Parallélisation de la ligne de partage des eaux dans le cadre des graphes à arêtes valuées sur architecture multi-cœurs

Yosra Braham

THÈSE DE DOCTORAT EN COTUTELLE

Présentée à

L'ECOLE NATIONALE D'INGENIEURS DE MONASTIR

Pour obtenir le grade de Docteur

De l'Université Paris-Est MLV
Ecole Doctorale Mathématique et STIC
Spécialité : Informatique

De l'Université de Monastir
Ecole Doctorale Sciences et Technologies
Spécialité : Génie Electrique

Intitulée

Parallélisation de la Ligne de Partage des Eaux dans le cadre des Graphes à Arêtes Valuées sur architecture multi-cœurs

Elaborée par

Yosra BRAHAM

Soutenue le 24 Novembre 2018, devant le Jury composé de :

Mme. Najoua ESSOUKRI BEN AMARA	Professeur à l'ENISo, Université de Sousse	Présidente du Jury
M. Olivier DEFORGES	Professeur, INSA de Rennes	Rapporteur
Mme. Jihène MALEK	Maître de conférences à ISSATS, Université de Sousse	Rapporteur
M. Serge WEBER	Professeur à Université de Lorraine	Examinateur
M. Mohamed AKIL	Professeur émérite, ESIEE-Paris, Université Paris-Est	Directeur de thèse
M. Mohamed Hédi BEDOUI	Professeur à la FM-Monastir, Université de Monastir	Directeur de thèse

Laboratoire de Recherche LTIM (LR12ES06), FMM, Monastir, Tunisie
Laboratoire de Recherche LIGM (UMR 8049 CNRS), Equipe A3SI, ESIEE-Paris, France

Année Universitaire: 2018/2019

Cette thèse est dédiée à mes parents,

sans lesquels je n'aurais jamais vu le jour.

Mon mari Karim,

Ma fille Kenza,

Toute ma famille, ma belle-famille et tous mes amis.

Remerciements

Je tiens à exprimer mes vifs remerciements à Madame **Najoua Essoukri Ben Amara** Professeur à l'Ecole Nationale d'Ingénieurs de Sousse, pour le grand honneur qu'elle m'a fait en acceptant de présider ce Jury.

Mes remerciements vont également à Monsieur **Olivier Déforges** Professeur à l'INSA de Rennes et Madame **Jihène Malek** maître de conférences à l'ISSAT Sousse pour avoir accepté d'évaluer ce travail,

Je remercie aussi Monsieur **Serge Weber** Professeur à l'Université de Lorraine-Nancy pour avoir accepté de faire partie de ce jury.

Je tiens par ailleurs à exprimer toute ma reconnaissance à ceux qui ont contribué à ce travail, en particulier mes directeurs de thèse Monsieur **Mohamed Akil** et Monsieur **Mohamed Hédi Bedoui** de m'avoir dirigé et encadré tout au long de mes travaux de recherche, pour leurs disponibilités et leurs efforts pour la réalisation de ce travail.

Un grand merci à **Yaareb Elloumi** Maître-assistant à l'ISITCOM et membre du laboratoire TIM à la Faculté de Médecine de Monastir pour ses conseils et son aide à l'élaboration de ce travail.

Je tiens à remercier aussi mes collègues et amis du laboratoire TIM à la Faculté de Médecine de Monastir et de l'équipe A3SI à l'ESIEE Paris. Une fois de plus je remercie mon père Abdelmajid, ma mère Radhia, mon beau-père Abdelmajid, ma belle-mère Monia, ma sœur Nadia, mon frère Karim, ma belle-sœur Rania et mon beau-frère Omar pour leurs soutiens et leurs encouragements durant ces années d'études. Plus personnellement, je remercie mon mari, pour son aide, son écoute et surtout son amour qui m'a été essentiel durant ces années. Je remercie tous mes plus proches amis Arij, Omelkhir, Imen, Ibtissem, Maroua, Ines, Imane, Ines, Zina, Oussema, Mahmoud.

Résumé

Notre travail s'inscrit dans le cadre de la parallélisation d'algorithmes de calcul de la Ligne de Partage des Eaux (LPE) en particulier la LPE d'arêtes qui est une notion de la LPE introduite dans le cadre des Graphes à Arêtes Valuées. Nous avons élaboré un état d'art sur les algorithmes séquentiels de calcul de la LPE afin de motiver le choix de l'algorithme qui fait l'objet de notre étude qui est l'algorithme de calcul de noyau par M-bord. L'objectif majeur de cette thèse est de paralléliser cet algorithme afin de réduire son temps de calcul. En premier lieu, nous avons présenté les travaux qui se sont intéressés à la parallélisation des différentes variantes de la LPE et ce afin de dégager les problématiques que soulèvent cette tâche et les solutions adéquates à notre contexte. Dans un second lieu, nous avons montré que malgré la localité de l'opération de base de cet algorithme qui est l'abaissement de la valeur de certaines arêtes nommées arêtes M-bord, son exécution parallèle se trouve pénalisé par un problème de dépendance de données, en particulier au niveau des arêtes M-bord qui ont un sommet non minimum commun. Dans ce contexte, nous avons proposé trois stratégies de parallélisation de cet algorithme visant à résoudre ce problème de dépendance de données. La première stratégie consiste à diviser le graphe de départ en un ensemble de partitions qui sont traitées en parallèle par *P* processeurs. Les résultats partiels obtenus au niveau de chaque partition sont ensuite fusionnés pour aboutir au résultat global. La deuxième stratégie consiste à diviser les arêtes du graphe de départ en des sous-ensembles d'arêtes indépendantes obtenus en alternant selon un certain ordre les arêtes horizontales et les arêtes verticales. Les arêtes de chaque sous-ensemble sont ensuite traitées en parallèle. La troisième stratégie consiste à examiner les sommets au lieu des arêtes du graphe initial tout en préservant le paradigme d'amincissement sur lequel est basé l'algorithme séquentiel initial. Par conséquent, l'ensemble des sommets non-minima adjacents aux sommets minima sont traités en parallèle. En dernier lieu, nous avons étudié la parallélisation d'une technique de segmentation basée sur l'algorithme de calcul de noyau par M-bord. Cette technique comprend les étapes suivantes : la recherche des minima régionaux, la pondération des sommets et le calcul des sommets minima et enfin calcul du noyau par M-bord. A cet égard, nous avons commencé par faire une étude relative à la dépendance des données des différentes étapes qui la constituent et nous avons proposé des algorithmes parallèles pour chacune d'entre elles.

