threads($\beta$) ;
    # pragma omp parallel for ;
    for (j=1; j $\leq$ h; j++) do
        ob=(Object) extract(Image(i, j));
        if light(ob)) then
            dual(Ob);
        end
    end
end
```

In both the algorithm 1 and the algorithm 2, there are different functions :

- the function load(String URL) loads an image, according to its URL;
- the function pretreated (Matrix m) uses operators of binarization and filtering on the image ;
- the function extract(Matrix m) allows to get an elementary object generated by the grid superimposed on the image;
- The function light(Object x) consists of verifying if or not all the pixels in the object ob respect certain contraints to make it elligible for computing its dual ;
- the function dual(Object x) executes effectively the dual of the object ob, in application of the standard Hough Transform to an object. It udpates the accumulator data, discussed forward in details in section III-C.

On the other hand, Fig. 2 presents a graph to modelize threads created by algorithm 2 with α=3 and β=3, following the Fork/Join model.

Let us study this model of graph in Fig. 2. Suppose that I=$\{1,2,...,w-1,w\}$ and J=$\{1,2,...,h-1,h\}$. We have obviously i $\in$ I and j $\in$ J where i, j are the counters of two loops :

- At the first level of loop with the counter i, the number of threads being α, each thread t_k manages at average $[\frac{w}{\alpha}]$ iterations with different value i $\in$ I. $[\frac{w}{\alpha}]$ is an integer value and k$\in\{1,2,...,\alpha-1,\alpha\}$.
  If $\alpha >$ w, there will be some k$\in\{1,2,...,\alpha-1,\alpha\}$-I, corresponding to the threads t_k that will never created. But thread creation depends on availability of resources such as microprocessor and memory. That means even If $\alpha \leq$ w, threads will be created according to availability of resources. The future works will study predictions of availability of resources to manage threads that could even used in thread management in operating system and in security to detect malicious software, using additional resources.
- At the second loop with the counter j $\in$ J, each thread with a new iteration on i $\in$ I, creates β threads to manage some iterations on j $\in$ J. For instance, the thread with the iteration i=1, creates β=3 threads to manage together all the iterations on j $\in$ J. Among β threads, in the same manner as the first level, each thread will compute $[\frac{h}{\beta}]$ iterations. As a conclusion, in considering an initial image I with its size height x width : The number of pixels is then determined by (height x width), to be submitted and shared for analysis by all the threads.

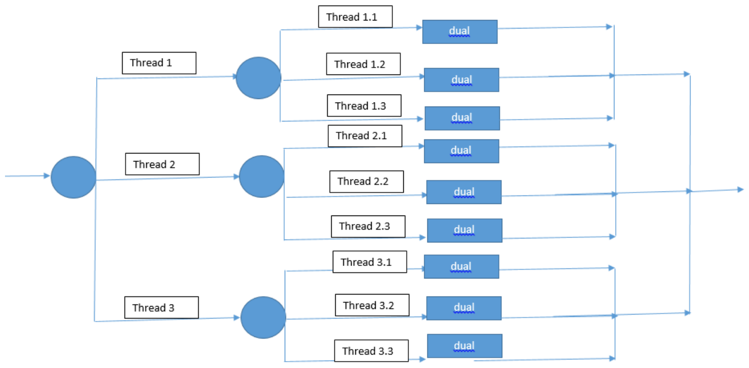

Fig. 2. Threads Created by Openmp Primitives with α=3 and β=3, Accordingly to Algorithm 2

image, corresponding to the size width. While the second loop is associated to the column of the image, corresponding to the size height . Finally, α x β threads work together on the same image and each thread will process approximately $[\frac{w}{\alpha}]$ x $[\frac{h}{\beta}]$ pixels.

The function dual(Object x) updates the accumulator and is studied in more details in section III-C.

C. Analysis of Parallel Algorithms to Update the Accumulator Data

This section focuses on data processing in the accumulator : it concerns precisely computation of dual(Object), as mentioned previously in calling, in algorithm 1. The number of threads used to update the accumulator is defined by the parameters γ, θ, as described in details in algorithm 3. Consider a accumulator with the size (wacc x lacc).

