Computing the Component-Labeling and the Adjacency Tree of a Binary Digital Image in Near Logarithmic-Time

Fernando Díaz del Río[1], Helena Molina-Abril[2], and Pedro Real[2]

[1] Department of Computer Architecture and Technology, University of Seville, Seville, Spain

[2] Department of Applied Mathematics, University of Seville, Seville, Spain

habril@us.es

Abstract. Connected component labeling (CCL) of binary images is one of the fundamental operations in real time applications. The adjacency tree (AdjT) of the connected components offers a region-based representation where each node represents a region which is surrounded by another region of the opposite color. In this paper, a fully parallel algorithm for computing the CCL and AdjT of a binary digital image is described and implemented, without the need of using any geometric information. The time complexity order for an image of $m \times n$ pixels under the assumption that a processing element exists for each pixel is near $O(log(m+n))$. Results for a multicore processor show a very good scalability until the so-called memory bandwidth bottleneck is reached. The inherent parallelism of our approach points to the direction that even better results will be obtained in other less classical computing architectures.

Keywords: Component-Labeling · Adjacency tree · Digital image Parallelism

1 Introduction

Connected component labeling (CCL) of binary images is one of the fundamental operations in real time applications, like fiducial recognition [6] or classifying objects as connected components (CCs). The labeling operation transforms a binary image into a symbolic matrix in which every element (pixel) belonging to a connected component is assigned to a unique label. Currently, there are mainly four classes of CCL algorithms: Multi-scan algorithms, Two-scan algorithms, Tracing-type algorithms and Hybrid algorithms mixing the previous ones. All of them (including the fastest one) use raster or Tracing-type approaches,

This work has been supported by the Spanish research projects MTM2016-81030-P (AEI/FEDER, UE) and TEC2012-37868-C04-02 of Ministerio de Economía y Competitividad and the VPPI of the University of Seville.

scanning the whole binary image or its contours in a sequential manner. They begin labeling the first pixel and so the second one as a function of the first pixel label. This local processing proceeds progressively until the last pixel is reached. This fact necessarily implies data dependencies between the labeling of one pixel and the previous one, which prevents these methods from using a pure parallel approach. In terms of time complexity, this means that linear order $O(N)$ (being N the number of pixels) cannot decrease independently of the number of available processing units.

In relation to the representation of digital objects or, alternatively, binary digital images, various topological models have been exhaustively used. Adjacency trees (also called topological, inclusion or homotopy trees [2,16,17], and here AdjT, for short) offer a classical region-based representation in terms of rooted tree of certain topological and spatial properties of the connected components in a binary image. Within an AdjT, each node represents a distinct foreground (FG) or background (BG) component, and an edge between two nodes means that one of them is surrounded by the other. The root in an AdjT always represents the unique BG component "surrounding" the image (if it does not exist, it can be artificially created) and two 2D binary digital images are topologically equivalent if and only if their AdjTs are equivalent. An example of an AdjT of the binary image in Fig. 2 is shown in Fig. 4 (left). Aside from image understanding [18] and mathematical morphology applications [7,10,15], AdjTs have encountered exploitation niches in geoinformatics, dermatoscopics image, biometrics, etc. (see [3,5,6] for instance). Therefore, finding fast algorithms for segmenting and computing the AdjT of a 2D digital binary image is crucial for solving important problems related to topological interrogation in the current technological context.

It is evident that the compression of those nodes of a CCL tree (CCLT) satisfying the neighboring condition "having the same color", directly yields to the AdjT. In this paper we present a novel method for computing CCL and AdjT but reducing the number of operations, so that computation time and memory consumption are sensibly decreased, whereas the degree of parallelism is extended to every single pixel.

2 Related Works

Parallel implementations for computing topological magnitudes can be achieved via two approaches. On the one hand, there is some space for parallelism when codifying scan or tracing-based CCL algorithms. These algorithms contain two main stages: the scanning phase where provisional labels are assigned to pixels depending on their neighbors, and some kind of union-find technique to collect label equivalence information in the previous assignment. For example, dividing the image into strips is a classical data partition technique for obtaining parallelism. The second stage must then use a more sophisticated union-find technique for the provisional labels to get to the CCL. There is also some room for parallelism in this phase, and many works have addressed different variations (see [8,10,15]) including tuning parallel algorithms for specific computers (see [1]).

On the other hand, topology is the ideal mathematical scenario for promoting parallelism in a natural way, although it drives to less classical parallelism approaches. The nature of the topological properties is essentially qualitative and local-to-global, having the additional advantage that its magnitudes are robust under deformations, translations and rotations. Nevertheless, the results in the literature in that sense are rare. Up to now, the only topological invariant that has been calculated using a fully parallel computation is the Euler number [3]. Other authors have recently proposed other parallel algorithms that compute some aspects of the homological properties of binary images [13]. In [4], a digital framework for parallel topological computation of 2D binary digital images based on a sub-pixel scenario was developed, modeling the image as a special abstract cell complex [11], in order to facilitate the generalization of this work to higher dimensional images. In addition, some software libraries of flexible C++ (RedHom [16]) have appeared for the efficient computation of the homology of sets. These libraries implement algorithms based on geometric and algebraic reduction methods.