Afin d'évaluer nos contributions, nous avons implémenté les différents algorithmes parallèles proposés dans le cadre de cette thèse sur une architecture multi-cœurs à mémoire partagée. Les résultats obtenus ont montré des gains en termes de temps d'exécution. Ce gain est traduit par des facteurs d'accélération qui augmentent avec le nombre de processeurs et ce quel que soit la taille des images à segmenter.

Mots clé: Segmentation d'images en niveaux de gris; Ligne de Partage des Eaux; Graphe; Algorithmes Parallèles; Architectures Multi-cœurs.

Abstract

This work deals with the parallelization of a particular notion of the Watershed Transform defined in the framework of the Edge Weighted Graphs. It reviews first the related works which were interested in the parallelization of the others notions of the watershed in order to identify the main challenges that such a task raises. Then, it focuses on a specific algorithm called the M-border Kernel algorithm with a main objective to optimize its running time. Study carried out in this context showed that despite the locality of the edges lowering operation on which this algorithm is based, the parallelization of this latter raises a data dependency problem at the level of the M-border edges having a common non-minimum vertex.

Three parallelization strategies are proposed to solve this issue. The first strategy consists in dividing the initial graph into partitions processed in parallel by *P* processors. Partial results obtained at the level of each partition are then merged to elaborate the global result. The second strategy consists in dividing the graph edges into independent subsets obtained by alternating according to a specific order horizontal and vertical edges. Each subset edges are then processed in parallel. The third strategy consists in scanning the graph by vertices instead of edges while preserving the thinning paradigm on which the sequential algorithm is based. Therefore, the set of non-minima vertices adjacent to the minima ones are processed in parallel.

Based on these contributions, a parallelization of a segmentation technique based on the M-border kernel algorithm is proposed. This technique is composed of three main steps which are: regional minima detection, vertices valuation and M-border kernel computation. For the two first steps a parallelization strategy is proposed while for the M-border computation the third strategy presented above was adopted.

The proposed algorithms were implemented on a shared memory multi-core architecture. The obtained results show a notable gain in terms of speedup factors that increases with the number of processors whatever the resolution of the input images is.

Keywords: Gray Scale Image Segmentation; Watershed; Graphs; Parallel Algorithms; Multi-core Architectures.

Table des matières

Chapitre 2: Calcul parallèle et Lignes de Partage des Eaux

Chapitre 3: Implémentation parallèle d'un algorithme de calcul de la LPE d'arêtes sur architecture multi-cœurs

Liste des figures

Liste des tableaux

Liste des Acronymes

LPE Ligne de Partage des Eaux

IRM Imagerie par Résonance Magnétique

FAS Filtre Alterné Séquentiel

FAH File d'Attente Hiérarchique

CPU Central Processing Unit

GSP Graphe à Sommets Pondérés

GAV Graphe à Arêtes Valuées

RVB Rouge Vert Bleu

FIFO First In First Out

LCI Lower Completed Image

SPF Shortest Path Forest

PCAM Partitioning communication Agglomeration Mapping

SISD Single Instruction Single Data

MISD Multiple Instruction Single Data

SIMD Single Instruction Multiple Data

MIMD Multiple Instruction Multiple Data

SMP Symmetric multi-Processing

PRAM Parallel Random Access Machine

GPU Graphics Processing Units

OpenMP Open Multi-Processing

MPI Message Passing Interface

PVM Parallel Virtual Machine

SPMD Single Program, Multiple Data

NARM Not A Regional Minimum

BFS Breadth First Search

FPGA Field-Programmable Gate Array

BCG Boundary Component Graph

AMD Advanced Micro Devices

API Application Programming Interface

DFS Depth First Search

MR Minimum Régional

SMR Sous-Minimum Régional

FMR Faux Minimum Régional

ADP Average Degree of Parallelism

Introduction Générale

En traitement d'images, la segmentation par la Ligne de Partage des Eaux (LPE) désigne une famille de méthodes de segmentation issues de la Morphologie Mathématique qui considère une image à niveaux de gris comme un relief topographique tel que l'altitude d'un pixel correspond à son niveau de gris. Cette famille, introduite par H. Digabel et C. Lantuéjoul [3], permet de partitionner les pixels d'une image en un ensemble de régions connexes séparées par un contour fermé qui constitue la LPE.

Les LPE peuvent être classées en trois principales catégories : la LPE par immersion qui consiste à simuler une montée progressive du niveau d'eau à partir des minima du relief, la LPE par ruissellement qui consiste à suivre, à partir de chaque pixel de l'image, la ligne de plus grande pente jusqu'à atteindre un minimum et la LPE topologique qui consiste à déformer progressivement le relief en préservant certaines caractéristiques topologiques jusqu'à ce qu'il soit réduit à une structure fine correspondant à la LPE.

