HAL
open science

Parallel Computation of Watershed Transform in Weighted Graphs on Shared Memory Machines

Yosra Braham, Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui

To cite this version:

Yosra Braham, Yaroub Elloumi, Mohamed Akil, Mohamed Hedi Bedoui. Parallel Computation of Watershed Transform in Weighted Graphs on Shared Memory Machines. Journal of Real-Time Image Processing, In press, 10.1007/s11554-018-0804-x . hal-02121832

HAL Id: hal-02121832

https://hal.science/hal-02121832

Submitted on 7 May 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Noname manuscript No.
(will be inserted by the editor)

Parallel Computation of Watershed Transform in Weighted Graphs on Shared Memory Machines

Yosra Braham · Yaroub Elloumi · Mohamed Akil · Mohamed Hedi Bedoui

Received: 13 December 2017 / Accepted: date

Abstract Watershed Transform is a widely used image segmentation technique that is known to be very data-intensive and time-consuming. The M-border Kernel Algorithm computes watersheds in the framework of Edge Weighted Graphs and allows to preserve the topology of the initial map. Parallelization represents an effective solution to accelerate it. However, this task remains challenging due to the nature of this technique. In this paper, we address this problem. We start by analyzing the Data Dependency Issues that this algorithm raises when dealing with parallel execution. With respect to that, we propose a parallelization strategy that opts for vertices scanning instead of edges scanning of the graph while preserving the thinning paradigm on which the M-border Kernel Algorithm is based. We show that this strategy overcomes the problem of the simultaneous lowering of two adjacent M-border edges that may occur when edges scan is used. The implementation of the proposed algorithm on a shared memory multicore architecture proves its effectiveness in terms of speedup. In fact, the experimental results show that a

Y. Braham and Y. Elloumi
ESIEE Paris, LIGM, A3SI
BP 99, 2 Bd Blaise Pascal, 93162 Noisy-Le-Grand, France
and
Medical Technology and Image Processing Laboratory, Faculty of Medicine, University of Monastir
Avenue Avicenne, Monastir, 5019, Tunisia
E-mail: yousra.braham@esiee.fr

M. Akil
ESIEE Paris, LIGM, A3SI
BP 99, 2 Bd Blaise Pascal, 93162 Noisy-Le-Grand, France
E-mail: mohamed.akil@esiee.fr

M. H. Bedoui
Medical Technology and Image Processing Laboratory, Faculty of Medicine, University of Monastir
Avenue Avicenne, Monastir, 5019, Tunisia
E-mail: medhedi.bedoui@fmm.rnu.tn

speedup factor of 5.55 is achieved using 8 processors for 2048×2048 images over the performance of the sequential algorithm using a single processor on the same architecture. Furthermore, the gain in terms of execution time and thus speedup is guaranteed whatever is the size of images on which the algorithm is applied. In fact, a speedup factor of 5.55 is obtained for 2048×2048 images, 5.11 for 1024×1024 images and 4.45 for 512×512 images using 8 cores.

Keywords Image segmentation · parallel algorithms · real-time performance · Watershed Cuts · shared memory multicore architecture.

1 Introduction

In the field of image processing, Watershed Transform is considered as one of the most popular methods used for gray scale image segmentation. This transform denotes a family of methods derived from Mathematical Morphology that considers a gray scale image as a topographic relief where the gray level of a pixel represents its altitude. The watershed is defined as the ridges separating the catchment basins in this relief. Several algorithms that implement the watershed in the discrete case were proposed [17, 22, 23]. These algorithms can be figured out into two main categories according to the approach they are based on: Watershed by flooding [2, 18, 24, 25] and Topographical Watershed [19].
According to [1], most of the existing algorithms have a drawback in that they do not preserve the topological properties of the image. In fact, they produce a binary result that gives no information about the contrast between the obtained regions. Furthermore, they produce contours that do not correspond precisely to the significant ones of the original image. In [1, 7, 20], the authors defined a new approach for Watershed Transform computation that allows to preserve the image topology. They proposed a new framework allowing the precise definition of the discrete watershed and leading to what is known as the Topological Watershed.
In this context, the authors in [10, 11] studied the Topological Watersheds in the framework of Edge Weighted Graphs. They introduced the notion of Watershed Cuts and have established its consistency [8, 9] and optimality in the sense of their equivalence to the separations induced by the Minimum Spanning Forest [5, 6, 13] relative to the minima subgraph. Watershed Cuts have important properties detailed in [9–11, 17] making them stand as a fundamental step in a great deal of powerful image segmentation processing. They can be defined equivalently by their attraction basins or by the lines separating these attraction basins.
Two algorithms were proposed to compute Watershed Cuts [10, 11]. The first one relies on the extraction of flows. It explores the steepest descent paths, mixing Depth First and Width First iterations. It produces a flow partition, and therefore a Watershed Cut. The second one is a linear time algorithm that is based on a thinning paradigm called border thinning which lower, until idempotence, the values of edges that satisfy a local property. As a result,

the minima of the transformed map constitute a Minimum Spanning Forest of the original map and consequently induce a Watershed Cut. This algorithm, called the M-border Kernel Algorithm, is among the most efficient Watershed Algorithms [11].

The Watershed Transform computation through its various algorithms remains a relatively consuming task in terms of time and memory use [16]. In fact, it requires several scans of the image, graph building, etc. This leads to the creation and processing of complex data structures. In different application domains, images size is still growing, which leads to a similar rise in the processing time. In particular, the M-border Kernel Algorithm plays a key role in a lot of real-time image processing. The features described previously present reliable constraints to implement it.

To face this issue, parallelization represents both an interesting and challenging track to explore. In this paper we tackle this issue. For this aim, we focus on a particular Watershed Algorithm which is the M-border Kernel Algorithm [11] and we investigate its parallelization in order to achieve real-time processing. Our choice of this algorithm is motivated by its theoretical and implementation properties. From a theoretical standpoint, the M-border Kernel is a linear time algorithm (i.e. its execution time increases linearly when the size of the input Edge Weighted Graph increases [28]) that is based on a thinning paradigm obtained by iteratively lowering down the values of the graph edges that satisfy a local property. Such a locality is very helpful for parallel implementation of this algorithm since the edges verifying a certain condition can be processed independently. However, the locality should not be misinterpreted in a way that massive parallelism can be applied in the parallel implementation. In fact, the Data Dependency of the algorithm should be analyzed. From the implementation standpoint, neither a hierarchical queue nor a sorting step is required.

In this paper we firstly demonstrate that despite the locality of the lowering operation, its parallel execution raises a dependency problem especially at the level of adjacent M-border edges. Secondly, we propose a parallelization strategy of this algorithm that overcomes this dependency issue. This strategy consists in examining non-minima vertices adjacent to minima ones instead of edges of the initial Edge Weighted Graph. Therefore, a set of independent vertices are processed in parallel.

The implementation of the proposed parallel M-border Kernel Algorithm on a shared memory multicore architecture shows a gain in terms of execution time and thus speedup factor according to both number of cores and images sizes. Furthermore, the preliminary steps of this algorithm are parallelized. Thus, parallel Image Segmentation Technique based on Watershed Cut computation using the M-border Kernel Algorithm is implemented on the same architecture and the obtained results are evaluated in terms of number of cores and images sizes.

The remainder of this paper is organized as follows: in the second section, we introduce basic notions and notation used in this paper and required to handle the computation of Watershed Cuts in the framework of Edge Weighted

Graphs. In the third section, we introduce and analyze the sequential M-border Kernel Algorithm. In the fourth section we propose a parallelization strategy of this algorithm after studying its Data Dependency. In the fifth section we present and discuss the experimental results of the proposed parallel M-border Kernel Algorithm. In the sixth section we discuss the results obtained by the implementation of a gray scale Image Segmentation Technique based on Watershed Cut computation using the M-border Kernel Algorithm. Finally, we conclude with a summary.

2 Preliminaries

This section is devoted first to an essential background material for Edge Weighted Graphs. It introduces the basic concepts required to handle the M-border Kernel Algorithm in both sequential and parallel versions.
An Edge Weighted Graph (or Weighted Graph) is defined as a triple $G = (V, E, F)$ where V is a finite set of elements called vertices, E is a subset of pairs u=$\{x, y\}$ of elements of V called edges such that $x \neq y$ and F is a function that weights the edges of G.
A digital gray scale image can be represented by a special type of Weighted Graphs whose drawing in a Euclidian Space forms a regular square tiling. The set of vertices V corresponds to pixels and the set of edges to any adjacency relation. In this paper we consider the 4-adjacency. The edges are in that case horizontal and vertical. Furthermore, we denote by F the map from E to $\mathbb{Z}$ that assigns to each edge u=$\{x, y\}$ the dissimilarity between the gray level of x and y that is called the altitude of u, i.e. $F(u) = \{|I(x) - I(y)|$ such that u=$\{x, y\}\}$. We denote by VF the map from V to $\mathbb{Z}$ such that for any $x \in V$, $VF(x)$ is the minimal altitude of an edge which contains x, i.e. $F(x)$ = min$\{F(u)$ such that $u \in E$, $x \in u$ $\}$ and is called the altitude of x.
A set of edges of G is a (regional minimum) of F (at altitude $k \in \mathbb{Z}$) if k is the altitude of any edge of this set and the altitude of any edge adjacent to this set is strictly greater than k. We denote by E_{min} the union of all minima edges of F. A vertex is considered as a minimum vertex of VF if it belongs to a minimum edge of F. The set of minima vertices of F called V_{min} is composed of the union of all minima vertices of VF. The subgraph of G composed of the vertices set and edges set of all minima of F and VF is denoted by M=(E_{min}, V_{min}).
Figure 1 illustrates these notions. It shows a 5×5 image transformation to a 5×5 Weighted Graph and its corresponding minima subgraph.