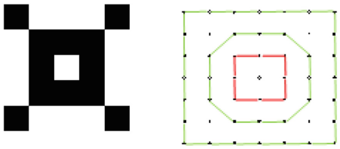


Fig. 1. Holes of 4-adjacent CCs are 8-adjacent CCs and vice versa for 2D binary digital images based on square pixel.

In relation to previous works, we construct our scaffolding on the basis of the two following basic topological properties: "being adjacent to" and "being surrounded by". Moreover, we take advantage of the powerful duality properties that the topological invariants of connected components and holes have in the context of 2D binary digital images based on square pixel. In other words, we exploit that the holes of 4-adjacent CCs must be 8-adjacent CCs and vice versa (see Fig. 1). As a result, all the algorithms of this work use simple connectivity graphs (CGs) as their basis. Our simplification allows us to reduce the number of operations, the computation time and the memory consumption, and to extend the degree of parallelism to every single pixel.

3 A Convenient Topological Framework for Computing CCL and AdjT

In a few words, topological analysis of digital images studies their degree of connectivity, defining fixed adjacency relations between pixels as "local neighborhood measures". It is obvious that a unique tree covering all the pixels of an image can be built (no matter the intensity of the pixels) going through all the pixels of the image using always, for example, the North direction, until the upper border is found and then change to East direction until the most north-eastern pixel is reached. Pixels that connect different colors should be marked as candidates of frontiers between CCs (region frontiers). Any tree covering the image plus the region frontier candidates is then an instance of a connectivity tree that holds the complete information of the binary image. Note that for 2D binary digital images, there are two types of connectivity we must deal with, which are: CCs and holes. However, these two concepts can be reduced to one for a binary image since a hole can be seen as a CC that is surrounded by another CC of different color. Concerning topological analysis, all local neighboring conditions used here are derived from 4-adjacency relations. Concerning the output of this processing, the white nodes of the AdjT are 4-CCs and the black ones are 8-CCs. Let us limit ourselves to say that a hole of a 4-connected FG object can be interpreted as an 8-connected component of the BG. For this reason the algorithm presented here considers the FG with 4-adjacency and the pixels of the BG with 8-adjacency. Then, a minimal tree can be computed having always the correct number of FG 4-connected components and BG 8-connected components.

Using a combinatorial optimization process, it is possible to find a connectivity tree in which every CC has only one region frontier pixel. Each one of these special pixels marks a bond between two neighboring CCs of different color. For example, in Fig. 3 these pixels are marked with a number, which is the representative label of each CC. In the figure, black regions can be compressed to the pixels labeled with the numbers 92, 165, 194, 200, 226, 258, whereas white areas are represented by the pixels 49, 77, 179. We can say that these region frontier pixels are the "attractors" of each one of the trees that contain a CC. In this paper, a connectivity tree that holds this property is called a connected component labeling tree (CCLT). Following the path given by this compression (that is going along the connectivity tree) the different frontier pixels can be found. In the light of the above, in this paper a fully parallel algorithm for computing the CCL (and the AdjT) of a binary digital image based on square pixels is implemented through the building of one CCLT. This is achieved without the need of using any geometric information. Once the combinatorial optimization process has been carried out, the whole image (and, thus, every CC) can be compressed to just one pixel. Hence, the compression of those nodes of a CCLT whose neighboring condition is "having the same color" directly yields to the AdjT. To carry out this process, the local neighboring connectivity information (that is, those of the adjacent pixels) is first transformed into a global connectivity graph (CG) tracking a unique direction in a number of iterations equal to $log(m+n)$ (being m the image width, and n the image height). Finally, CCs are extracted by fusing the regions of the previous global graph in a parallel way using a reduced number of iterations (see Sect. 4). A more detailed example is shown in Fig. 2. On the left a face-like binary image is depicted, whereas on the right, its corresponding black (FG) 4-connected components are identified.

Note that 4-adjacency prevents the nose from being connected with the glasses. Each FG CC has been assigned a black representative pixel (marked with little triangles in Fig. 2 right), which can be called "attractor" as every link of the simplified CCLT collapses in them. Conversely, FG holes can be represented by attractors of the BG CCs. These are marked with little downwards arrows, and can be called BG attractors, for analogous reasons (that is, all the BG CC collapses in it). An additional BG attractor in the most left-bottom corner has been added to represent a virtual attractor for the whole image. Whereas the meaning of an attractor is intuitive and sufficient for understanding these concepts, an exact definition of the attractors is given in Sect. 4.