Dans ses différentes variantes, la LPE se présente comme une méthode de traitement qui demande dans la majorité des cas, plusieurs parcours de l'image, la construction de graphes et le passage par des structures de données dynamiques et complexes [35, 86]. Ceci fait de la LPE une méthode de segmentation relativement gourmande en termes de temps d'exécution, et ce, malgré son efficacité et ses propriétés intéressantes. Ce problème se manifeste davantage dans les domaines où l'on manipule des images de grandes tailles tels que l'imagerie médicale dans ses versions 2D, 3D et 4D. Fournir des résultats de segmentation par LPE en des temps raisonnables devient un défi qu'il faut relever pour une exploitation pratique de cette technique.

1. Cadre et motivation

La parallélisation de la Ligne de Partage des Eaux constitue une solution pour optimiser cette transformation en termes de temps d'exécution. Depuis son introduction, plusieurs travaux se sont intéressés à la parallélisation de la LPE. Dans ce contexte, les stratégies proposées dans la littérature ont traité différentes versions séquentielles de cette famille de méthodes. Les travaux menés dans le cadre de cette thèse s'insèrent dans cette lignée. Ils s'intéressent, en effet, à la parallélisation d'une notion particulière de la LPE nommée LPE d'arêtes (ou *coupures par LPE*) introduite dans [7, 16, 17,22]. Les algorithmes de calcul de cette variante de la LPE sont parmi les algorithmes séquentiels de calcul de cette

transformation les plus performants en termes de qualité de résultat notamment en ce qui concerne la précision des contours obtenus.

Ces travaux entrent dans un cadre plus général du projet « *cœur* » du Laboratoire de Technologies et Imagerie Médicale " LR12ES06" de l'Université de Monastir [97,108,109] et du projet de recherche portant sur l'étude de la dynamique du cœur mené au sein du Laboratoire LIGM, équipe A3SI – ESIEE Paris [1, 79, 80]. Ce projet a pour objectif d'aider les médecins à interpréter les séquences spatio-temporelles d'IRM cardiaque. Ainsi, il consiste en la mise en œuvre des procédures informatiques d'analyse et de visualisation de séquences temporelles constituées d'une suite d'images cardiaques tridimensionnelles acquises au cours du cycle cardiaque en se basant sur l'étude d'outils dérivés de la Morphologie Mathématique tel que les transformations géodésiques (dilatation et érosion), le Filtre Alterné Séquentiel (FAS), les transformations homotopiques, la LPE, etc. Ce projet a abouti à la proposition d'une chaîne de traitements séquentiels qui assure la segmentation et la visualisation 3D/4D (3D+t) du ventricule gauche du cœur à partir de coupes bidimensionnelles d'IRM cardiaque décalées dans le temps et dans l'espace (Ciné-IRM). Dans cette chaîne (voir Figure 1), l'étape de segmentation constitue l'étape la plus importante. Elle est elle-même constituée de trois sous étapes permettant de segmenter respectivement la frontière endocardique, la frontière épicardique et le muscle myocardique. Ces étapes font appel à des traitements qui sont issus de la Morphologie Mathématique en l'occurrence :

- la dilatation géodésique, la dilatation adaptative et les transformations homotopiques qui assurent la reconnaissance des régions,
- la Ligne de Partage des Eaux (LPE) qui permet la délinéation des frontières entre les régions,
- le filtre alterné séquentiel (FAS) pour le lissage des frontières segmentées.

Par rapport à cette chaîne, l'objectif du projet dans lequel s'inscrivent les travaux de cette thèse consiste à son optimisation en termes de temps d'exécution. Parmi les trois sous-étapes de segmentation mentionnées ci-dessus, nous nous sommes focalisés sur l'étape de délinéation de l'épicarde en utilisant la technique de la LPE car c'est l'étape la plus coûteuse en termes de temps de calcul. Cette étape est basée sur un algorithme de calcul de la LPE d'arêtes dans le cadre des graphes à arêtes valuées [17] qui est l'algorithme de calcul de noyau par M-bord [30]. Nous nous intéressons à cet algorithme d'une part, vu son efficacité

parmi les méthodes existantes dans la littérature de calcul de LPE et d'autre part, vu la qualité des résultats qu'il produit et qui ont été validés dans le contexte de l'imagerie cardiaque.

Cet algorithme considère les images comme graphes dont les arêtes sont valuées. Il opère par une transformation d'amincissement qui consiste à abaisser, jusqu'à idempotence, les valeurs de certaines arêtes satisfaisant une propriété simple qui peut être testée localement (les arêtes M-bord).