By analogy previously to section III-B related to the proposed algorithms, in algorithm 3, the number of γ threads is created, just before beginning the first loop with the counter m. These threads share iterations on a table that contains the number of lacc cells.

Algorithms 2 and 3 are generic : they can be specialized and adapted to any grid with its specific geometric shape, through computing object dual. For instance, the main specialized algorithm 6 in appendix is based on a triangular grid, implemented in Section IV.

Algorithm 3: dualObject(m, n : integer)

```
Result: the accumulator accc
acc: Matrix;
m, n : Integer ;
omp_set_num_threads($\gamma$) ;
# pragma omp parallel for ;
for (m=1; m $\leq$ lacc;m++) do
    omp_set_num_threads($\theta$) ;
    # pragma omp parallel for ;
    for (n=1; n $\leq$ wacc; n++) do
        if verify(m, n) then
            acc[m][n]=acc[m][n]+1;
        end
    end
end
return acc;
```

IV. Simulation and Discussions

Simulation considers the case study of a triangular mesh superimposed on an image. It follows and extends the initial works on the Hough Transform method, applied to triangular shapes in a serial execution, introduced by Ouedraogo and others in [22], similar to compute the dual of rectangles proposed by Traore and others in [14], also in a serial execution.

At the present time, computers have several microprocessors as Dual Core, Quad Core, i5, i7, i9 and super calculators with a shared memory that allows and increases performance to execute multiple threads.

In both the serial execution and the parallel execution, simulation uses a computer with the following characteristics :

- Processor : Intel(R) Core(TM) i5 CPU M 480@2.67GHz 2.67GHz
- Memory usable : 4,00 Go
- operating system : Kali Linux

Different tests have been realized on the initial images in figures 3 and 4. Illustrations take into account parameters as the threshold to indicate the number of vote in the accumulator, the rate of pixels in each triangle represented by the value α, the height and the base of the triangle and the number of threads.

The previous generic algorithms 2 and 3 have been detailed and specialized by the main algorithm 6 calling algorithms 4, 5 and 7, all in appendix for recognition in a triangular mesh. Implementations of parallelization have been realized in python, using opencv and pymp available in [17] to bring openmp-like functionality to python.

.

A. The Case of the Parameters (Threshold =100, the Number of Threads =2, The Height of the Triangle =4 Pixels, The Base of the Triangle=4 Pixels)

An analysis of the execution time for the parallel case and the serial case in varying the value α, has been summarized in the Tables I and II. It reveals effectively that the parallel computing of object dual is obviously better than the serial case.

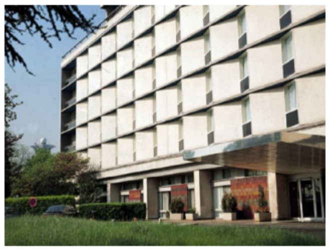
Fig. 3. A Building Image

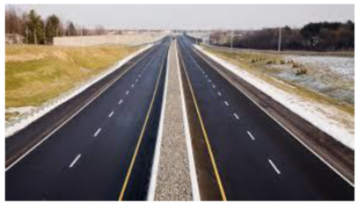
Fig. 4. A Road Image

TABLE I. The Execution Time (in Second) in the Serial Case and the Parallel Case for the Building Image with the Parameters (Threshold =100, The Number of Threads =2, The Height of the Triangle =4 Pixels, The Base of the Triangle=4 Pixels)

α	Number of detected lines	**Serial case**	**Parallel case**	Time difference
0.1	370	8.7	6.5	1.8
0.15	243	7.8	5.6	2.2
0.2	243	5.6	5.6	0
0.25	62	6.5	4.6	1.9
0.3	62	6.5	6.5	0
0.35	6	4.5	3.5	1
0.4	6	4.5	3.5	1
0.45	2	3	2.5	0.5
0.5	2	3	2.5	0.5
0.55	0	2.4	2.1	0.3

TABLE II. The Execution Time (in Second) in the Serial Case and the Parallel Case for the Road Image with the Parameters (Threshold =100, The number of Threads =2, The Height of the Triangle =4 Pixels, The Base of the Triangle=4 Pixels)

α	Number of detected lines	**Serial case**	**Parallel case**	Time difference
0.1	236	5	4	1
0.15	169	4.8	3.85	0.95
0.2	169	4.8	3.85	0.95
0.25	37	4.1	3.4	0.7
0.3	37	4.1	3.4	0.7
0.35	3	3.3	2.7	0.6
0.4	3	3.3	2.7	0.6
0.45	0	2.2	2.1	0.1

Difference between the processing time in the serial case and the parallel case, also appears in the Fig. 6 and 8, where the curve of the serial case is more up on the curve of the parallel case with the small values of α.