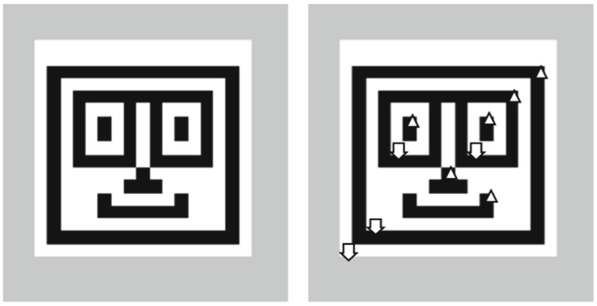
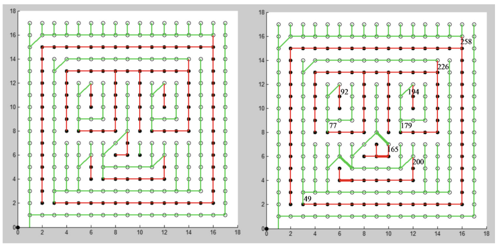


Fig. 2. Left: original face-like image. Right: FG attractors (little triangles) and BG attractors (downwards arrows).

The results for Fig. 3 (left) returned by our algorithm that generates the CCL and the AdjT (see next section) are summarized in the table in Fig. 3 (right) and Table 1. These tables contain the FG and BG ordered pairs. The silhouette contained in Fig. 2 has a total of 6 FG attractors and 4 BG attractors. For each FG attractor, FG attractor pair table (Fig. 3) gives the BG attractor that goes to (that is, the hole that surrounds this component) and some additional information (area and perimeter of each region). Numbers for attractors are pixel labels, that is, linear indexes over the image (following a column convention, that is, the linear indexes for a matrix when all its elements are numbered following its columns). The perimeter is computed by counting the 8-adjacent pixels that enclose a CC (which means that a CC composed of a single pixel has a perimeter of 8 pixels).

Likewise, BG attractor pair table (Table 1, left) gives the FG attractor where each BG attractor points to (that is, the black component that surrounds a hole). Note that AdjT can be automatically extracted from these tables. The matrix of Table 1 (right) is a possible compact representation of the AdjT and it gives the crossed including relations of FG and BG attractors using this notation: rows are FG attractor indexes (labels); columns are BG attractor labels; 1 means that a BG attractor is included on an FG attractor, −1 means that an FG attractor is included on a BG attractor. Finally Fig. 4 (left) comprises a graphical



FG Attr. index	BG Attr. Index where it points to	FG CC area (in pixels)	FG CC perimeter (in pixels)
92	77	2	10
165	49	4	14
194	179	2	10
200	49	9	24
226	49	37	48
258	17	54	62

Fig. 3. Left: final CCLT having the minimal number of FG 4-connected components and BG 8-connected components. Indexes of BG/FG attractors are on the right of each attractor. Right: F_{TABLE}: the corresponding FG attractor table. Numbers are the linear indexes for a matrix numbered following its columns.

Table 1. Left: AdjT of Fig. 3. Rows are FG attractor labels; columns are BG attractor labels; values are: 1 = BG attractor included on the FG attractor, -1 = FG attractor included on the BG attractor. Right: B_{TABLE}: BG attractor table of Fig. 3. Numbers are the linear indexes for a matrix numbered following its columns

BG Attractor index	FG Atractor index where it points to	BG CC area (in pixels)	BG CC perimeter (in pixels)
49	258	82	54
77	226	10	18
179	226	10	18

	49	77	179
92	0	-1	0
165	-1	0	0
194	0	0	-1
200	-1	0	0
226	-1	1	1
258	1	0	0

representation of a weighted AdjT of Fig. 3. Filled (empty, resp.) circles are FG (BG, resp.) CC. The notation for the numbers $i : \{a, p\}$ represents: $i = index(label)$, $a = area$ of the CC, $p = perimeter$ of the CC.

4 A Parallel Algorithm for Building the CCLT

As previously explained, CG should pair any FG pixel with another FG pixel, except those that are the possible attractors of a CC. The key point is that each pairing must be done in a convenient direction so that only one unpaired pixel (the attractor) exists for each CC. In this case, if we followed the links from any pixel, this stream of links would fall on this attractor. In Fig. 5 (left) a simple shape is depicted where every FG pixel has been linked following a simple criterion: if its North neighbor were an FG pixel, it would be linked to it; if it were not, it would be linked with its East adjacent pixel if it had the FG color; if East adjacent pixel had not FG color either, it is marked as a possible attractor and connected to the north BG pixel. This pairing is called here “North-then-East criterion” or simply NE criterion. For the 8-adjacent BG pixels it is convenient that the criterion uses the opposite direction: SswW (South-then-Southwest-then-West), thus obtaining a set of possible BG attractors. This would complete a possible CG. Hence, a false FG attractor can be defined as a pixel whose north and east adjacent BG pixels points to two different BG attractors (little triangles in Fig. 7, left). Likewise, a false BG attractor is a pixel whose south, southwest and west adjacent FG pixels points to two different FG attractors (downward arrows in Fig. 7, left).