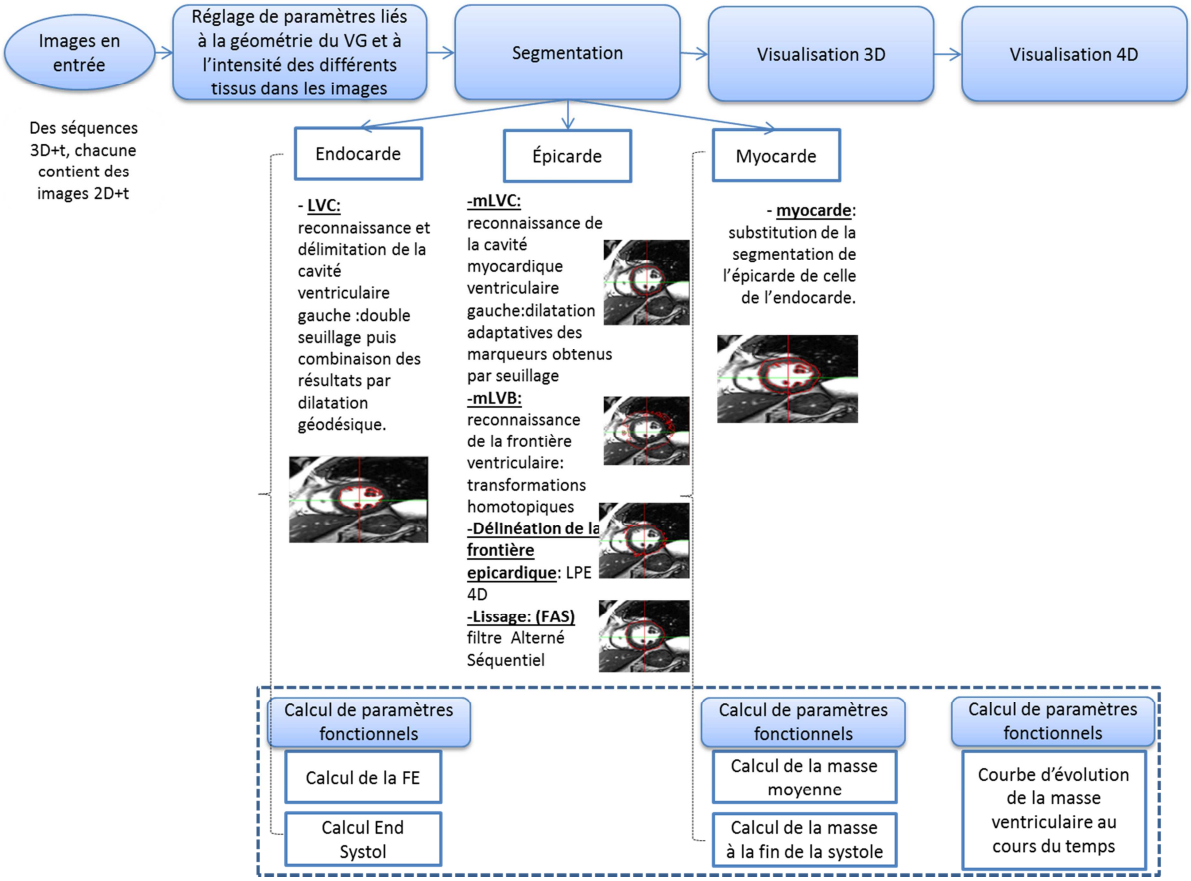


Figure 1. Chaîne de traitement de l'outil de segmentation et reconstruction 3D du myocarde ventriculaire gauche à partir de coupes 3D+t d'IRM cardiaque

La majorité des algorithmes de calcul de la LPE qui existent dans la littérature, parmi lesquels [5, 6, 9, 10, 11, 13, 14, 18, 21, 31, 51], nécessitent, soit une étape de tri, soit l'utilisation d'une File d'Attente Hiérarchique (FAH), soit une structure de données complexes. En revanche, l'algorithme de calcul de noyau par M-bord [30] ne nécessite aucune de ces étapes et il fonctionne en un temps linéaire. De plus, il est basé sur une propriété locale ce qui le rend adapté au parallélisme.

Pour la mise en œuvre du parallélisme, nous avons opté pour des architectures de plus en plus évoluées tels que les processeurs CPU multi-cœurs. Ces architectures permettent d'accélérer les algorithmes grâce aux différents niveaux de parallélismes qu'elles offrent (parallélisme d'instructions, parallélisme de données et parallélisme des threads). Ceci a permis de concevoir des implantations parallèles d'algorithmes de calcul de LPE, plus rapides.

2. Organisation de la thèse

Le présent manuscrit est structuré en quatre chapitres :

Dans le chapitre 1, nous commençons par rappeler les différentes définitions de la LPE ainsi que les algorithmes séquentiels permettant de la calculer que ce soit dans le cadre des graphes à sommets pondérés (GSP) ou des graphes à arêtes valuées (GAV). Nous présentons ainsi la notion de la LPE d'arêtes ou coupures par LPE introduite dans [16, 17,22] dans le cadre des GAV. Cette notion de la LPE peut être définie d'une façon équivalente par les bassins d'attraction ou par les frontières qui les séparent. Nous présentons dans ce cadre deux algorithmes linéaires [8,30] qui permettent de la calculer. Nous dressons enfin un bilan sur les différents algorithmes séquentiels de calcul de la LPE étudiés en se basant sur des critères de comparaison qui touchent principalement l'aspect algorithmique d'un côté et la qualité des résultats de segmentation qu'ils produisent d'un autre côté.

Dans le chapitre 2 nous abordons la problématique de l'accélération de la LPE par parallélisation. Dans ce cadre nous commençons par rappeler les concepts et techniques de base du calcul parallèle. Nous mettons l'accent surtout sur les architectures qui vont nous servir de support d'implémentation, à savoir les architectures multi-cœurs à mémoire partagée. Dans la deuxième partie de ce chapitre nous passons en revue les travaux de référence qui se sont intéressés à la parallélisation de la LPE. Nous dressons dans ce cadre un bilan sur ces travaux, dans lequel nous citons les principales stratégies de parallélisation qui ont été adoptées.

Dans le chapitre 3, nous commençons tout d'abord par détailler la variante de la LPE qui fera l'objet de la parallélisation dans ce travail en l'occurrence l'algorithme de calcul de noyau par M-bord. Nous rappelons aussi les caractéristiques qui ont motivé, entre autres, l'utilisation de cet algorithme. Nous présentons ensuite l'étude que nous avons menée sur cet algorithme en termes d'étapes qui la composent et d'analyse des dépendances de données.

Une telle étude étant primordiale en vue d'élaborer les stratégies de parallélisation adéquates. Ces stratégies feront l'objet de la partie suivante du chapitre. Nous détaillons, dans ce contexte, les différentes propositions que nous avons élaborées à cet égard, tout en discutant et motivant nos propositions. Enfin, nous présentons et discutons, dans la dernière partie de ce chapitre, les principaux résultats expérimentaux que nous avons obtenus suite à la mise en œuvre de nos contributions sur une architecture multi-cœurs à mémoire partagée.