The Fig. 5 and 7 show decreasing the number of straight lines for increasing values of α.

Moreover, the surface of each triangle of the grid is determined by (height x base)/2 : that means $\frac{(4x4)}{2}$ =8 pixels.

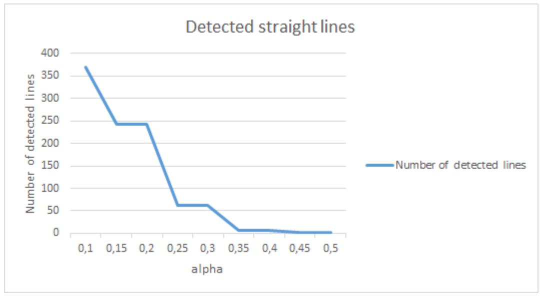

Fig. 5. The Number of Detected Lines in the Building Image Related to the Table I, Accordingly to the Values α

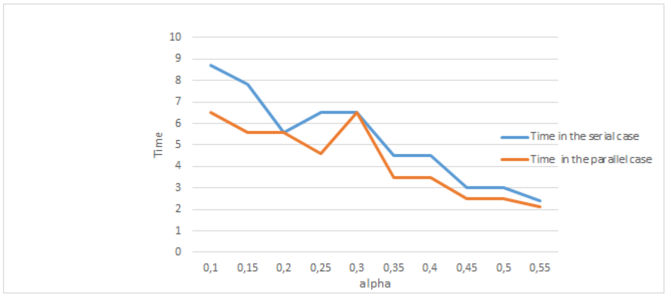

Fig. 6. The Curves of the Execution Time for the Building Image Related to the Table I

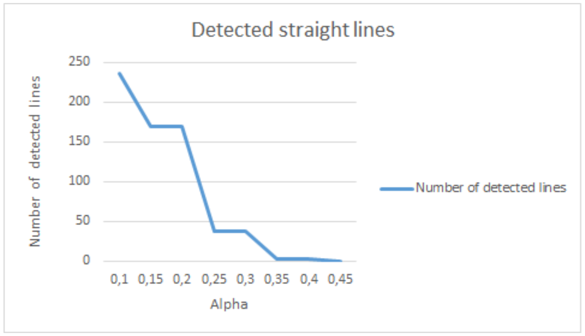

Fig. 7. The Number of Detected Lines in the Road Image Related to the Table II, Accordingly to the Values α

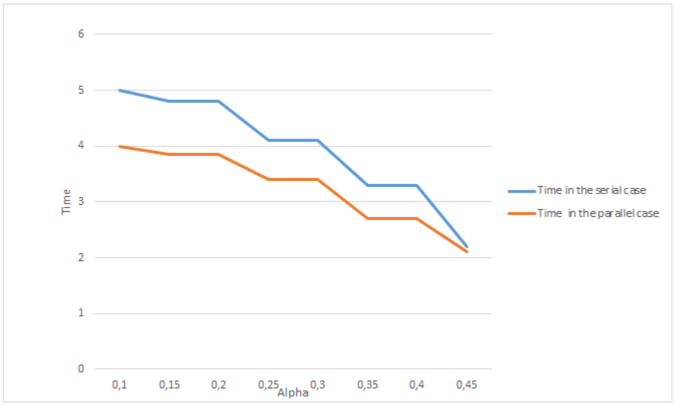

Fig. 8. The Curves of the Execution Time for the Road Image Related to the Table II

An image has (w x h) pixels, divided into $[\frac{(w.h)}{8}]$ triangles.