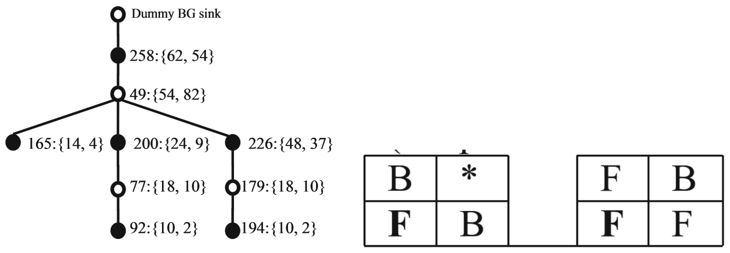


Fig. 4. Left: a graphical representation of the AdjT of Fig. 3. Filled (empty, resp.) circles are FG (BG, resp.) CCs. Notation $i : \{a, p\}$ represents: i = index (label), a = area of the CC, p = perimeter of the CC. Right: The 2×2 patterns that represent attractors in F*. B* matrices are resp. (pixel of reference in bold).

However, things are not so simple in the general case, because many pixels can have both: a North and an East neighbor, and only one must be selected for the pairing. For instance, the right picture of Fig. 5 shows a spiral shape where the direction of the every pairing was done in an ad-hoc form, so that only one unpaired pixel remains (the most northeast FG pixel). Note that the NE criterion is not preserved for many FG pixels (correspondingly with respect to the SswW criterion for BG pixels). The same for the BG: its only unpaired pixel is the dummy attractor on the most southwest corner. The key is how these directions are selected in parallel to produce the desired pairing. Global information about the shape of every CC is needed to choose this correctly, that is, it is impossible a priori to discover which pairing must select every pixel to get to the correct CCLT. Nevertheless, there exists a high amount of parallelism in this process. In order to get to the CCLT, we propose two main steps: Generating a CG as parallel as possible; and secondly, transforming this CG into a correct CCLT through the cancellation of pairs of a false FG attractor with a BG one. This process must be iterated until no false attractor remains.

As stated before, an algorithm that tries to extract global information of an image must include some pieces to search the relation between remote parts of the image. Using the properties of the tree-like structures, those sequential pieces can be reduced significantly. Figure 6 draws a sketch of the process to obtain the minimal tree structures needed to extract the CCLT and AdjT of an image, preserving its combinatorial nature. From the image I, a local CG based on the local information of each pixel can be first computed. Here "local" means that

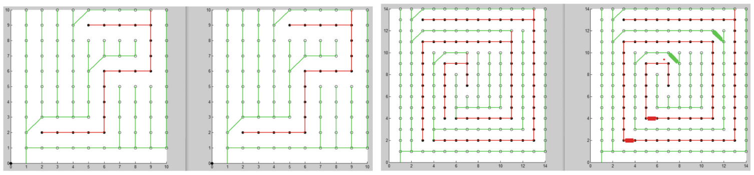


Fig. 5. Two figures with one CC. Left: a simple shape where the pixel pairing is accurate by NE criterion. Right: a spiral, where this simple criterion is not valid.

the computation of every link is based only on the values of its (e.g. 4-) adjacent pixels. This tree will contain the links from each pixel to its immediate neighbors. Then, through successive iterations we can get to a different global CG. Here "global" means that each pixel knows the link to its (possibly far) attractor in this CG. FG, BG attractors of the initial CG are not yet the true attractors (see Fig. 7, where the FG CC has resulted in two attractors; one of them must be false). This tree must be transformed so as to contain only true attractors, which means that we have reached the correct CCLT. Using the CCLT, every attractor can be related with another attractor of opposite color that contains the first. This is a representation of the AdjT.

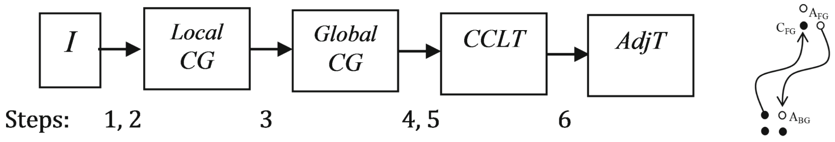
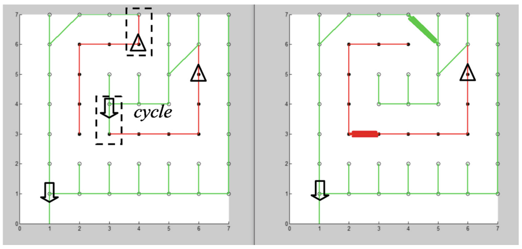


Fig. 6. Left: steps involved in a sequential CCLT building. Right: a cycle from an FG attractor A_{FG} to C_{FG} through a BG attractor A_{BG}, where $C_{FG} = A_{FG}$.