3 Linear-time M-border Kernel Algorithm

The M-border Kernel Algorithm proposed by J. Cousty and al in [11] is an efficient linear time algorithm that computes Watershed Cuts in the framework

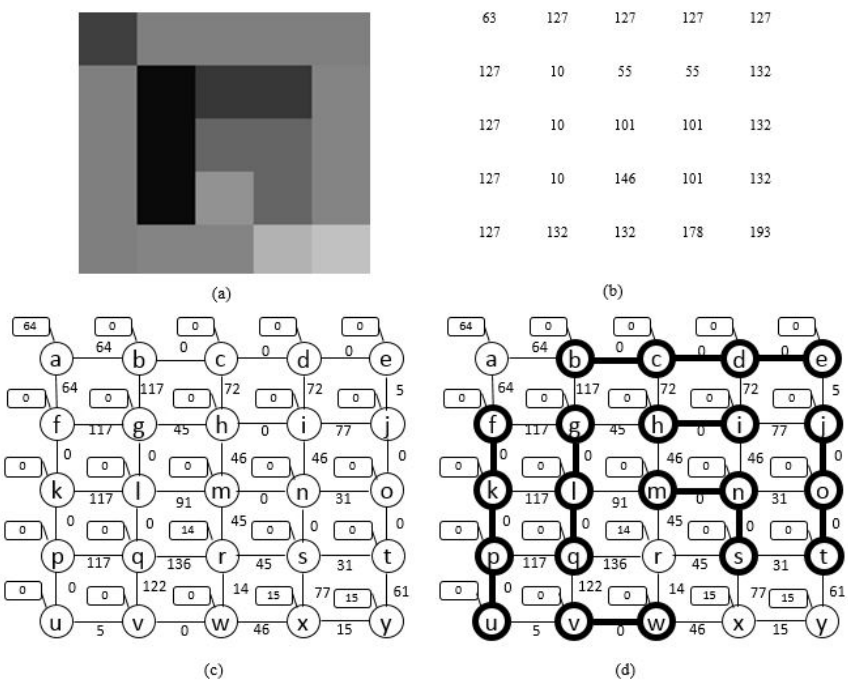

Fig. 1 Image transformation to an Edge Weighted Graph $G(V,E,F)$ and its corresponding minima subgraph M (a) initial image of size 5×5 (b) its corresponding matrix of size 5×5 (c) its corresponding Edge Weighted Graph and (d) its corresponding minima subgraph in bold

of Weighted Graphs. In the following, we describe its principle.
Consider a Weighted Graph $G(V,E,F)$ and its corresponding minima subgraph M=(E_{min},V_{min}), let u=$\{x,y\} \in E$. If only one vertex of $u \in M$ then u is said to be outgoing from M. If $F(u) > max(VF(x),VF(y))$, then u is said to be locally separating. If $F(u) = max(VF(x),VF(y))$ and $F(u) > min(VF(x),VF(y))$, then u is said to be a border edge. Moreover, an edge u is said to be a minimum-border or M-border, if it is a border edge and if exactly one vertex of u is a minimum vertex.
Taking an edge $u \in E$, the lowering of a map F at u is the map F_1 such that $F_1(u) = min_{x \in u}\ \{F(x)\}$ and $F_1(v) = F(v)$ for any edge $v \in E \setminus \{u\}$ [11]. The M-border Kernel Algorithm is based on a thinning transformation that consists in iteratively lowering the values of M-border edges until stabilization. Algorithm 1 describes the different steps of this algorithm. It requires two preliminaries steps: regional minima detection of the initial map and vertices valuation. Initially and as described in the iterative loop at line 3 of Algorithm 1, outgoing edges are stored in a set L. After that, in the main loop (line 5 of Algorithm 1), the set L is browsed and the lowering process is applied on

M-border edges (line 11), then, the sets V_{min} and E_{min} are updated (lines 14 and 15). The update of the set L is done by adding new outgoing edges as described in the loop on line 16. At the end of Algorithm 1, F is an M-border Kernel of the initial map. The minima of the transformed map constitute a Minimum Spanning Forest relative to the minima of the initial one and, hence, induce a Watershed Cut.

Algorithm 1: Sequential M-border Kernel Algorithm [11]

Data: (V, E, F) a Weighted Graph, $M = (V_{min}, E_{min})$ its minima subgraph and VF the vertices valuation function.
Result: F an M-border kernel of the initial map and M its minima subgraph.

```
1  // Initialization
2  L := φ;
3  foreach (u ∈ E outgoing from (V_min, E_min)) do
4      L := L ∪ {u}; // The set L contains outgoing edges from M
5  while (it exists u ∈ L) do
6      L := L \ {u}; // Remove the edge to be processed from the set L
7      if (u is a border edge for F) then
8          x :=vertex of u such that VF(x) < F(u);
9          y :=vertex of u such that VF(y) = F(u);
10         // The lowering process
11         F(u) := VF(x);
12         VF(y) := F(u);
13         // Update of the minima subgraph M
14         V_min := V_min ∪ {y};
15         E_min := E_min ∪ {u};
16         foreach (v={y',y} ∈ E such that y' ∉ V_min) do
17             L := L ∪ {v};// add new outgoing edges from M to the set L
```

4 Parallel M-border Kernel Algorithm

In parallel programming, Data Dependency study is a crucial step that permits to know which data and operations can be performed simultaneously. We begin this section by such a study applied to the M-border Kernel Algorithm. Based on the result of this study we then propose a parallelization strategy for this algorithm.

4.1 Data Dependency Analysis

The M-border Kernel Algorithm is based on an elementary local operation called lowering. This operation does not involve any information from other

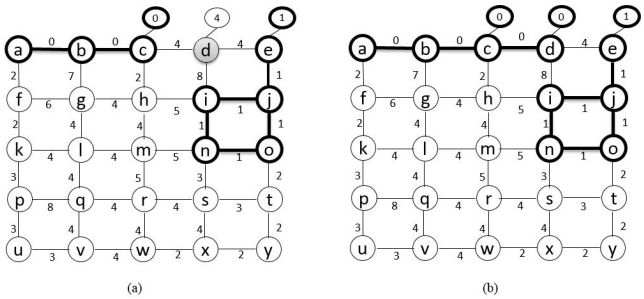

Fig. 2 A graph G and a map F (a) in bold minima of F (b) the edge $\{c,d\}$ is lowered and the edge $\{d,e\}$ is not an M-border edge anymore

edges in order to perform it. Consequently, a set of independent M-border edges can be lowered in parallel. The result will be an M-border thinning and is optimal in the sense of its equivalence to the separations induced by the Minimum Spanning Forest relative to the Minima. This optimality is guaranteed whatever the order in which the values of edges are lowered.
However, a particular edge configuration causes a problem when dealing with simultaneous lowering. This configuration occurs when two or more adjacent M-border edges have a non-minimum common vertex. In this case, only one of the adjacent M-border edges must be lowered. Figure 2 illustrates this configuration. Fig. 2 (a) presents a Weighted Graph G and a map F in which minima of F are depicted in bold. In this figure, edges and vertices values are assigned as indicated in section 2. Edges $\{a,f\}$ and $\{o,t\}$ are two non-adjacent M-border edges so they can be processed in parallel. However, the edges $\{c,d\}$ and $\{d,e\}$ are two adjacent M-border edges. So, they cannot be lowered in parallel. In fact, if the edge $\{c,d\}$ is lowered, the edge $\{d,e\}$ is not an M-border edge anymore as shown in Fig. 2 (b).

4.2 Proposed parallel M-border Kernel Algorithm

To cope with the Data Dependency problem, we propose a parallelization strategy of the M-border Kernel Algorithm that is based on the examination of vertices instead of edges in order to obtain a set of edges that forms the Watershed Cut.
The main idea of the proposed parallel algorithm (Algorithm 2) consists in examining in parallel the non-minima vertices adjacent to the minima ones of the initial map while preserving the thinning paradigm on which the sequential M-border Kernel Algorithm is based.
As noted in the loop on line 6 of Algorithm 2, for each non-minimum vertex that has an adjacent minimum vertex, if the edge connecting these two vertices is an M-border edge, then, the non-minimum vertex and the M-border edge detected are lowered as shown on lines 9-10 and the rest of neighbors are not examined. Furthermore, an update of the minimum subgraph is performed as

noted on lines 12 and 13. Finally, the set of non-minima vertices is updated by deleting the lowered vertex as noted on line 14.