Dans le chapitre 4, nous présentons une technique de segmentation basée sur l'algorithme de calcul de noyau par M-bord. Dans la première partie de ce chapitre, nous détaillons les différentes étapes de cette technique et étudions la dépendance des données de chaque étape. Suite à cette étude, nous proposons une stratégie de parallélisation pour chacune de ces étapes afin d'optimiser la totalité de la technique de segmentation en termes de temps d'exécution. Dans la deuxième partie de ce chapitre, nous présentons et discutons les résultats obtenus suite à l'implémentation des différents algorithmes parallèles proposés sur la même architecture multi-cœurs utilisée dans l'expérimentation et l'évaluation des algorithmes parallèles de calcul de noyau par M-bord proposés dans le chapitre 3.

Nous clôturons enfin ce manuscrit par une conclusion générale dans laquelle nous dressons un bilan sur le travail élaboré et les perspectives qui permettent de le compléter et de l'améliorer.

Chapitre 1 : Ligne de Partage des Eaux et segmentation

1. **Introduction**
2. **Définition mathématique d'une image discrète**
   - **2.1. Notion de connexité et de voisinage**
   - **2.2. Représentation d'une image numérique**
3. **Notions communes et notation**
   - **3.1. Notion de Graphe**
   - **3.2. Graphe à arêtes valuées et graphe à sommets pondérés**
4. **Représentation d'une image discrète par un graphe**
5. **Segmentation par Ligne de Partage des Eaux**
6. **Approches de calcul de la Ligne de Partage des Eaux**
   - **6.1. La LPE par immersion**
   - **6.2. La LPE par ruissellement**
   - **6.3. La LPE topologique**
7. **Ligne de Partage des Eaux et sur-segmentation**
8. **LPE, image et graphe**
9. **Algorithmes de calcul de la Ligne de Partage des Eaux**
   - **9.1. Algorithmes de calcul de la LPE dans le cadre des graphes à sommets pondérés**
   - **9.2. Algorithmes de calcul de la LPE dans le cadre des graphes à arêtes valuées**
   - **9.3. Bilan sur les algorithmes séquentiels de calcul de la LPE**
10. **Conclusion**

1. Introduction

En topographie, la Ligne de Partage des Eaux (LPE) a été étudiée dès le 19ème siècle par Maxwell [1] et Jordan [2]. Elle a été introduite dans le domaine du traitement d'images, par Digabel et Lantuéjoul [3] comme une étape fondamentale pour la segmentation d'images. Elle désigne dans ce cadre une famille de méthodes issues de la Morphologie Mathématique qui considère une image à niveaux de gris comme un relief topographique, et ce en associant à chaque pixel une altitude qui correspond à son niveau de gris. Dans ce relief, les régions de l'image correspondent aux bassins versants (les vallées du relief). La segmentation revient dans ce cas à trouver les crêtes formant les limites entre ces bassins versants (les contours entre les régions).

Dans ce chapitre nous nous intéressons à cette famille de méthodes de segmentation. Pour ce faire, nous commençons d'abord par introduire les notions de base nécessaires à sa définition notamment les notions d'image discrète, de voisinage et de graphe. Nous présentons ensuite le fondement de cette méthode de segmentation. Nous passons en revue dans ce cadre ses différentes variantes et détaillons le phénomène physique engendré par chacune d'elles. Nous terminons enfin par une étude des algorithmes les plus connus permettant de calculer et de mettre en œuvre ces différentes variantes [9, 10, 70, 11, 21, 66, 4, 20, 17].

2. Définition mathématique d'une image discrète

Une image discrète I est considérée comme une fonction dont le domaine de définition, noté $D(I)$ est un espace discret représentant une partie de $\mathbb{Z}^n$: $D(I) \subseteq \mathbb{Z}^n$. Dans cette définition n est un entier positif supérieur à 1 ($n \geq 2$) et désigne la dimension de l'image. Le domaine $D(I)$ prend la forme $[0, d_1]\times[0, d_2]\times\ldots\times[0, d_n]$. Un élément de l'image ainsi définie correspond à un point p de cet espace de coordonnées $(x_1, x_2,\ldots, x_n)$.

En se basant sur cette définition, une valeur de $n = 2$ permet de retrouver le cas de l'image bidimensionnelle (*2D*). Un élément de l'image, appelé dans ce cas *Pixel*, est représenté par un carré d'arête 1 et centré en un point de $\mathbb{Z}^2$. De même, une valeur de $n = 3$, permet de modéliser le cas de l'image tridimensionnelle (*3D*) dans laquelle chaque élément appelé *voxel* est représenté par un cube d'arête égale à 1 et centré en un point de $\mathbb{Z}^3$. Notons que le cas où $n = 4$ fait souvent référence à une séquence temporelle d'images *3D*.

Le domaine d'application de l'image est borné. Il est donné par :

– [0; 255] pour une image en niveaux de gris où 0 correspond au noir et 255 correspond au blanc;
– $[0; 255]^3$ pour une image en couleurs *RVB* (Rouge, Vert, Bleu) où chacun des canaux est échelonné de la même façon qu'une image en niveaux de gris.

2.1 Notions de connexité et de voisinage

Sur une grille discrète de $\mathbb{Z}^n$, deux points sont considérés comme connexes si la distance qui les sépare est égale à 1. Dans la littérature, il existe plusieurs métriques permettant de mesurer la distance entre deux points p et q d'un espace discret. Parmi les distances les plus communes dans ce cadre nous citons :

- La distance de Manhattan :

$$\|p - q\|_1 = \sum_{i=1}^{n} |p_i - q_i| \qquad (1.1)$$

- La distance de Tchebychev :

$$\|p - q\|_\infty = \max(|p_1 - q_1|, \dots, \dots, |p_n - q_n|), 1 \leq i \leq n \qquad (1.2)$$

Dans une grille, la distance de Manhattan permet de tenir compte des déplacements verticaux et horizontaux alors que la distance de Tchebychev permet de tenir compte des déplacements dans les directions horizontales verticales et diagonales.