From now on, let us suppose that the border of the whole image is composed of BG pixels, which belongs to an external dummy BG attractor. The aim is to build an optimal gradient vector field with only one (FG or BG) attractor for each (resp. FG or BG) CC. The first step of Fig. 6 computes an initial CG of the image I. The computation of every link is exclusively based on the values of its adjacent pixels. The rest of the steps are necessary to transform this CG into a CCLT detecting the representative FG, BG attractors. The second step determines which pixels are possible BG/FG attractors, that is, those that have a link that connects FG and BG pixels. The key point is that the FG graph must be built on the opposite direction than that of the BG. Next, the third step introduces global relations between pixels and attractors, so that the attractor for each pixel is determined when following the vector field of the CG. In sequential form, a pixel can track its links and then check if its neighbor pixel is an attractor. If not, this operation would be repeated for the next neighbor and so on, until an attractor is reached. Each pixel can store a label of the attractor to which it points. Finally, we have a label matrix representing the CG. The matrix of Fig. 7 (right) is ann example of this representation for the simple image in Fig. 7 (left). Using a column convention, the dummy BG attractor has the label 7, and the FG attractor, representative of the FG CC (see Fig. 7, center) is numbered with label 38. Meanwhile, there is another false BG attractor (label 18) and another false FG attractor (label 23), which are underlined in Table 4. These false attractors must be coupled (step 4 of Fig. 6) for the final CCLT, so that the underlined label 23 would be substituted by 38, and the underlined label 18 for 7. The next step consists of transports, or equivalently the fusion of those parts of a same CC, performed by a CG combinatorial optimization process in order to get a tree that has as many nodes as 4-connected components the image has.



7	7	7	7	18	18	7
7	23	23	23	18	18	7
7	23	18	18	18	38	7
7	23	18	18	18	38	7
7	23	38	38	38	38	7
7	7	7	7	7	7	7
7	7	7	7	7	7	7

Fig. 7. A transport that transforms an initial CG into the corresponding simplified CCLT (Left). Links enclosed by a rectangle are to be transported to the thicker links in the CCLT (Center). Label representation, containing for each pixel a label to an FG/BG attractor in the initial CG (Right). (Color figure online)

Graphically, a simple conversion of an initial CG (left) into the CCLT (center) is shown in Fig. 7. Note that the CG of Fig. 7 (left) has one cycle (see the red and green edges surrounding the word "cycle"). Only one cancellation of a pair of false attractors is needed to get the CCLT. CG links that are enclosed by a dotted rectangle are transported to the thicker links for the CCLT. FG attractors are depicted with little triangles, whereas the BG attractors with downward arrows. It can be easily shown that selecting opposite directions for the BG pixel and for the FG pixels when building the CG implies that every BG attractor breaks an FG CC, and vice versa. Thus, by canceling FG-BG attractor pairs until only one attractor would remain for each FG and BG CC, so the transformation from CG into the CCLT is accomplished. After this process there must be only one attractor for each FG (resp. BG) CC. It is worth to note that the process of link transporting is done exclusively handling the CG. The links in the CG enclosed by a dotted rectangle in Fig. 7 (left) are transported, in such a way that both false FG and BG attractors disappear. This is depicted with thicker links in the CCLT (Fig. 7, center). Any transport implies the re-labeling of the label representation (like the matrix of Fig. 7, right). Note that finally the remaining BG attractor is located on the SW corner of the image. The couples to be cancelled can be found by following a path along the CG and by transporting its corresponding links. Yet more, it can be shown that most of these cancellations can be done in parallel, as demonstrated below.

Finally these attractors will define the AdjT in a straightforward form without any geometric computation: Simply, each FG attractor is connected through the CCLT to another BG attractor. And vice versa: each BG attractor is linked to another FG attractor (Fig. 3). So the question is now: what parts of a CCLT building can be done in parallel for the huge amount of pixels that a digital image can have? Whereas first two steps of Fig. 6 are independent for every pixel (thus trivially parallel), the crucial step Fig. 6 requires in principle a sequential processing. Nevertheless, most of the attractors can be coupled in parallel if next properties are taken into account. Let us consider the FG attractor A_{FG} in Fig. 6 (right). The adjacent East BG pixel of A_{FG} fell (going to South direction along the BG path defined by the CG) to a BG attractor B_{FG}. Likewise, the adjacent West FG pixel of B_{FG} arrives (going to North direction along the FG path of the CG) to an FG attractor CFG. All the pairs of FB and BG attractors that fulfill A_{FG} = CFG can be cancelled in parallel, because (a) there are BG and FG paths that connect them, and (b) any tree structure has a unique root. Due to (a) the link of the false FG attractor can be transported so as to join the two FG pixels that the BG attractor was separating. Likewise for the link of the false BG attractor. These are the transports from Fig. 7, left to center. Moreover because of (b), there cannot exist two false FG attractors that use the same false BG attractor to be cancelled.