Algorithm 2: Parallel M-border Kernel Algorithm

Data: a Weighted Graph (V, E, F) , $M = (V_{min}, E_{min})$ its minima subgraph and VF the vertices valuation function;
Data: P number of processors;
Result: F M-border Kernel of the initial map F and M its minima

```
1  // Initialization
2  V_Nmin= V \ {V_min} ; // The set V_Nmin contains non-minima vertices
3  while ( V_Nmin ≠ φ) do
4      (S_1, ... ,S_P)=Dynamic_Partition(V_Nmin,P);
5      foreach (p ∈ {1, ... ,P } ) do in parallel
6          foreach ( x ∈ S_p and x ∈ V_Nmin and x adjacent to y ∈ V_min )
           do
7              if ( u={x,y} is an M-border edge ) then
8                  // Lowering process
9                  F[u]:=F[y];
10                 F[x]:=F[u];
11                 // Update of the minima subgraph M
12                 E_min:= E_min ∪ {u} ;
13                 V_min:= V_min ∪ {x} ;
14                 V_Nmin:= V_Nmin \ {x} ; // Remove the vertex processed
                     from the non-minima vertices set
```

The loop on line 6 of Algorithm 2 consists in examining all vertices that belong to $V \setminus V_{min}$ set. This loop is scheduled on P processors. The latest processing is performed for all non-minima vertices (V_{Nmin}) as indicated in the loop on line 3.

When applying Algorithm 2 on a Weighted Graph, we distinguish several cases: in the first case, a non-minimum vertex has only one neighbor vertex that is labeled as minimum vertex. In this case, if the edge linking these two vertices is an M-border edge, both the edge and the non-minimum vertex are lowered otherwise the lowering process is not applied and the edge is a separating edge. In the second case, a non-minimum vertex has more than one neighbor vertex that is labeled as a minimum vertex and the edge linking them is an M-border edge. In this case, only one adjacent vertex and an M-border edge are lowered. Figure 3 details these cases: Fig. 3 (a) represents a Weighted Graph where edges and vertices altitudes are assigned as indicated in section 2. In this figure the minima subgraph is depicted in bold. Figures 3 (b) and 3 (c) illustrate the first case. The vertex m is adjacent to an unique minimum vertex which is the vertex n, but the edge $\{m, n\}$ is not an M-border edge. Thus, neither the edge $\{m, n\}$ nor the vertex m is lowered as shown in Fig. 3 (b). However, in Fig. 3 (c) the non-minima vertex f is adjacent to a unique minimum vertex which is the vertex a. Since the edge linking these two vertices is an M-border

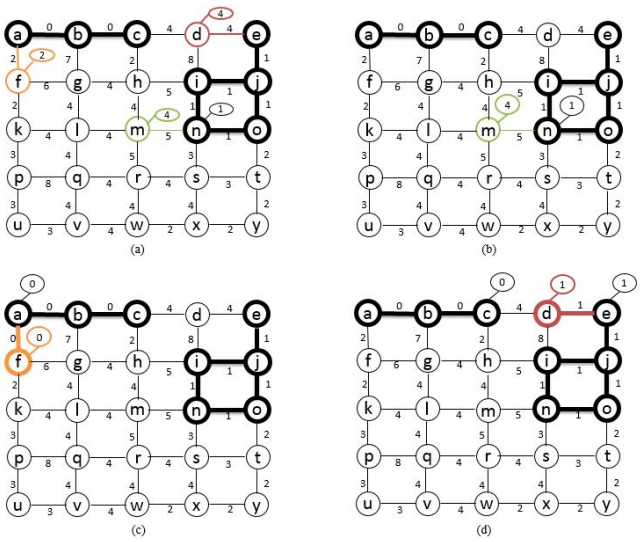

Fig. 3 An Edge Weighted Graph and its minima (in bold) (a) case of a non-minimum vertex adjacent to an unique minimum vertex (but the edge linking the two vertices is not an M-border edge) (b) case of a non-minimum vertex adjacent to an unique minimum vertex and lowering the M-border edge linking them (c) case of a non-minimum vertex adjacent to more than one minimum vertex (d)

edge, both the edge $\{f,a\}$ and the vertex f are lowered as shown in Fig.3 (c). The second case is illustrated in Fig. 3 (d). The non-minimum vertex d is adjacent to three minima vertices which are e, i and c. Hence, only the edge $\{d,e\}$ which is an M-border edge is lowered together with vertex d.

Taking the Weighted Graph of Fig. 4 (a) and its corresponding minima subgraph $M = (V_{min}, E_{min})$ which is depicted in bold as an example to be treated using P=3 processors, only five iterations are required to obtain the final result as shown in Fig. 4. At each iteration, the partition of non-minimum vertices on P processors is performed according to a Dynamic Partition Algorithm: Algorithm 3 detailed in the next section.

4.3 Dynamic Partition Algorithm

In our proposed parallel M-border Kernel Algorithm, at each iteration, the vertices are distributed on P processors using a Dynamic Partition Algorithm. This algorithm consists in computing, at each iteration, partitions $(S_1, \ldots, S_P)$ of a set S containing vertices to process by P processors. Each partition is treated by a processor. As noted in the loop on line 4 of Algorithm 3, for each processor p such as $p = \{1, \ldots, P\}$ the starting and ending indexes of data to process are computed and copied in a constant time.

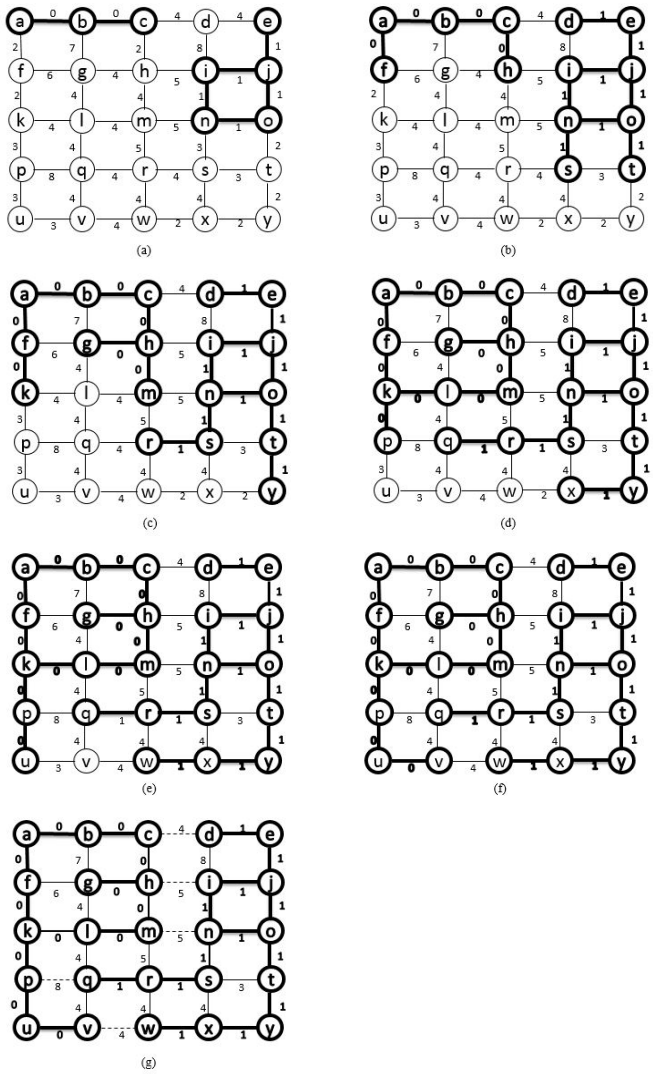

Fig. 4 $G(V, E, F)$ a Weighted Graph and M its corresponding minima depicted in bold (a) Application of Algorithm 2 on G using P=3 processors (b), (c), (d), (e) and (f) and final result with the Watershed Cuts depicted in dashed lines (g)

Algorithm 3: Dynamic_Partition

Data: S the set of vertices to be treated in parallel ;
Data: P the number of processors.
Result: $(S_1, \ldots, S_P)$ the sets of vertices to be treated by each processor.

```
// Initialization
Q= |S|/P;
R= |S| mod P; // R is the rest of the division of |S| by P
foreach (Processor p ∈ {1, ...,P}) do in parallel
    if (p ≤ R) then
        start[p]=(p-1)*(Q+1);
        end[p]:=start[p] + Q;
    else
        start[p]=(p-1)*Q+R;
        end[p]:=start[p]+ Q - 1;
    S_p:=S[start[p], ...,end[p]];
```