Le voisinage d'un point p est donné par l'ensemble des points qui sont immédiatement connexes à p. Selon la distance utilisée et les types de déplacements qu'elle permet, différents types de voisinages peuvent être obtenus. Cette notion de voisinage est formalisée par la définition suivante :

Définition d'un voisinage [89]:

Etant donné un point $p \in \mathbb{Z}^n$ et un entier tel que $0 \leq k < n$. Autour d'un point p, il est possible de définir n voisinages, notés v_k. Chaque v_k est donné par l'ensemble des points $q \in \mathbb{Z}^n$ qui vérifient les deux conditions suivantes :

$$\|p - q\|_\infty \leq 1 \; et \|p - q\|_1 \leq n - k \qquad (1.3)$$

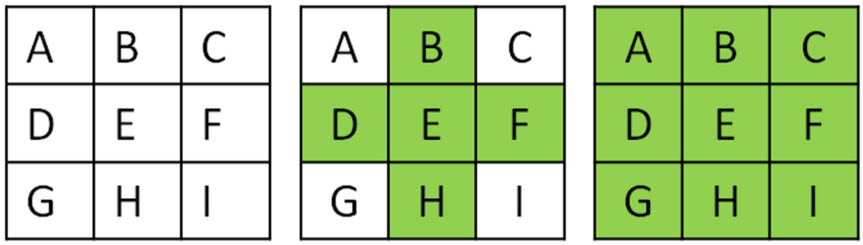
Partant de cette définition plusieurs voisinages peuvent être obtenus pour une partie d'un espace discret de $\mathbb{Z}^n$. Dans le cadre des images discrètes, ces voisinages, bien qu'ils soient

valables peuvent produire des résultats différents pour un même algorithme selon que l'un ou l'autre est utilisé.

Ainsi dans $\mathbb{Z}^2$ ($n = 2$), qui correspond au cas de l'image 2D, il est possible de distinguer :

- le ***4-voisinage*** (correspond à $k=1$) : deux pixels sont voisins s'ils ont un côté en commun. Ainsi chaque pixel n'appartenant pas au bord de l'image, possède 4 pixels voisins selon les directions verticales et horizontales;
- le ***8-voisinage*** (correspond à $k=0$): deux pixels sont voisins s'ils ont au moins un coin en commun, ainsi chaque pixel n'appartenant pas au bord de l'image possède 8 pixels voisins selon les trois directions horizontales, verticales et diagonales.

La figure 1.1 illustre ces deux cas. En effet, si nous considérons l'image *2D* de taille *3×3* de la figure 1.1 (a), associée au *4*-voisinage, alors le pixel *E* a pour voisins les pixels {*B, D, F, H*}. Par contre, si elle est associée au *8*-voisinage, alors le pixel *E* a pour voisins les pixels {*A, B, C, D, F, G, H, I*}.



(a) ***(b)*** ***(c)***

Figure 1. 1 (a) Image 2D de taille 3*3 (b) cas du 4-voisinage et (c) cas du 8-voisinage

Dans $\mathbb{Z}^3$ ($n = 3$), qui correspond au cas de l'image 3D, trois types de voisinage peuvent être distingués:

- le ***6-voisinage*** (correspond à $k=2$): deux voxels sont voisins s'ils ont une face en commun. Ainsi chaque voxel qui n'appartient pas à la frontière de l'image possède 6 voxels voisins;
- le ***18-voisinage*** (correspond à $k=1$): deux voxels sont voisins s'ils ont au moins une arête en commun, ainsi chaque voxel qui n'est pas au bord de l'image possède 18 voxels voisins;
- le ***26-voisinage*** (correspond à $k=0$): deux voxels sont voisins s'ils ont au moins un sommet en commun, ainsi chaque voxel qui n'est pas au bord de l'image possède 26 voxels voisins.

2.2 Représentation d'une image numérique

Une image discrète est généralement représentée sous forme vectorielle (matrice). Toutefois certaines techniques de traitement d'images nécessitent des structures de données

plus élaborées qui permettent de coder, en plus de l'information intrinsèque à chaque pixel, des informations supplémentaires relatives souvent aux notions de voisinage et aux relations spatiales entre les différentes composantes de l'image. A cet égard, les graphes offrent un cadre bien adapté.

Certains algorithmes de calcul de la Ligne de Partage des Eaux [8, 9, 15, 17, 18, 21, 22, 23, 30, 56] utilisent les graphes comme support de représentation de l'image en particulier la variante de la LPE qui fait l'objet de notre étude dans le cadre de cette thèse [8, 17, 30]. Son principe exploite, des notions théoriques issues de la théorie des graphes. De ce fait, nous consacrons la section suivante pour rappeler les notions élémentaires nécessaires pour appréhender l'application de ces algorithmes sur les images représentées sous forme de graphes et introduire par la même occasion les notations qui seront utilisées tout au long de ce manuscrit.

3. Notions communes et notations

3.1 Notion de graphe

Mathématiquement, un graphe noté G est défini comme une paire ($V(G)$, $E(G)$) où $V(G)$ est un ensemble fini d'éléments et $E(G)$ un ensemble composé de paires non ordonnées de $V(G)$:

$$E(G)= \{\{x,y\} \mid x \in V(G), y \in V(G) \text{ et } x \neq y\} \qquad (1.4)$$

Si $V(G) \neq \emptyset$, G est dit *non vide*.

Chaque élément de $V(G)$ est appelé *nœud* ou *sommet* de G (*Vertex* en Anglais) alors que chaque élément de $E(G)$ est appelé *arête* de G (*Edge* en Anglais).

3.1.1 Chemin dans un graphe

Soit G un graphe et $\pi=<x_0,\ldots x_k>$ une séquence ordonnée de sommets de G. π est un *chemin* de x_0 à x_k dans G si pour tout $i \in [1, k]$, x_i est adjacent à x_{i-1}. Dans ce cas, x_0 et x_k sont dits *liés* pour G. Si $k=0$, alors π est dit un chemin trivial dans G.