The parallel Algorithm 1 consists of the following steps. From image I, the possible attractors based on the local information of each pixel can be fully determined in parallel, and the same for the initial CG, for example using the North-then-East criterion (steps 1–4 of Algorithm 1). Using this local information and through successive iterations, the global CG can be obtained, which corresponds to the steps 5 to 9 of Algorithm 1). Now the possible FG, BG attractors can be efficiently coupled in parallel (steps 10–21 of Algorithm 1). At the end of this stage, we obtain the CCLT comprised in the final pointer matrix P of Algorithm 1, and in the true attractors, one for each FG and BG CC. Note that BG attractors now are the holes of the FG CCs (and vice versa). Finally, by means of steps 22–23, inclusion relations between BG and FG CCs can be extracted from the label pointed by each attractor.

As our aim is to describe the inherent parallelism that can be exploited in the CCL tree building, the notation followed here describes the algorithm in an OCTAVE/MATLAB-like form, which indicates in an direct way what are the data parallelism and the real data dependences. Therefore it is evident how each sentence could be implemented in a SIMD processor (or in SIMD kernels) or in SIMT oriented GPUs. Also OpenMP codes can be written almost directly through this notation, just by transforming each matrix operation into two nested loops, the outer of which can be commanded by a directive #pragma omp parallel for. An additional advantage is that the memory access patterns can be clearly observed with this notation. This can give a fast idea of the computing times because memory access is currently the most important bottleneck in current multicore processors [14,21]. For similar reasons, those sentences that can be executed in parallel (which have no real data dependences) are grouped in the same step. For example, the range of elements that can be processed in parallel is shown for each matrix (vector) with the notation $A(1:m, 1:n)$. This means that the operation is extended all over the elements in rows $1, 2, 3, \ldots, m$ and in columns $1, 2, 3, \ldots, n$. Furthermore, we have avoided those matrix operations that cannot be done in an element-by-element way (like matrix inversions, matrix multiplication, etc.). Nevertheless, matrix operations that can be executed in a fully parallel form are introduced with the OCTAVE/MATLAB notation (e.g. $A. * B$ means an element-by-element multiply). Therefore, only one loop "for" and another one loop "while" that present dependences among its iterations are encountered. In addition, the algorithm does not have any conditional sentence. Some auxiliary matrices and predicative-like code, have transformed conditional operations into element-by-element logical ANDs or multiply operations for the possible results. This also promotes efficiency when using SIMD kernel codification and prevents the so-called thread divergence for GPUs, promoting a better performance on these platforms [14]. Supposing an image of m $\times$ n pixels and p processing elements, time complexity order can be trivially obtained according to the notation of Algorithm 1, and because most of the operations are done in an element-by-element form. Steps 1–4 are of this kind, so their time complexity is $O(m \times n/p)$. Steps 6–8 proceed in the same manner, but they are surrounded by a "for" loop (steps 5 and 9) with $log2(m + n - 1)$ iterations. Thus, their complexity is $O((m \times n/p) \times log(m + n))$. Next step 10 can be computed fully in parallel, which supposes a complexity of $O(m \times n/p)$. Step 15 is similar to 10, but because many false attractors were previously deleted from matrixes $F*$ and $B*$ in 10, each "surviving" possible attractor has to be found. This supposes a searching of a variable length s10 that depends on the characteristic of the image. Steps 17 and 19 have the same complexity of 10 with different lengths s17 and s19, respectively. Moreover, steps 17 and 19 are enclosed by a while loop (16 and 21), with a number of iterations q, that is in general very little. These searching lengths s10, s17 and s19, and the number of iterations q can be related with image shapes; but for the random images of different densities [22], maximum values $smax = max\{s10, max, s17, max, s19, max, s22, max\}$ were very low: 72 for 512×512 pixels, 94 for 1 Mpixels and 176 for 4 Mpixels. Besides, q reached 3 only for one of the random images, whereas was 1 for all the tested real ones. In other words, the most time consuming steps are 5 to 9. Summarizing previous steps 1–21, it can be stated that, under the assumption that a processing element exists for each pixel ($p = m \times n$), time complexity order for computing the CCL is very near to the logarithm of the width plus the height of the image. Step 22 involves only the true FG and BG CC (namely v), and can be done in parallel for any CC because matrix P comprises all the connectivity information. The only iterative procedure here is again the number of hops needed to find a true attractor. Thus, the time complexity order for this step is $O(s22 \times v/p + s22, max)$, where $s22, max$ is the maximum number of hops along the pointer matrix P to find an attractor. Step 23 can be done in parallel, being $O(v/p)$ its time complexity. To sum up, time complexity order for computing the CCL and the AdjT, under the assumption that a processing element exists for each pixel, is $O(log(m+n)) + O(q \times smax)$.