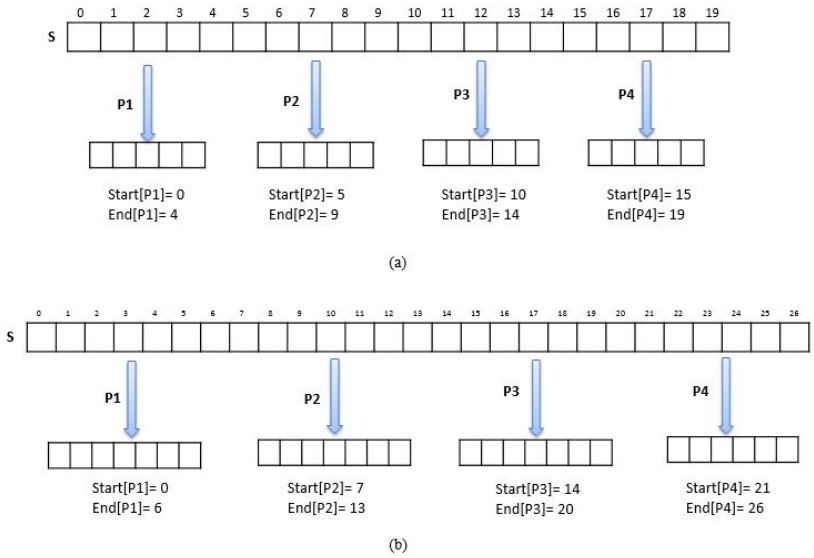

Fig. 5 Dynamic Partition (Algorithm 3): case of R=0 (a) and case of $R \neq 0$ (b)

Let $Q \in \mathbb{N}$ be the quotient of the division of $|S|$ by the number of processors P and let $R \in \mathbb{N}$ be the rest of this division. Algorithm 3 generates partitions of size Q or $Q+1$. In fact, depending on the value of R, two cases are distinguished: in the first case R=0, then, the obtained partitions are balanced and are of size Q, otherwise, the first R partitions are of size $Q+1$ and the $(P-R)$ remaining partitions are of size Q. These two possible cases are illustrated in Fig. 5.

5 Parallel implementation and experimental results

In this section, we describe first the platform of test on which the M-border Kernel Algorithm is implemented and the performance measures used to evaluate its parallelization. We describe then the image dataset used for the experimental assessments. Finally, we present and discuss the obtained results.

5.1 Target platform and performance evaluation

Multicore architectures have become a popular way to implement dynamic, highly asynchronous, concurrent programs. There are mainly two models of parallel programming paradigms for multicores CPUs environments: shared memory and distributed memory [14,15]. In the distributed memory model, each processor has its own local memory and its content is not replicated anywhere else. However, for the shared memory model, there is one common shared memory for all processors. For the first model, the communication of

Table 1 Description of the image dataset that sequential and parallel algorithms are applied on

Number of images	10	10	10	10	10	10
Resolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Number of Vertices in the correesponding Graph	4096	16384	65536	262144	1048576	4194304
Number of Edges in the correesponding Graph	8192	32768	131072	524288	2097152	8388608
The range of the number of minima	12-487	13-4065	17-16220	67146-36880	253886-165548	408764-879834

data occurs through discrete messages sent from a processor to another. In this case, the Message Passing Interface (MPI) is the standard language for parallel programs. For the second model, thread communication is efficient because they operate in the same address space. The main advantage of Message Passing model is scalability while the shared memory is faster. With respect to the context of our work we opted for the shared memory model for the implementation of the proposed parallel algorithms. Therefore, experiments are carried out on a computer with 64 GB of RAM and an Intel Xeon processor E5-2640 v3 (8 cores, a basic frequency of 2.6 GHz and a smart cache of 20 MB) using OpenMP as an API extension to the C/C++ language for the support of the shared memory model.

The performance of the proposed parallelization strategy is evaluated through the execution time and thus, the speedup factor. Speedup is used to measure the gain of performance of parallel algorithms compared to sequential ones which deal with the same problem. It represents the gain in execution time of a task implemented using P processors compared to its implementation using a single processor on the same architecture.

5.2 Assessment image dataset

The image dataset used for the experimental evaluations is detailed in Table 1. It is composed of 60 standard and synthetic gray scale images of different sizes (64×64, 128×128, 256×256, 512×512, 1024×1024 and 2048×2048). The standard images used are selected among those which are commonly used in image processing benchmarking (such as Lena, cameraman, peppers...). These images contain different types of scenes.

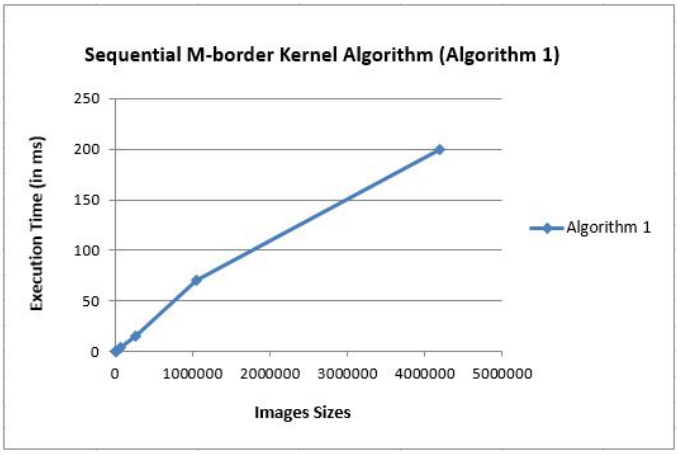

Fig. 6 Execution time of Algorithm 1 according to the images sizes

5.3 Experimental results

In this section, the results obtained by implementing both sequential and parallel M-border Kernel Algorithms are presented and discussed. The evaluation of the proposed parallel algorithm is discussed in terms of number of cores and images sizes.

Experimental results of the sequential M-border Kernel Algorithm Taking into account the dataset described in Table 1, Fig. 6 represents the execution time of Algorithm 1 with respect to the images sizes. In this figure, for each size, the associated execution time corresponds to the average of the execution times of 10 images. It shows that the execution time increases with respect to the segmented images sizes and thus, to the number of edges of their associated Weighted Graph.

Experimental results of the parallel M-border Kernel Algorithm Figure 7 illustrates some results obtained by applying the parallel M-border Kernel Algorithm taking into account the two preliminary steps on images of different sizes. The obtained contours by Algorithm 2 are superimposed on the initial images.

In image segmentation, one of the evaluation criteria of parallel implementation of algorithms is the ability of the parallel version to produce a result as similar as possible to the one produced by the sequential version. Such an ability is evaluated using similarity metrics such as Dice [26] and Jaccard [27] coefficients. Table 2 presents the values obtained for these two coefficients when comparing the sequential and the parallel M-border Kernel segmentation results. Each value corresponds to the average of values of 10 images having the same size. This table shows that the parallel algorithm produced results

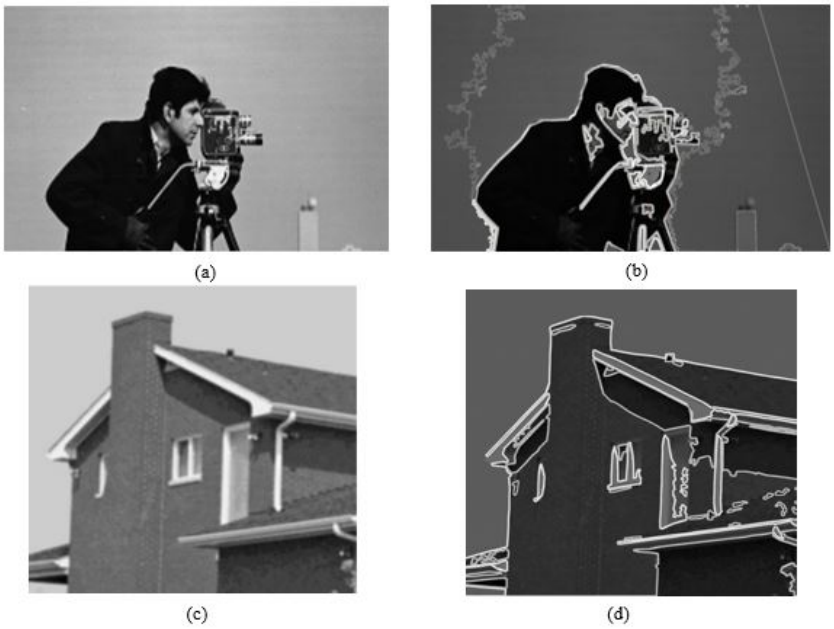

Fig. 7 (a) and (c) initial images and (b) and (d) their corresponding segmented images obtained by applying the parallel M-border Kernel Algorithm using 4 cores

Table 2 Quantitative assessment between parallel and sequential algorithms

Number of images	10	10	10	10	10	10
Resolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Dice index	0,963	0,959	0,949	0,950	0,963	0,969
Jaccard index	0,981	0,979	0,968	0,974	0,975	0,978

that are similar to the sequential ones at a satisfactory rate of about 95 to 98%.

In order to study the performance of the parallel algorithm according to the number of processors, we proceed to increasing iteratively the number of cores from 1 to 8 for an image of size 2048×2048. Figure 8 presents the execution time and the speedup of the proposed parallel algorithm. This figure shows that this algorithm allows a gain in terms of execution time. It implies a speedup of 5.99 using 8 cores.

In order to evaluate the impact of the images resolutions on our parallel algorithm, we apply Algorithm 2 on each image of the dataset described in Table 1 while increasing the number of cores.