3.1.2 Adjacence de sommets et adjacence d'arêtes

Soit G (V, E, F) un graphe :

- Deux sommets x, y de $V(G)$ sont dits *adjacents* pour G s'il existe une arête qui les lie, i. e. $\exists\, u \in E \mid u=\{x, y\}$.
- Deux arêtes u et v de $E(G)$ sont dites *adjacentes* pour G si elles possèdent un sommet en commun, i. e. $u \cap v \neq \phi$.

3.1.3 Graphe connexe

Soit G un graphe. G est dit *connexe* si toute paire de sommets de G est liée pour G.

3.1.4 Sous-graphe, composante connexe et complémentaire de graphe

Soient X et H deux graphes. Si $V(Y) \subseteq V(X)$ et $E(Y) \subseteq E(X)$, alors Y est dit un *sous-graphe* de X. Cette inclusion est notée comme suit : $Y \subseteq X$.

Par ailleurs, Y est appelé une *composante connexe* de X si Y est un *sous-graphe connexe* de G et Y est maximal pour cette propriété. En d'autres termes, pour tout graphe connexe G, si $Y \subseteq G \subseteq X$ alors $G=Y$.

Soit $V' \subseteq V$ un ensemble de sommets de G. Le *complémentaire* de V' *noté* $\overline{V'}$ est l'ensemble $V \backslash V'$. De même, le *complémentaire* d'un ensemble d'arêtes E' noté $\overline{E'}$ est l'ensemble $E \backslash E'$.

3.1.5 Arête adjacente et sommet adjacent à un sous-graphe

Soient G ($V(G)$, $E(G)$) un graphe et X ($V(X)$, $E(X)$) un sous graphe de G :

- une arête $u = \{x, y\} \in E(G)$ est dite *adjacente* à X dans G si $\{x, y\} \cap V(X) \neq \varnothing$ et $\{x, y\} \notin E(X)$.
- le sommet parmi x et y qui n'appartient pas à $V(X)$ est dit *adjacent* à X dans G.

3.2 Graphes à Arêtes Valuées et Graphes à Sommets Pondérés

Il est possible d'associer aux sommets d'un graphe des poids. Un tel graphe est dit un *graphe à sommets pondérés*. Cette pondération se fait à l'aide d'une fonction de pondération.

De la même manière, il est possible d'associer aux arêtes d'un graphe des valeurs. Le graphe est dit dans ce cas *à arêtes valuées*. La valuation se fait à l'aide d'une fonction de valuation.

Un graphe à sommets pondérés (respectivement à arêtes valuées) auquel est adjointe une fonction *F* de pondération (respectivement de valuation) est noté *G* (*V*(*G*), *E*(*G*), *F*(*G*)).

Il existe dans la littérature différentes fonctions de pondération des sommets et de valuation des arêtes. Ces fonctions peuvent fournir des données qualitatives ou quantitatives et restent dépendantes du domaine d'application dans lequel le graphe est utilisé [32, 33, 34].

Dans le reste de ce manuscrit, et pour des raisons de simplification, nous adopterons l'appellation *valuation* aussi bien pour les sommets que pour les arêtes. Par ailleurs, soit $\mathbb{F}(E)$ (respectivement $\mathbb{F}(V)$) la famille de toutes les fonctions de valuation des arêtes (respectivement des sommets) de *G*, $F \in \mathbb{F}$ désignera toute fonction de valuation pouvant porter aussi bien sur les sommets que sur les arêtes de *G*. Ainsi, pour tout *x* arête dans *E*(*G*) (respectivement sommet dans *V*(*G*)), *F*(*x*) désignera la valuation de *x*.

Un graphe de sommets *V*, d'arêtes *E* et muni d'une fonction de valuation *F* (qui peut être appliquée sur les arêtes comme sur les sommets) sera ainsi noté *G* (*V*, *E*, *F*) au lieu de *G* (*V*(*G*), *E*(*G*), *F*(*G*)).

3.2.1 Plateau dans un graphe

Soit un graphe *G* (*V*, *E*, *F*), *X* un sous-graphe de *G* et *K* un ensemble d'éléments qui peut désigner aussi bien *E* ou *V* selon la valuation étudiée de *G*. *X* est dit un *plateau* de niveau *h* si :

- *X* est connexe;
- $\forall e \in K, F(e) = h$.

3.2.2 Extrema régionaux dans un graphe

Dans un graphe à sommets valués, un *extremum régional* pouvant être un *maximum*, ou respectivement un *minimum* régional, est un sous-ensemble de sommet(s) dont la valuation est supérieure, respectivement inférieure, à la valuation de tous les sommets adjacents à ce sous-ensemble.

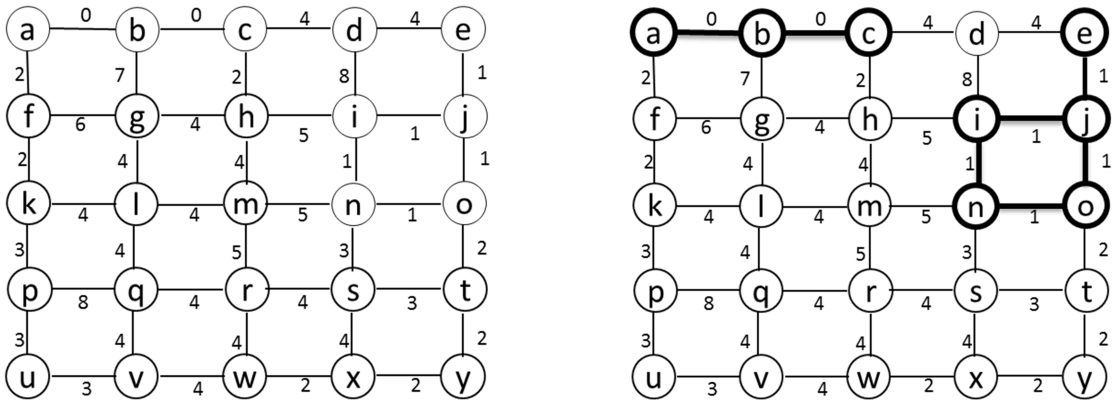
Cette notion d'*extrema* peut être également étudiée sur les arêtes. Dans ce cas, c'est la valuation des arêtes qui sera considérée au lieu de celle des sommets tout en conservant la même définition.