5 Testing Results and Conclusions

A complete implementation was done in C++/OpenMP through a direct translation of Algorithm 1. The compiler was Microsoft Visual Express. The server where tests were carried out was an Intel Xeon E5 2650 v2 with: 2.6 GHz, 8 cores, 8×32 KB data caches, Level 2 cache size 8×256 KB, Level 3 cache size 20 MB, maximum RAM bandwidth: 59.7 GB/s. Experiments were run 25 times and minimum times were collected, because this server runs concurrently lots of processes and this increases unfairly timing measurements. However, mean times differs only by a 10% wrt. to the minimum ones. Figure 8 shows the results for random images (taken from YACCLLAB [22]) with different sizes and densities (percentage of FG pixels). Although absolute computation times (being between 0.3 and 0.5 s for 512×512 images of different densities) are not faster than that of YACCLAB, this comparison is not fair since our method computes both black and white CCs, thus having a complete representation in terms of the AdjT, whereas classical CCL methods return only black CC labels. Nevertheless, speedup (time for various threads divided by time for 1 thread) is near the number of threads (Fig. 8), which points out that achieved scalability is very satisfactory for all image sizes and densities.

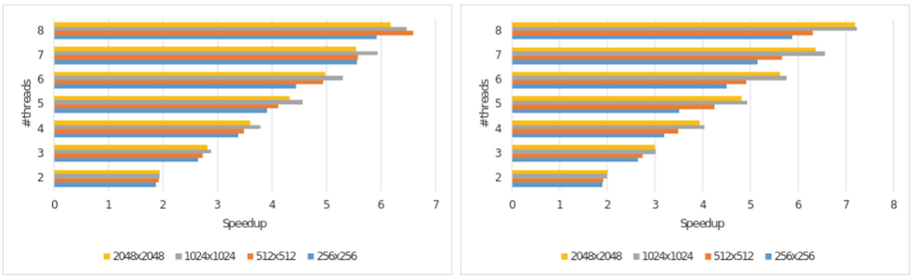


Fig. 8. Left: speedup for 1 to 8 threads for images of different sizes and: Right: density of 0.9. Left: density of 0.4.

In future works we will define more formally the notions of our algorithm so that additional properties will be exhibited. This would also serve to extend our method to bigger dimensions.

Algorithm 1. Given a binary matrix I, computes $P(CCL), B_{TABLE}, F_{TABLE}$ (column 1: index of the BG/FG attractor resp.; column 2: index of the FG/BG attractor, resp.), and AdjT. B means BG and F FG value

```
1: $I_{CC} \leftarrow I(2:m-1, 2:n-1)$; % Central matrix
   $I_{NC} \leftarrow I(1:m-2, 2:n-1)$; % North adjacent matrix. Similar for other 4-adjacent matrices $I_{EC}, I_{SC}, I_{WC}$ (East, South, West) and 8-adjacent matrices $I_{SW}, I_{SE}, I_{NW}, I_{NE}$
2: $F^* \leftarrow (I_{NC} == BG).*(I_{EC} == BG).*(I_{CC} == FG)$;
   $B^* \leftarrow (I_{NC} == FG).*(I_{NE} == BG).*(I_{CC} == FG).*(I_{EC} == FG)$;
   % See 2x2 patterns in Figure 4 (right)
3: $R, C \leftarrow ndgrid(1:m, 1:n)$; % auxiliary matrices that contain a grid of row and column indexes.
4: $X_{NE} \leftarrow (I_{NC} == BG).*(I_{EC} == FG).*(I_{CC} == FG)$;
   $Y_{NE} \leftarrow (I_{NC} == FG).*(I_{CC} == FG)$ % initial X,Y directions
   $P \leftarrow F^*.*((C-1)*m+R)$; % initial local CG as an NE pointer matrix. Only attractors are set with column indexes.
5: **for** $k = 1 : log2(m+n-1)$ **do**
6:     $R_{hop,NE} \leftarrow R - Y_{NE}$;
       $C_{hop,NE} C + X_{NE}$; % row, column indexes are "moved" to the North or East according to $X_{NE}, Y_{NE}$ values.
7:     $L_{hop,NE} \leftarrow sub2ind(Y_{NE}, R_{hop,NE}, C_{hop,NE})$; % R, C matrices are converted into column indexes.
8:     $P \leftarrow P(L_{hop,NE})$; % pointer matrix is updated.
       $X_{NE} \leftarrow X_{NE} + X_{NE}(L_{hop,NE})$;
       $Y_{NE} \leftarrow Y_{NE} + Y_{NE}(L_{hop,NE})$; % XNE, YNE are updated
9: **end for**% After this loop, P contains the global CG
10: %First coupling using East BG pixel to each FG attractor.
    $A_{FG} \leftarrow P(F^*(2:m-1, 2:n-1))$; % Vector of FG attractors
    $A_{FG,East} \leftarrow P(F^*(2:m-1, 3:n))$;% East BG pixels to FG attractors
11: $A_{BG} \leftarrow P(A_{FG,East})$; % Vector of BG attractors
12: $C_{FG} \leftarrow P(A_{BG} + m)$; %FG attractors from West FG ABG pixels
13: $A_{cancel} = (A_{FG} == C_{FG})$; % Logical Vector of FG/BG attractors that must be cancelled in parallel.
14: $N_{cancel} = count(A_{cancel})$;% # FG/BG attractors cancelled.
    $B^*(P(A_{BG}(A_{cancel}))) \leftarrow 0$; % BG attractors are deleted from logical matrix of BG attractors and from auxiliary matrices.
    $F^*(P(A_{FG}(A_{cancel}))) \leftarrow 0$; % The same for FG attractors. % Here, labels in matrix P are also updated according to the link transport of section 4.
15: % Step 10 is repeated for Vectors of North BG pixels to the FG attractors and FG attractors from the South FG pixels to ABG. Each addressing along P must be iterated until an FG or BG attractor is found (because, in previous steps, many false attractors were deleted from $F^*$ and $B^*$).
16: **while** $N_{cancel} > 0$ **do**
17:     % Step 10 is repeated. Each addressing along matrix P must be iterated until an FG or BG attractor is found.
18:     $N_{cancel} = count(A_{cancel})$
19:     % Step 15 is repeated. Again addressing along matrix P must be iterated until an FG or BG attractor is found.
20:     $N_{cancel} = count(A_{cancel}) + N_{cancel}$; % total number of FG/BG attractors cancelled in current "while" iteration.
21: **end while**
22: % Extract attractor pair tables $F_{TABLE}, B_{TABLE}$ from previous attractors using P.
23: % Compute AdjT using attractor pair tables $F_{TABLE}, B_{TABLE}$
```