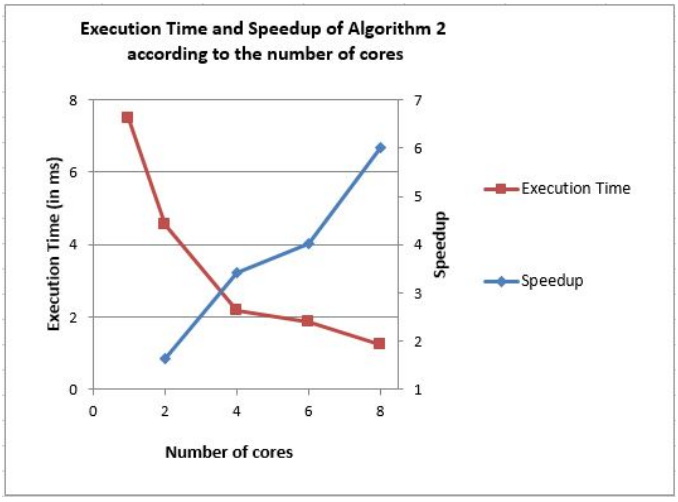

Fig. 8 Execution time and speedup of the parallel M-border Kernel Algorithm (Algorithm 2) according to the number of cores for a 2048×2048 image

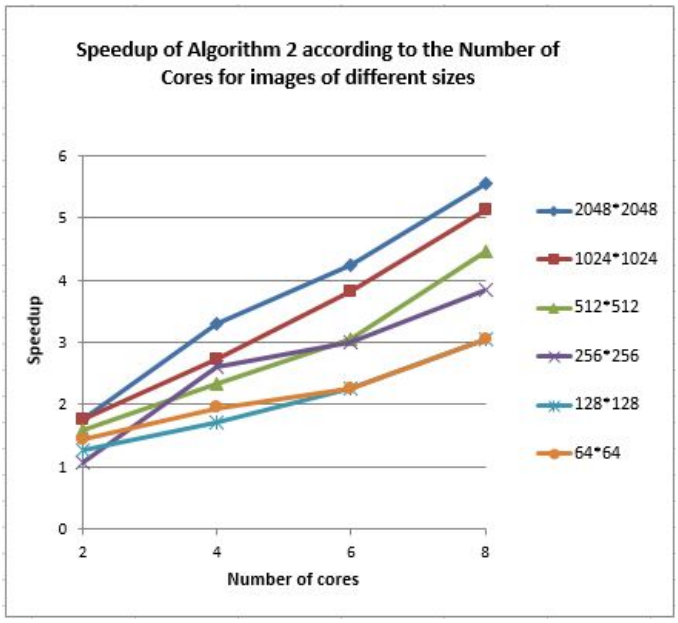

Fig. 9 Speedup of the proposed parallel M-border Kernel Algorithm applied on images of different sizes according to the number of cores

Table 3 Speedup of Algorithm 2 applied on images of different sizes using 8 cores

Number of images	10	10	10	10	10	10
Resolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Speedup	3.03	3.05	3.83	4.45	5.11	5.55

Figure 9 illustrates the obtained results. Each curve in Fig. 9 represents the evolution of the speedup according to the number of cores for a given image size. The obtained curves show that our proposal enables a gain in terms of speedup whatever the size of images is. Table 3 presents the speedup factor of algorithm 2 for images of different sizes using 8 cores. Note that for each size the corresponding speedup factor is the average speedup of 10 images of this size.

Algorithm 2 is based on a dynamic partitioning. The partition of vertices to be processed in parallel by P processors is performed at each iteration using Algorithm 3. This algorithm produces balanced partitions. Thus, all processors are assigned with the same workload. Consequently, idle time is minimized and parallelism performance is better preserved.

6 Application: Image Segmentation based on Watershed Cut computation using the parallel M-border Kernel Algorithm

The Image Segmentation Technique based on Watershed Cut computation using the parallel M-border Kernel Algorithm is composed of three main steps: the regional minima detection of the initial map, the vertices valuation of the Weighted Graph and finally the M-border Kernel computation. The two first steps are preliminary steps and are common to some other Watershed Algorithms. The third step constitutes the main core of the technique that we are dealing with in this paper. To the best of our knowledge, this algorithm has not been parallelized in the literature. That is why we focused on its parallelization in the previous section.
Parallelizing the Image Segmentation Technique based on the M-border Kernel Algorithm consists in parallelizing its three steps. Two parallelization strategies can be considered in this context: either a global parallelization of the three steps or an individual parallelization of each step. Since each step needs the global result of the previous one, we opted for the second parallelization strategy that is illustrated in Fig. 10.

6.1 Parallel Regional Minima Detection Algorithm

The sequential Regional Minima Detection Algorithm [12] generates the set of all regional minima edges of the initial map F. It consists in examining

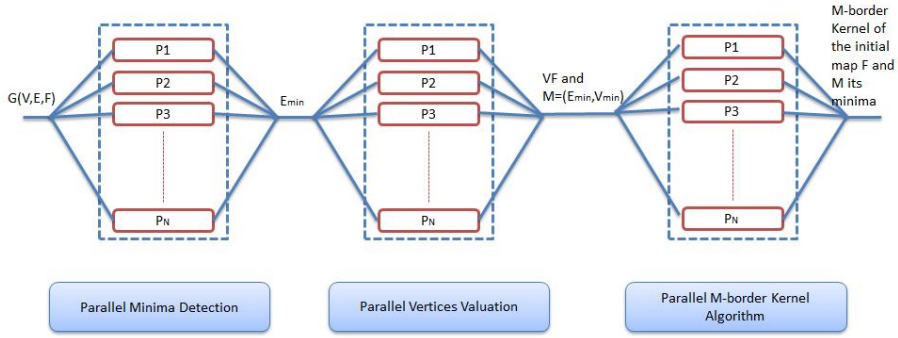

Fig. 10 The proposed parallelization strategy of the Image Segmentation Technique based on the M-border Kernel Algorithm

iteratively the set of edges in a raster scan order. For each edge u, the plateau that contains u is browsed. If a neighbor edge of u whose value is lower than that of u is found, then the edge u and its corresponding plateau are labeled as non-minima edges. If not, the plateau to which the edge u belongs is expanded and labeled as a regional minimum. Figure 11 represents an Edge Weighted Graph and the minima obtained when applying the sequential Regional Minima Detection Algorithm on it.

For the parallelization of this algorithm, we propose a strategy that is based on the Split and Merge model. To prove the correctness of our proposal, we discuss in the following the possible merge configurations that may occur and the decisions taken to remedy each one.

Let G_1 and G_2 be two graphs such that $G_1 \cup G_2 = G$ as shown in Fig. 12 (a). The result of applying the sequential Regional Minima Detection Algorithm independently on these graphs is shown in Fig. 12 (b). This figure shows that two local regional minima are detected respectively in G_1 and G_2. In fact, the minimum of altitude 0 is detected in both graphs. Furthermore, the minimum of altitude 0 detected in G_2 is adjacent to the minimum of altitude 2 detected in G_1. However, the merge of G_1 and G_2 does not lead to the detection of global regional minima of G illustrated in Fig. 11 (b). In fact, plateau of altitude 2 detected as a regional minimum in G_1 is not labeled as minimum in the graph G. Furthermore, the same minimum of altitude 0 is labeled in both graphs G_1 and G_2 by different labels. We distinguish two possible cases: in the first case, a regional minimum is extended on more than one subgraph of G. In the second case, a local regional minimum detected in a subgraph of G has a neighbor edge that belongs to an adjacent subgraph which has a lower altitude.

In order to resolve this issue, regional minima edges that belong to the boundaries of each subgraph must be browsed and their altitudes must be compared to their neighbor edges that belong to adjacent subgraphs. On the basis of this comparison, two decisions are taken: fusion of minima plateaus or deletion of bad minima. This processing is called “Merge Step”. An example of deletion

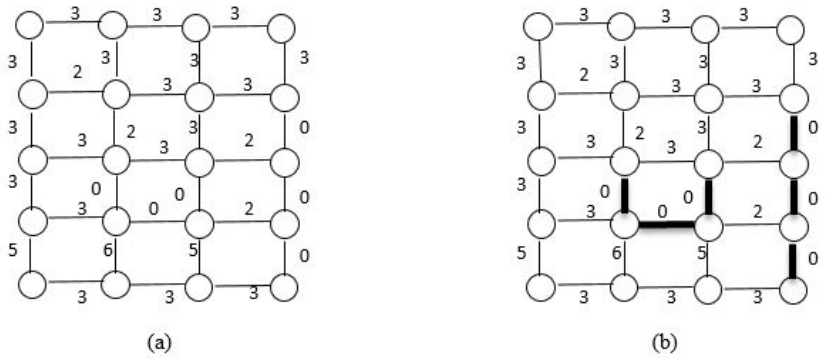

Fig. 11 Regional Minima Detection Algorithm applied on a Weighted Graph G

of bad minima is shown in Fig. 12 (c). In fact, the plateau of altitude 2 that belongs to G_1 has a neighbor minimum edge with an inferior altitude that belongs to G_2. Therefore, this plateau is not considered as minimum anymore and thus its label is deleted. On the other hand, the minima plateaus of altitude 0 detected in G_1 and G_2 belong to the same global minimum. Thus, these minima are merged and labeled by the same label as shown in the same figure. As a result, the minima obtained by applying the sequential Regional Minima Detection Algorithm on the graph partitioned into 2 subgraphs are equal to those obtained by applying the sequential Regional Minima Detection Algorithm on the graph G (cf. Fig. 11 (b) and Fig. 12 (c)).