Valid triangles have the number of internal lit pixels, superior to [8 . α] : they will be considered for computing dual. For instance, with α=0.5, it appears at least (8 . α)=4 lit pixels for each triangle, to be valid.

Thus, if the number of valid triangles is the value nv, then the loose triangles will be ($[\frac{(w.h)}{8}]$-nv) in computing dual. That leads to loss approximately (($[\frac{(w.h)}{8}]$-nv).8) pixels (for all the image), representing a possibility to reduce the execution time.

If the value α is increasing, then the number of pixels (8 x α) to be considered for triangle validation will be reduced : that reduces effectively the number of straight lines to be detected and the execution time for any case (serial case and parallel case). In more details, with the value α increasing, the surface to be respected in each valid triangle will become small, but included more triangles that are engaged in the process : more straight lines passes through a reduced number of aligned triangles. More aligned triangles indicate reduced corresponding straight lines that pass through them, accordingly to the value of the threshold applied to the accumulator.

B. The Case of the Parameters (α =0.1, The Number of Threads =2, The Height of the Triangle =4 Pixels, The Base of the Triangle=4 Pixels)

With a fixed α =0.1, in varying the threshold, the Tables III and IV show respectively the execution time for the building image and for the road image : it presents naturally reduction in the number of detected lines and the execution time. Because the number of cells, having the maximum vote is reduced and the time taken to read the accumulator data is short for the serial case and for the parallel case.

TABLE III. The Execution Time (in Second) in the Serial Case and the Parallel Case for the Building Image with the Parameters (α =0.1, The Number of Threads =2, The Height of the Triangle =4 Pixels, The Base of the Triangle=4 Pixels)

Threshold	Number of detected lines	**Serial case**	**Parallel case**
100	370	8.7	6.5
125	53	7.6	5.6
150	13	7.6	5.4
175	7	7.6	5.3
200	5	7.4	5.3
225	2	7.8	5.3
250	1	7.4	5.4
275	0	7.7	5.4

TABLE IV. The Execution Time (in Second) in the Serial Case and the Parallel Case for the Toad Image with the Parameters (α =0.1, The Number of Threads =2, The Height of the Triangle =4 Pixels, The Base of the Triangle=4 Pixels)

Threshold	Number of detected lines	**Serial case**	**Parallel case**
100	236	5	4
125	24	4.83	3.8
150	6	4.8	3.7
175	1	4.7	3.7
200	0	4.7	3.7

C. The Case of the Parameters (Threshold =100, Number of Threads =2, The Height of the Triangle =8, The Base of the Triangle=8)

Tables V and VI illustrate the case of a large triangle with the parameters (height = 8 pixels, base =8 pixels).

TABLE V. THE EXECUTION TIME (IN SECOND) IN THE SERIAL CASE AND THE PARALLEL CASE FOR THE BUILDING IMAGE WITH THE PARAMETERS (THRESHOLD =100, NUMBER OF THREADS =2, THE HEIGHT OF THE TRIANGLE =8, THE BASE OF THE TRIANGLE=8)

α	Number of detected lines	**Serial case**	**Parallel case**	Time Difference
0.1	140	7	5.3	1.7
0.15	79	6.7	4.8	1.9
0.2	57	6.2	4.5	1.7
0.25	19	5.3	4	1.3
0.3	5	4.1	3.2	0.9
0.35	3	3.1	2.5	0.6
0.4	3	2.4	2	0.4
0.45	3	1.9	1.8	0.1
0.5	3	1.8	1.7	0.1
0.55	3	1.8	1.7	0.1
0.6	3	1.8	1.7	0.1
0.65	3	1.8	1.7	0.1
0.7	3	1.8	1.7	0.1
0.75	3	1.8	1.7	0.1

TABLE VI. THE EXECUTION TIME (IN SECOND) IN THE SERIAL CASE AND THE PARALLEL CASE FOR THE TOAD IMAGE WITH THE PARAMETERS (THRESHOLD =100, NUMBER OF THREADS =2, THE HEIGHT OF THE TRIANGLE =8, THE BASE OF THE TRIANGLE=8)

α	Number of detected lines	**Serial case**	**Parallel case**	Time Difference
0.1	97	4.5	3.6	0.9
0.15	55	4.3	3.5	0.8
0.2	29	4.1	3.2	0.9
0.25	8	3.6	3	0.6
0.3	0	2.9	2.6	0.2

As a conclusion, for any case of triangle, the difference between the parallel execution time and the serial one will be more large, in favour to the parallel case that has a short reduced time with the small values of α.