References

1. Bhattacharya, P.: Connected component labeling for binary images on a reconfigurable mesh architecture. J. Syst. Arch. **42**(4), 309–313 (1996)
2. Buneman, O.P.: A grammar for the topological analysis of plane figures. Mach. Intell. **15**, 383–393 (1969)
3. Chiavetta, F., Di Gesù, V.: Parallel computation of the Euler number via connectivity graph. Pattern Recognit. Lett. **14**, 849–859 (1993)
4. Diaz-del-Rio, F., Real, P., Onchis, D.: A parallel homological spanning forest framework for 2D topological image analysis. Pattern Recognit. Lett. **83**, 49–58 (2016)
5. Cohn, A., Bennett, B., Gooday, J., Gotts, N.: Qualitative spacial representation and reasoning with the region connection calculus. GeoInformatica **1**(3), 275–316 (1997)
6. Costanza, E., Robinson, J.: A region adjacency tree approach to the detection and design of fiducials. Video Vis. Graph., 63–99 (2003)
7. Cucchiara, R., Grana, C., Prati, A., Seidenari, S., Pellacani, G.: Building the topological tree by recursive FCM color clustering. In: 16th IEEE ICPR, vol. 1, pp. 759–762 (2002)
8. Gupta, S., Palsetia, D., Patwary, M.M.A., Agrawal, A., Choudhary, A.N.: A new parallel algorithm for two-pass connected component labeling. In: IEEE IPDP Symposium, pp. 1355–1362 (2014)
9. Heijmans, H.J.: Connected morphological operators for binary images. Comput. Vis. Imag. Understand. **73**(1), 99–120 (1999)
10. Kalentev, O., Rai, A., Kemnitz, S., Schneider, R.: Connected component labeling on a 2D grid using CUDA. J. Parallel Distrib. Comput. **71**, 615–620 (2011)
11. Kovalevsky, V.: Algorithms in digital geometry based on cellular topology. In: Klette, R., Žunić, J. (eds.) IWCIA 2004. LNCS, vol. 3322, pp. 366–393. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30503-3_27
12. Keshet, R.: Shape-tree semilattice. J. Math. Imag. Vis. **22**(2–3), 309–331 (2005)
13. Murty, A., Natarajan, V., Vadhiyar, S.: Efficient homology computations on multicore and manycore systems. In: 20th Annual International Conference on High Performance Computing, pp. 333–342 (2013)
14. Oxley, J.G.: Matroid Theory, vol. 3. Oxford University Press, Oxford (2017). NVIDIA, Cuda C best practices guide version. http://developer.nvidia.com/
15. Patwary, M., Ali, M., Refsnes, P., Manne, F.: Multi-core spanning forest algorithms using the disjoint-set data structure. In: 26th IEEE IPDP Symposium, pp. 827–835 (2012)
16. Institute of Computer Science, Jagiellonian University (2017). REDHOM, Redhom. http://redhom.ii.uj.edu.pl/
17. Ranwez, V., Soille, P.: Order independent homotopic thinning for binary and grey tone anchored skeletons. Pattern Recognit. Lett. **23**(6), 687–702 (2002)
18. Rosenfeld, A.: Adjacency in digital pictures. Inf. Control **26**, 24–33 (1974)
19. Serra, J.: Image Analysis and Mathematical Morphology. Academic Press, Cambridge (1982)
20. Stell, J., Worboys, M.: Relations between adjacency trees. Theor. Comput. Sci. **412**(34), 4452–4468 (2011)
21. Williams, S., Waterman, A., Patterson, D.A.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM **52**, 65–76 (2009)
22. YACCLAB - Yet Another Connected Components Labeling Benchmark (2017). https://github.com/prittt/YACCLAB