The regional minima detection in subgraphs G_1 and G_2 is an independent processing. Thus, it can be executed in parallel in order to reduce the execution time of this step.

The main idea of the proposed parallelization strategy of the Regional Minima Detection Algorithm can be summarized as follows: splitting the initial graph into equal and non-overlapped subgraphs or partitions in order to apply the Minima Detection Algorithm locally on each one and then performing the "Merge Step" to obtain the adequate global result. Based on the same methodology, the "Split Step" can be applied to each partition. It leads to increase the parallelism level when performing the Minima Detection Algorithm. The "Merge Step" is applied on pairs of neighbor subgraphs. This principle allows the merging of subgraphs results in a pyramidal way [3,4] in order to perform the independent subgraphs merge in parallel. Thus, the initial graph is partitioned into P subgraphs such that $U_{i=1..P}\ \{G_i\} = G$ with P is the number of available processors. After processing the minima detection and labeling on each partition, the merge of local regional minima is performed in order to obtain global regional minima.

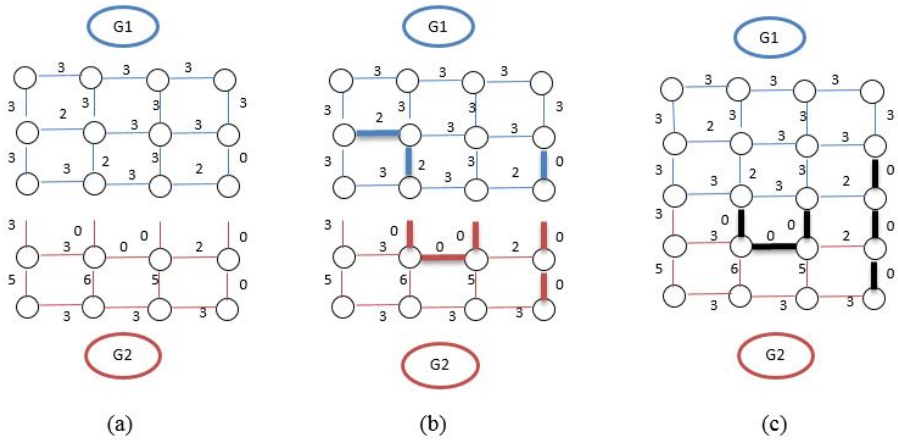

Fig. 12 A Weighted Graph: partitionned into two non-overlapped subgraphs (a) the regional minima obtained in each subgraph independently before the merge step (b) and after the merge step (c)

Algorithm 4: Parallel Regional Minima Detection Algorithm

Data: (V, E, F) a Weighted Graph and P the number of processors.
Result: E_{min} the set of regional minima edges of the initial graph.

```
// Initialization
foreach ( processor p ∈ (1, ... , P) ) do in parallel
    L_p := φ;
E_min := φ;
// Split Step
(partition_1, ...,partition_P):= Static_Partition((V,E,F), P) ;
// Regional Minima Detection applied on partitions in parallel
foreach ( processor p ∈ (1, ... , P) ) do in parallel
    foreach ( u ∈ partition_p ) do
        if (u is not yet processed) then
            E_min := E_min ∪ {u};
            L_p := L_p ∪ {u};
            // Browse the plateau to which u belongs
            while ( L_p ≠ φ ) do
                foreach ( w ∈ L_p ) do
                    L_p := L_p \ {w};
                    foreach ( v ∈ E neighbor of w) do
                        if (( w ∈ E_min) & (F[v] < F[w]) ) then
                            E_min := E_min \ {w}; //w is processed but not
                             labeled as a minimum edge
                            L_p := L_p ∪ {w};
                        else if ( F[v] = F[w] ) then
                            if ( ( w ∈ E_min & v not yet processed ) || ((w
                             is processed and ∉ E_min) & (v not processed
                             or ∉ E_min) ) ) then
                                E_min := E_min ∪ {v};
                                L_p := L_p ∪ {v};
// Merge Step
E_min:=Pyramidal_Merge(F, (E_min(1),..., E_min(P)) ,(partition_1,
    ...,partition_P), P);
```

In the Split Step noted on line 6 of Algorithm 4, each processor computes the boundaries of its corresponding partition. These boundaries are defined by the indexes of the first (start_index) and the last (end_index) horizontal (respectively vertical) edges as noted in Algorithm 5 presented in Annex 1. Note that these boundaries are determined in a way to produce non-overlapped and balanced partitions ($partition_1$, ... , $partition_P$). Then, each processor copies the indexes of edges between "start_index" and "end_index" of both horizontal and vertical edges. The Split Step of the Weighted Graph into P partitions generates $P-1$ boundaries to be merged. In this paper we opt for a pyramidal merge described in [3, 4]. This is allowed due to the fact that such a division ensures that, at each step, the partition containing a given label will be merged with only one partition at a time. Note that a label is present in only one partition. Furthermore, if the neighbor's edges of u belonging to P_{i+1} are labeled as regional minima edges, they must have the same label since they belong to the same partition. The merge step is applied only on adjacent boundaries as noted in Algorithm 6 presented in Annex 2. At the partitions boundaries, each vertical edge u belonging to the partition of a processor P_i has three neighbor edges belonging to the partition corresponding to the processor P_{i+1}: two horizontal edges and one vertical edge (without taking into account the edges situated at the boundaries of the Weighted Graph which have only two neighbor edges). To minimize the scan cost, the value of u is compared only to the lowest neighbor edge belonging to P_{i+1}. It should be reminded that the merge of the results at the boundaries leads either to the fusion of minima labels or to the deletion of labels, i. e. the edges are no longer labeled as minima edges. To minimize the time cost, the update of labels was not made directly on the Weighted Graph. We used a correspondence table [21] to save the equivalences and/or the deletion of labels. Note that initially each processor P_i has its own correspondence table. The size of the local correspondence table of each processor is the number of regional minima in its corresponding partition. Furthermore, each processor labels the regional minima in its corresponding partition by a local set of labels. Then, at each step the local correspondence tables are merged in a pyramidal way [3,4] until a final global correspondence table is obtained. In this context, if the number of partitions is a power of 2 (i. e., $P = 2 \times q$), then the number of steps required to obtain global correspondence table is given by (1) [4]:

$$q = log_2(P) \tag{1}$$

The Average Degree of Parallelism (ADP) is then given by the number of boundaries to be treated divided by number of steps (cf. Eq. 2) [4]:

$$ADP = (2^q - 1)/q \tag{2}$$

Figure 13 illustrates this processing. It represents an 8×8 Weighted Graph partitioned on P=4 processors. Each processor has its local correspondence table which will be merged until obtaining the global correspondence table. In this case, the total number of steps is 2 and the average parallelism is equal

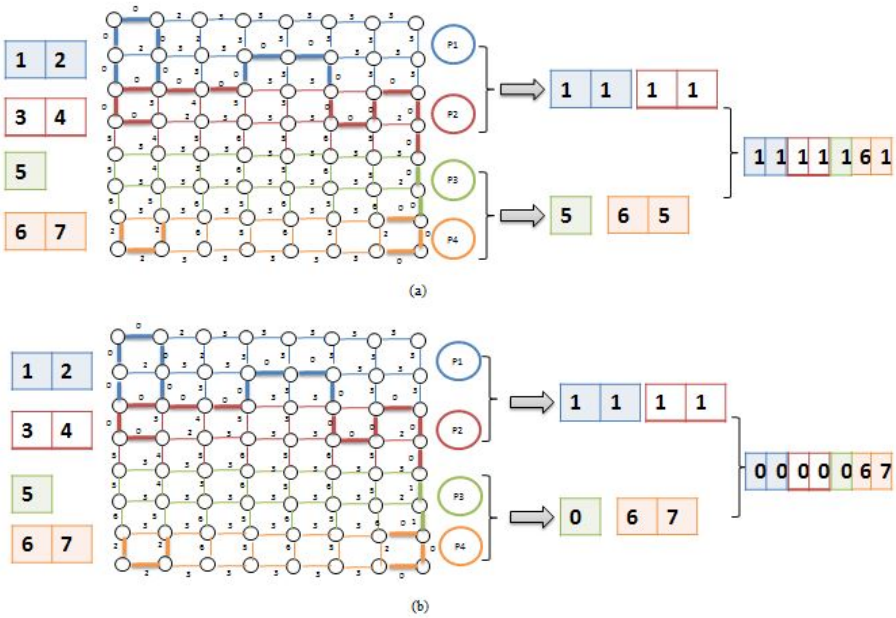

Fig. 13 Pyramidal merge of local results using the correspondence tables: case of equivalence between minima labels (a) and case of deletion of bad minima labels (b)

to 3/2=1.5.
Finally, the labels update step consists in applying, in parallel, the global correspondence table on the graph.