D. Straight Line Recognition in Real Images

This section is focusing on the application of parallel Hough Transform to real images.

1) Straight Line Recognition in the Building Image with a Triangle 4 x 4: For the building image, the accumulator data and the results of straight line recognition are respectively illustrated in Fig. 9 in Fig. 10.

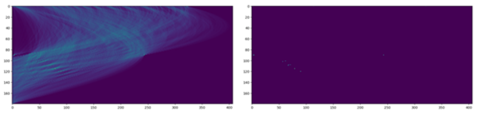

Fig. 9. The Accumulator Data after Application of the Parallel Computing: On the Left Image, The Sinusoid Curves and on the Right the Maximum Votes Superior to the Threshold =150

2) Straight Line Recognition in the Road Image with a Triangle 4 x 4 : For the road image, the accumulator data and the straight line detection also appear respectively in Fig. 11 and 12, in considering the parameters (thread=2, threshold=100, α=0.3).

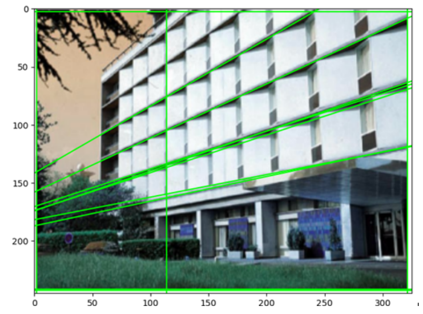

Fig. 10. Straight Line Recognition in the Building Image with the Parameters (Threads =2, Threshold = 150, α =0.1)

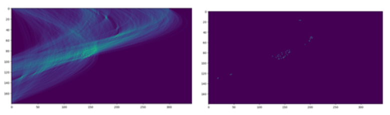

Fig. 11. The Accumulator Data after Application of the Parallel Computing: On the Left Image, The Sinusoid Curves; On the Right Image the Maximum Votes is Superior to the Threshold =100

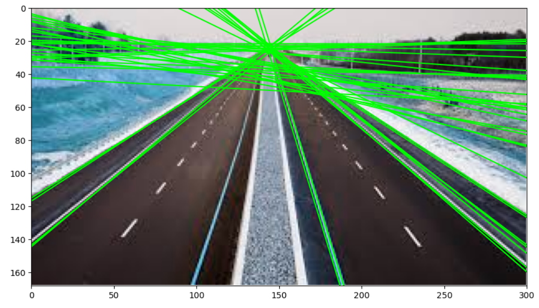

Fig. 12. Straight Line Recognition in the Road Image with the Parameters (Threads =2, Threshold = 100, α =0.3)

V. CONCLUSION AND PERSPECTIVES

A new parallel Hough method has been proposed effectively to compute the dual of objects based on the standard Hough transform, through algorithms using openmp library to create threads. An image is analyzed with a meshing technique and different threads, computing the dual of virtual generated objects for updating the accumulator data.

Experimental results have been realized with the pymp library in python for parallelization and reveal reduction on the execution time in parallel execution than in the serial one. As outputs, the detection of straight lines is effectively realized in a building image and a road image. But optimizations of the proposed algorithms to improve more the execution time in the parallel case and comparisons with similar parallel methods still remain to do .

In perspectives, the study of parallelism will focus on straight line recognition in a rectangular grid, in an hexagonal grid or in an octagonal grid.