6.2 Parallel Vertices Valuation

The Vertices Valuation step consists in browsing the vertices of the Weighted Graph in a raster scan order and attributing to each vertex the minimum value of the edges to which it belongs. In fact, each vertex x is explored, its neighbor's vertices are examined one by one and its altitude is compared to that of the edge linking the two vertices. If there is an edge with a lower altitude, then this altitude is attributed to x. Furthermore, if x belongs to an edge that is labeled as minimum edge, then, it is labeled as a minimum vertex. This processing is local and does not raise any Data Dependency Issue. Thus, several vertices can be treated in parallel. Consequently, we applied full parallelism for its implementation.

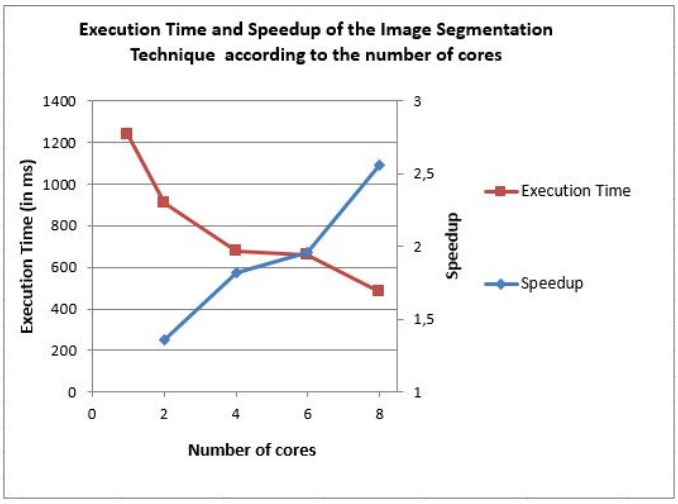

Fig. 14 Execution time and speedup of the parallel Image Segmentation Technique according to the number of cores for a 2048×2048 image

Table 4 Speedup of the parallel Image Segmentation Technique applied on images of différent sizes using 8 cores

Number of images	10	10	10	10	10	10
Resolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Speedup	3.34	2.17	2.46	2.67	2.25	2.09

6.3 Experimental results

As we proceeded for the assessment of the parallel M-border Kernel Algorithm, we evaluated the parallel Image Segmentation in terms of number of cores and images resolutions.

Figure 14 presents the execution time and the speedup of the parallel Image Segmentation Technique according to the number of cores. It shows that this technique allows a gain in terms of execution time. This gain implies a gain in terms of speedup that increases when the number of cores increases. The best speedup factor obtained using 8 cores is equal to 2.59 for a 2048×2048 image.

Figure 15 (respectively 16) presents the execution time (respectively speedup) of this technique applied on the dataset described in Table 1 and implemented using P={2, ..., 8} cores. Note that for each size, the associated execution time (respectively speedup) corresponds to the average values of the execution times (respectively speedup factors) obtained for 10 images. These figures show that our parallelization strategy enabled a gain in terms of execution time (respectively speedup) whatever the images size. Table 4 presents the speedup factors obtained for images of different sizes using 8 cores.

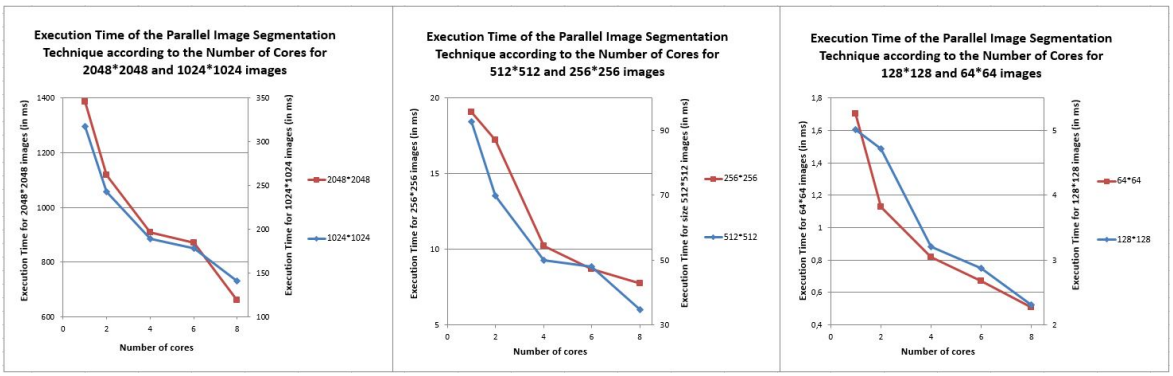

Fig. 15 Execution time of the parallel Image Segmentation Technique based on Watershed Cut computation using the M-border Kernel Algorithm applied on images of different sizes according to the number of cores

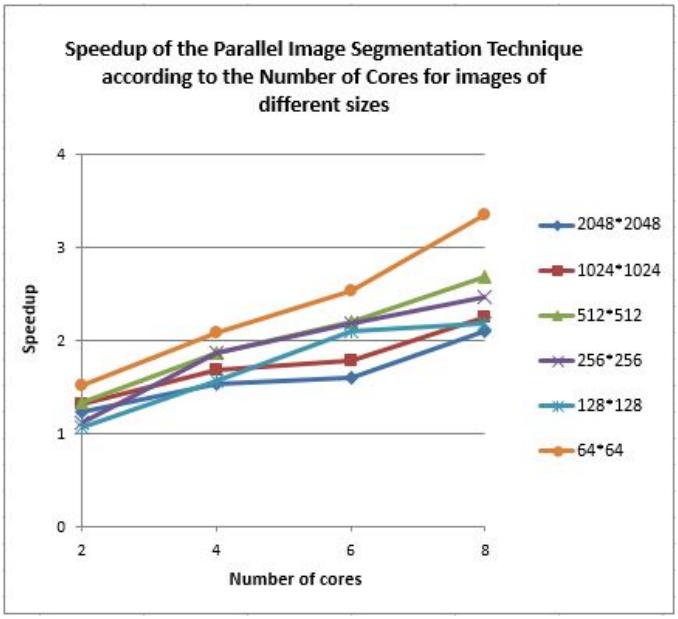

Fig. 16 Speedup of the parallel Image Segmentation Technique based on Watershed Cut computation using the parallel M-border Kernel Algorithm applied on images of different sizes according to the number of cores

In Figures 14 and 15, we note that the drop of the execution time of the parallel Image Segmentation Technique from 4 to 6 cores is lower than that from 6 to 8 cores. This is not due to the M-border Kernel computation step but to the regional minima detection step, and more precisely to the merge step. In fact, for P=4 cores, and according to Eq. 1, only 2 iterations are required to perform the pyramidal merge of local regional minima. For P=6 cores, 3 iterations are required to obtain the global result. Thus, when the number of cores

increases from 4 to 6, the regional minima detection processing time is reduced but the merge cost increases (number of iterations rises from 2 to 3). Consequently, this affects the performance of the whole parallel Image Segmentation Technique. However, when the number of cores rises from 6 to 8, the number of the required merge iterations remains the same (3 iterations). Thus, in that case, the execution time of the regional minima detection processing in each partition is reduced without increasing the merging cost. Therefore, the gain of the regional minima detection step in terms of time is more notable from 6 to 8 cores compared to the one from 4 to 6.

7 Conclusion

In this paper, we tackled the parallelization of a Watershed Cut Algorithm called M-border Kernel Algorithm. We showed that the parallel execution of this algorithm raises a Data Dependency problem at the level of adjacent M-border edges. To overcome this problem, we proposed a parallelization strategy of this algorithm that consists in examining non-minima vertices that are adjacent to minima ones instead of edges in order to obtain a set of edges that constitutes the Watershed Cut. We showed that this strategy allowed to overcome the problem of the simultaneous lowering of two or more adjacent M-border edges that may occur when edge scan is performed. Therefore, a set of independent vertices were processed in parallel. The aim was to reduce the execution time of this Watershed Algorithm while preserving the segmentation quality. The results of the segmentation using the parallel algorithm were similar to those of the sequential Watershed Algorithm which have been demonstrated to produce good results [8, 11]. Furthermore, the experimental results showed that the parallel M-border Kernel Algorithm allowed a gain in terms of execution time and thus speedup. This gain increases when the number of cores rises and is guaranteed whatever the images resolution is. The M-border Kernel Algorithm starts by detecting regional minima of the initial map followed by vertices valuation of the graph. These two preliminary steps were parallelized in order to improve the overall performances of the Image Segmentation Technique based on Watershed Cut computation using the M-border Kernel Algorithm. The experimental results showed that similarly to this algorithm, the speedup factor of the Image Segmentation Technique was guaranteed whatever the images resolution was and increased when the number of cores increased.

References

1. Bertrand, G.: On topological watersheds, J. Math. Imag. Vis, 22, 217–230 (2005).
2. Beucher, S., Lantuéjoul, C. : Use of watersheds in contour detection. Dans Proc. of the International Workshop on Image Processing Real-Time Edge and Motion Detection/ Estimation (1979).
3. Cabaret, L., Lacassagne, L., Etiemble, D.: Parallel Light Speed Labeling: An Efficient Connected Component Labeling Algorithm For Multi-Core Processors, ICIP2015.

4. Cabaret, L., Lacassagne, L., Etiemble, D.: Parallel Light Speed Labeling: an efficient connected component algorithm for labeling and analysis on multi-core processors. Journal of Real-Time Image Processing, (2016).
5. Chazelle, B.: A minimum spanning tree algorithm with inverse-Ackermann type complexity. Journal of the ACM, 1028-1047 (2000).
6. Cormen, T. H., Leiserson, C. and Rivest, R.: Introduction to algorithms. Second edition MIT (2001).
7. Couprie, M., Bertrand, G.: Topological grayscale watershed transform. SPIE Vis. Geomet. 3168 (5), 136–146 (1997).
8. Cousty, J. : Lignes de partage des eaux discrètes : théorie et application à la segmentation d'images cardiaques. Thesis in Computer science, Marne-la-Vallée University (2007).
9. Cousty, J., Bertrand, G., Najman, L., Couprie, M. : On watershed cuts and thinnings. Coeurjolly and al. D. Discrete Geometry for Computer Imagery, France. Springer, 4992 (1), pp.434-445 (2008).
10. Cousty, J., Bertrand, G., Najman, L., Couprie, M.: Watershed Cuts: Minimum Spanning Forests and the Drop of Water Principle. IEEE Transactions on Pattern Analysis and Machine Intelligence, Institute of Electrical and Electronics Engineers, 31 (8), pp.1362-1374 (2009).
11. Cousty, J., Bertrand, G., Najman, L., Couprie, M.: Watershed Cuts: Thinnings, Shortest Path Forests, and Topological Watersheds. IEEE Transactions on Pattern Analysis and Machine Intelligence, Institute of Electrical and Electronics Engineers, 925-939 (2010).
12. Enficiaud, R. : Algorithmes multidimensionnels et multi spectraux en Morphologie Mathématique : Approche par méta-programmation. Thesis in mathematical morphology (2007).
13. Graham, R. L., Hell, P.: On the history of the minimum spanning tree problem. Annals of the History of Computing, 43-57 (1985).
14. Grama, A.: Introduction to Parallel Computing. Pearson Education, Upper Saddle River (2003).
15. Kasim, H., March, V., Zhang, R., See, S.: Survey on parallel programming model. In: Cao, J., Li, M., Wu, MY., Chen, J. (eds.) Network and Parallel Computing. Lecture Notes in Computer Science, vol. 5245. Springer, Berlin, Heidelberg (2008).
16. Kôrbes, A., Giovani, B.V., Janito,V.F., Lotufo, R.A.: A Proposal for a Parallel Watershed Transform Algorithm for Real-Time Segmentation. Proceedings of Workshop de Visao Computacional WVC'2009.
17. Mahmoudi, R., Akil, M.: Analyses of the Watershed Transform. International Journal of Image Processing (5), pp.521-541 (2011).
18. Meyer, F. : Un algorithme optimal de ligne de partage des eaux. Dans Actes du 8ème Congrès AFCET, pages 847-859, Lyon-Villeurbanne, France (1991).
19. Meyer, F.: Topographic distance and watershed lines, Sig. Process. J. Spec. Issue Math. Morphol. Appl. Sig. Process. 38, 113–125 (1993).
20. Najman, L., Couprie, M., Bertrand, G.: Watersheds, mosaics and the emergence paradigm. Discrete Applied Mathematics 147 (2-3), 301-324 (2005).
21. Park, JM., Looney, C.G., Chen, H.C.: Fast Connected Component Labeling Algorithm Using A Divide and Conquer Technique. In the 15th International Conference on Computers and their Applications (2000).
22. Roerdink, J. B. T. M., Meijster,A.: The watershed transform : Definitions, algorithms and parallelization strategies. Fundamental Informatics, 41(1-2), 187-228 (2001).
23. Romero-Zaliz, R., Reinoso-Gordo, J.F.:An Updated Review on Watershed Algorithms. Springer International Publishing AG, C. Cruz Corona (ed.), Soft Computing for Sustainability Science (2018).
24. Soille, P.: Morphological Image Analysis, Springer-Verlag New York, Inc. Secaucus, NJ, USA (1999).
25. Vincent, L., Soille, P. : Watersheds in digital spaces : An efficient algorithm based on immersion simulations. IEEE Trans. Pattern Analysis and Machine Intelligence, 13(6), 583-598 (1991).
26. Lee R Dice: Measures of the amount of ecologic association between species. Ecology, vol. 26, no. 3, 297–302 (1945).
27. Jaccard, Paul. : The distribution of the flora in the alpine zone. New Phytologist, 11, 37–50 (1912).

28. Thomas, H., Charles, E., Ronald, L., Clifford, S.: Introduction to Algorithms. Third edition , 2009.

Annex 1

Algorithm 5: Static_Partition

Data: A Weighted Graph (V, E, F) of size rs*cs and P the number of processors;

Result: $(partition_1, \ldots, partition_P)$ partitions to be treated in parallel by P processors.

```
//Initialization
Q:= cs/P;
R:= cs mod P; // R is the rest of the division of cs by P
foreach (Processor p ∈ {1,...,P}) do in parallel
    if (p ≤ R) then
        cs_start[p]:=(p-1)*(Q+1);
        cs_end[p]:=cs_start[p] + Q;
    else
        cs_start[p]:=(p-1)*Q+R;
        cs_end[p]:=cs_start [p]+ Q - 1;
    //First index of horizontal edges
    HE_StartIndex[p] := cs_start[p] * rs;
    //Last index of horizontal edges
    HE_EndIndex[p] := (cs_end[p] + 1) * rs − 2;
    //First index of Vertical edges
    VE_StartIndex[p] := N + cs_start[p] * rs;
    //Last index of horizontal edges
    VE_EndIndex[p] := N + (cs_end[p] + 1) * rs − 1;
    partition_p:=E[[HE_StartIndex[p],...,HE_EndIndex[p]] +
      [VE_StartIndex[p],...,VE_EndIndex[p]]];
```

Annex 2

Algorithm 6: Pyramidal_Merge

```
Data: a map F and its minima M(F)
Data: (P_1, ...,P_N)set of partitions and N the number of processors.
Result: M(F) Merged Minima of F.
// Initialization
Nb_bords := N − 1; // Number of boundaries to process
Start:=0;
while (start < Nb_bords) do
    // Do in Parallel
    foreach (Partition P_i such that i ∈ (1, ... , N)) do in parallel
        if ( ( (P_i - Start) mod Step ) = 0 ) then
            True:=1;
        IsActive:=( True & (p < (N-1)) ) ;
        if (IsActive) then
            //Browse the low boundary of partition
            foreach ( u ∈ BP_i ) do
                //Compute the neighbour edge of u of the lowest altitude
                v:= MinVoisin (u); // MinVoisin is a function that returns
                 the edge neighbor v of an edge u which have the lowest
                 altitude
                if ( F(u) = F(v) ) then
                    if (u ∈ M(F) & v ∈ M(F)) then
                        // Merge plateaus of the minima u and v by
                         attributing the same label
                    else if (u ∈ M(F) & v ∉ M(F)) then
                        // Delete plateau of u from the minima set
                    else if (u ∉ M(F) & v ∈ M(F)) then
                        // Delete plateau of v from the minima set
                if ( F(u) < F(v) ) then
                    if (v ∈ M(F)) then
                        // Delete plateau of v from the minima set
                if ( F(u) > F(v) ) then
                    if (u ∈ M(F)) then
                        // Delete plateau of u from the minima set
    // Final Update of the equivalence table to obtain global labels
    foreach ( i ∈ (1, ... , Nb_Labels )) do
        h := i;
        while ( TabEq[h] ≠ h ) do
            h := TabEq[h];
        TabEq[i] := h;
    Start:=Step-1;
    Step:=2*Step;
```

Table 5 Symbols used in Algorithms

Symbol	Symbol Name	Meaning/Definition
mod	modulo	the rest of the division: remainder calculation
$*$	multiplication	multiplication
$-$	minus sign	substraction
$=$	equal sign	equality
$<$	strict inequality	less than
$>$	strict inequality	greater than
$\neq$	not equal sign	inequality
$\in$	element of	set membership
$\notin$	not an element of	no set membership
$:=$	assignment	assignment
ϕ	empty set	ϕ={}
&	and	and
$\|\|$	or	or
$A \setminus B$	relative complement	elements that belong to A and not to B
$\mathbb{Z}$	integer numbers set	$\mathbb{Z} = \{... - 3, -2, -1, 0, 1, 2, 3, ...\}$
$\mathbb{N}$	natural numbers set	$\mathbb{N} = \{0, 1, 2, 3, ...\}$