The future works will also analyze in detail, the internal structure of object dual to determine recognition for rectangles, triangles, hexagons and octagons in the image.

All these works will contribute forward to strengthen the building of a toolkit for the Hough Transform method, being integrated to a software for image processing, to bring out extensions for new functionalities, as a plug-in or a library for programming languages.

REFERENCES

[1] H. Maitre, *A review on Hough Transform*, Traitement du signal, 1985, volume 2, number 4, pages 305-317,

[2] M.Dexet,*Architecture d'un modeleur géométrique à base topologique d'objets discrets et méthodes de reconstruction en dimensions 2 et 3*, Université de Poitiers (France) Thèse en informatique et applications, 2006, decembre

[3] P.-V.-C. Hough, *Method and means for recognizing complex patterns*, In United States Pattent, 1962 volume 3069654, 47-64,

[4] R.O. Duda and P.E. Hart, *Use of the hough transform to detect lines and curves in pictures*, Communications of the ACM, 1972,15(1),11-15

[5] A. Sere, O. Sie and E. Andres, *Extended Standard Hough Transform for analytical line recognition*, 6th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT), Sousse, Tunisia, 2012, pp. 412-422, doi: 10.1109/SETIT.2012.6481950.

[6] M Dexet and E. Andres, *A generalized preimage for the digital analytical hyperplane recognition*,Discrete Applied Mathematics, 2009, volume 157, Issue 3, pages 476-489,

[7] Abdoulaye SERE, Dario COLAZZO, Oumarou SIE, *A Hough Transform Based On a Map-Reduce Algorithm*, International Journal of Engineering Research and Application ISSN : 2248-9622, Vol. 6, Issue 8 (part 2)

[8] Mateus Coelho, Dylan Sugimoto, Gabriel Melo, Vitor Curtis and Juliana Bezerra *A MapReduce based Approach for Circle Detection*, In Proceedings of the 14th International Conference on Software Technologies - Volume 1: ICSOFT, 454-459, 2019 , Prague, Czech Republic

[9] J.P.Reveillès, *Combinatorial pieces in digital lines and planes*. In proceeding SPIE 2573, Vision Geometry volume IV, issue 23, (1995)

[10] E.Andres , *Discrete linear objects in dimension n : the standard model*. In Graphical Models, volume 65, issue 1-3, pages 92-111(2003).

[11] Angella Yao, Juergen Gall, Luc Van Goo, *A Hough Transform-Based Voting Framework for Action Recognition*, IEEE Conference on computerVision and Pattern Recognition (CVPR 10), (2010).

[12] Gabriel Fanelli, Juergen Gall, Luc Van Goo, *Hough Transform-based Mouth Localization for Audio-Visual Speech Recognition*, British Machine Vision Conference, September (2009).

[13] Séré A., Ouedraogo T. F., Zerbo B.; *An improvement of the standard Hough transform method based on geometric shapes*, Future of Information and communication conference (FICC), 2018, 5-6

[14] Cheick Amed Diloma Gabriel TRAORE, Abdoulaye SERE, *Straight Line Detection with the Hough Transform Method based on a Rectangular Grid*, Fifth International Conference on Information and Communication Technology for Competitive Strategies (ICTCS-2020)

[15] Abdoulaye SERE, Yaya TRAORE, Frederic T OUEDRAOGO, *Towards New Analytical Straight Line Definitions and Detection with the Hough Transform Method*, International Journal of Engineering Trends and Technology 62.2 (2018): 66-73.

[16] Abdoulaye SERE, Cheick Amed Diloma Gabriel TRAORE, Yaya TRAORE, Oumarou SIE, *Towards Traffic Saturation Detection Based on the Hough Transform Method*. In: Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2. FTC 2020. Advances in Intelligent Systems and Computing, vol 1289. Springer, Cham.

[17] Awani Kendurkar, Mohith J. *A Comparative Analysis of Parallelisation Using OpenMP and Pymp for Image Convolution* . International Research Journal of Engineering and Technology (IRJET) ISSN: 2395 0056 Volume : 08 Issue: 09, Sep 2021 www.irjet.net p ISSN: 2395 0072

[18] Jack E Bresenham, *Algorithm for computer control of a digital plotter*, IBM Systems Journal, ACM.

[19] Wright William E., Rendering, *Parallelization of Bresenham's Line and Circle Algorithms*, IEEE Computer Society Press, https://doi.org/10.1109/38.59038

[20] M. W. Spratling, *A neural implementation of the Hough transform and the advantages of explaining away*, Image and Vision Computing, doi: 10.1016/j.imavis.2016.05.001

[21] Jošth R., Dubská M., Herout A., Havel J. (2011) *Real-Time Line Detection Using Accelerated High-Resolution Hough Transform*. In: Heyden A., Kahl F. (eds) Image Analysis. SCIA 2011. Lecture Notes in Computer Science, vol 6688. Springer, Berlin, Heidelberg

[22] Ouedraogo, M., Sere, A., Some, B.M.J., Traore, C.A.G.D, *Straight-Line Recognition Using a Triangular Grid*. In: Arai, K. (eds) Advances in Information and Communication. FICC 2022. Lecture Notes in Networks and Systems, vol 438. Springer, Cham.

APPENDIX

Algorithm 4: Building a triangular grid

```
Function meshing2(Nl, Nc,h, b: integers): table of integers
    Variables: tab: table of 6 integers
    Output: $tab[1]$ ($tab[3]$): base (height) of the triangle
            of our meshing ;
    $tab[0]$ : number of parts of length $tab[3]$ on the line ;
    $tab[2]$: number of parts of length $tab[1]$ on the
    column ;
    $tab[4]$ : the remainder of the Euclidean division of $Nl$
    by $h$ ;
    $tab[5]$ : the remainder of the Euclidean division of $Nc$
    by $b$ ;
    begin
    $tab[1] \leftarrow b$;
    $tab[3] \leftarrow h$;
    $tab[0] \leftarrow int[Nl/h]$;
    $tab[2] \leftarrow int[Nc/b]$;
    $tab[4] \leftarrow Nl \mod h$;
    $tab[5] \leftarrow Nc \mod b$;
    return $tab$;
```

Algorithm 5: Rate of lit pixels

```
Function count(img_pymp : image, A, B, C : 3 tables of
  two integers containing the coordinates of the vertices of
  the triangle) : real
    Variables: $k$, $l$ : integers;
    $D$ : table of two integers
    begin
    $k \leftarrow 0$;
    $l \leftarrow 0$;
    $D \leftarrow A$;
    if B[1]≥ A[1] and C[0] ≥ A[0] then
        for $x$ between $A[0]$ and $C[0]$ do
            $DE \leftarrow \frac{|x-C[0]| \times |A[1]-B[1]|}{|A[0]-C[0]|}$;
            $y_0 \leftarrow A[1] + round(DE)$;
            for $y$ between $A[1]$ and $y_0$ do
                if $img\_pymp[x,y] \neq 0$ then
                    $k \leftarrow k+1$;
                else
                    $l \leftarrow l+1$ ;
    if B[1]] ≤ A[1] and C[0] ≤ A[0] then
        for $x$ between $C[0]$ and $A[0]$ do
            $DE \leftarrow \frac{|x-C[0]| \times |A[1]-B[1]|}{|A[0]-C[0]|}$;
            $y_0 \leftarrow A[1] - E[DE]$;
            for $y$ between $y_0$ and $A[1]$ do
                if $img\_pymp[x,y] \neq 0$ then
                    $k \leftarrow k+1$;
                else
                    $l \leftarrow l+1$ ;
    return $\frac{k}{k+l}$;
```

Algorithm 6: Recognition of discrete lines

```
Pre-condition: $0 < \alpha < 1$, $0 < threshold$
Data: $\alpha$
Variables: tab : table of 6 integers ;
A, B, C : tables of two integers ; accum_row, accum_column, irho, Nl, Nc : integers;
accum : matrix of dimensions "$accum\_row \times accum\_column$";
threshold, $\alpha$, dtheta, rho, theta, DE : real;
Begin
% import of pymp library
import pymp;
% image reading
imge=cv2.imread('image.jpg');
img= pretraited image;
% the dimensions of the image ;
$Nl \leftarrow$ number of rows of image; $Nc \leftarrow$ number of columns of image;
%create a pymp type matrix
$img\_pymp \leftarrow pymp.shared.array([Nl,Nc])$;
% $\alpha$ : the rate of lit pixels from which the triangle is selected ;
$\alpha = 0.8$ ;
% threshold : the minimum number of votes ;
$threshold \leftarrow 150$ ;
% the dimensions of the virtual triangles to be entered manually;
$h \leftarrow 4$ ; $b \leftarrow 4$ ;
% the dimensions of the matrix "accum_pymp"
$accum\_row \leftarrow 180$ ; $accum\_column \leftarrow E(\sqrt{(}Nc^2 + Nl^2))$;
$dtheta = \pi/180$ ;
% creation of the pymp type accumulator matrix
$accum = pymp.shared.array([Ntheta,Nrho])$;
$accum\_seuil = pymp.shared.array([Ntheta,Nrho])$;
$tab \leftarrow meshing(Nl,Nc,h,b)$ ;
$H = Nl - tab[4]$ ; $L = Nc - tab[5]$ ;
% assign the pixel values of the pre-processed image to the matrix of type pymp
for x in range(Nl): do
    for y in range(Nc) do
        $img\_pymp[x,y] \leftarrow img[x,y]$

% walk through all the triangles of the grid without taking into account any residues and updating the
  accumulator
%
pymp.config.nested=False;
% the maximum number of threads allowed
pymp.config.thread_limit = 4;
with pymp.parallel(2) as p :
    for x in tab[3] and Nl with a step of tab[3]+1 do
        for y between tab[1] and Nc with a step of tab[1]+1 do
            $A[0] = x - tab[3]$ ; $A[1] = y - tab[1]$ ; $B[0] = x - tab[3]$ ; $B[1] = y$ ; $C[0] = x$ ;
              $C[1] = y - tab[1]$ ; $D = A$;
            if B[1]¿A[1] and C[0]¿A[0] then
                if count(img_pymp, A,B, C)≥ $\alpha$ then
                    for z between A[0] and C[0] do
                        $DE \leftarrow \frac{|x-C[0]| \times |A[1]-B[1]|}{|A[0]-C[0]|}$;
                        $y_0 \leftarrow A[1] + round(DE)$;
                        for t between A[1] and $y_0$ do
                            Acc_update();

            $A[0] = x$ ; $A[1] = y$ ; $B[0] = x - tab[3]$ ; $B[1] = y$ ; $C[0] = x$ ; $C[1] = y - tab[1]$ ; $D = A$;
            if B[1]¡A[1] and C[0]¡A[0] then
                if count(img_pymp, A,B, C)≥ $\alpha$ then
                    for z between C[0] and A[0] do
                        $DE \leftarrow \frac{|x-C[0]| \times |A[1]-B[1]|}{|A[0]-C[0]|}$;
                        $y_0 \leftarrow A[1] - E[DE]$;
                        for t between E[$y_0$] and A[1] do
                            Acc_update();

    if $tab[4] \neq 0$ then
        for z between H and Nl do
            for t between 0 and Nc do
                Acc_update();

    if $tab[5] \neq 0$ then
        for z between 0 and Nl do
            for t between L and Nc do
                Acc_update();

Search for maxima in the accumulator;
Line drawing;
End
```

Algorithm 7: Accumulator update

```
Procedure Acc_update(): void
    % Updating the accumulator
    if $img\_pymp[z,t] \neq 0$ then
        for itheta in p.range(accum_row) do
            $theta \leftarrow itheta \times dtheta$;
            $rho \leftarrow t \times cos(theta) + z \times sin(theta)$;
            $irho \leftarrow int(rho)$;
            if $irho > 0$ and $irho < accum\_column$ then
                $accum[itheta][irho] \leftarrow$
                  $accum[itheta][irho] + 1$
```