Soit *X* un sous-graphe de *G* (*V*, *E*, *F*), *N*(*X*) l'ensemble des éléments adjacents à *X* dans *G* (*N* pouvant être un ensemble de sommets ou d'arêtes selon le cadre d'application considéré) et *F* la fonction de valuation associée. *X* est dit un *maximum* (respectivement un *minimum*) régional de *G* de niveau *h* si :

- *X* est un plateau de niveau *h de G ;*
- $\forall e \in N,\ F(e) > h$ (respectivement $F(e) < h$).

L'union de tous les sous-graphes de *G* constituant des minima (respectivement des maxima) régionaux de *F* est notée *Min*(*F*) (respectivement *Max*(*F*)).

La figure 1.2 illustre cette notion pour le cas des minima régionaux dans un graphe à arêtes valuées : la figure 1.2 (a) représente un graphe à arêtes valuées tenant compte de la 4-adjacence. Les sous-graphes des minima du graphe (a) sont représentés en gras dans la figure 1.2 (b).



(a) (b)

Figure 1. 2 (a) un graphe à arêtes valuées et (b) en gras les sous-graphes des minima correspondants

3.2.3 Altitude d'un chemin dans un graphe

Soit *G* (*V, E, F*) un graphe valué et soit $\pi=<x_0,\ldots x_k>$ un chemin non trivial dans *G*. L'*altitude* de π notée $A_F(\pi)$ est donnée par la valuation maximale du chemin:

$$A_F(\pi) = \max\{F(x_i)\}\ avec\ k \in [0, k] \qquad (1.5)$$

Il est à noter que le chemin peut être spécifié par les sommets ou par les arêtes qui les lient. La fonction de valuation *F* est associée aux éléments considérés.

3.2.4 Altitude d'une connexion entre deux sous-graphes

Soit *G* (*V, E, F*) un graphe valué et soit $\pi = \langle x_0, \ldots x_k \rangle$ un chemin non trivial dans *G*. Soient *X* et *Y* deux sous-graphes de *G*, et $\Pi(X, Y)$ l'ensemble de tous les chemins de *X* à *Y* dans *G*.

L'altitude de la connexion (appelée également *valeur de connexion*) entre les sous-graphes *X* et *Y* dans *G* et pour la fonction de valuation *F*, notée $A_F(X, Y)$ est donnée par:

$$A_F(X,Y) = \min\{A_F(\pi)\ avec\ \pi\ \in \Pi(X,Y)\} \qquad (1.6)$$

3.2.5 Chemin descendant et chemin de plus grande pente

Soient *G* = (*V, E, F*) un graphe valué, *K* l'ensemble sur lequel est appliqué la valuation, $\pi = \langle x_0, \ldots, x_k \rangle$ un chemin dans *G* tel que pour tout $i \in [0, k]$, $x_i \in K$ et $N(x_i)$ l'ensemble des éléments adjacents à x_i.

π est un *chemin descendant* dans *G* pour *F* si pour tout entier $i \in [1, k]$, $F(x_{i-1}) \leq F(x_i)$.
π est un *chemin de plus grande pente* dans *G* pour *F* si, pour tout $i \in [1, k]$, $F(x_i) = min\ \{F(y) \mid y \in N(x_{i-1})\}$.

3.2.6 Notion d'extension et de coupures dans un graphe

Soient *G* (*V, E, F*) un graphe valué connexe, *K* l'ensemble sur lequel est appliquée la valuation et soient *X* et *Y* deux sous-graphes non-vides de *G*. *Y* est une *extension* de *X* (dans *G*) si $X \subseteq Y$ et si toute composante de *Y* contient exactement une composante de *X*.

Soit $C \subseteq E$. *C* est une *coupure de graphe* relative à *X* si $\bar{C}$ est une *extension* de *X* et si *C* est minimale pour cette propriété, i.e., $T \subseteq C$ et $\bar{T}$ est une extension de *X*, impliquent $T = C$.

4. Segmentation par Ligne de Partage des Eaux

La segmentation par Ligne de Partage des Eaux (LPE) consiste à considérer une image à niveaux de gris comme un relief topographique dans lequel l'altitude d'un pixel correspond à son niveau de gris de telle sorte que les bassins et vallées correspondent aux zones sombres (ayant les niveaux de gris les plus bas) de l'image alors que les montagnes et les lignes de crêtes correspondent aux zones claires (ayant les niveaux de gris les plus élevés).

Dans ce relief, un bassin versant, appelé aussi bassin d'attraction, est un territoire où toute goutte d'eau s'écoule vers son point le plus bas (qui correspond à un minimum régional de l'image). Cet écoulement se fait en suivant la pente la plus forte. La LPE désigne la limite géographique entre deux bassins versants. C'est l'ensemble des points à partir desquels une goutte d'eau peut s'écouler suivant des chemins différents amenant chacun à un minimum régional donné. La figure 1.3 illustre ces différentes notions. La segmentation par LPE consiste à calculer les bassins versants de ce relief (associés aux minima de l'image) et/ou les lignes qui les séparent (qui correspondent aux lignes de partage des eaux).

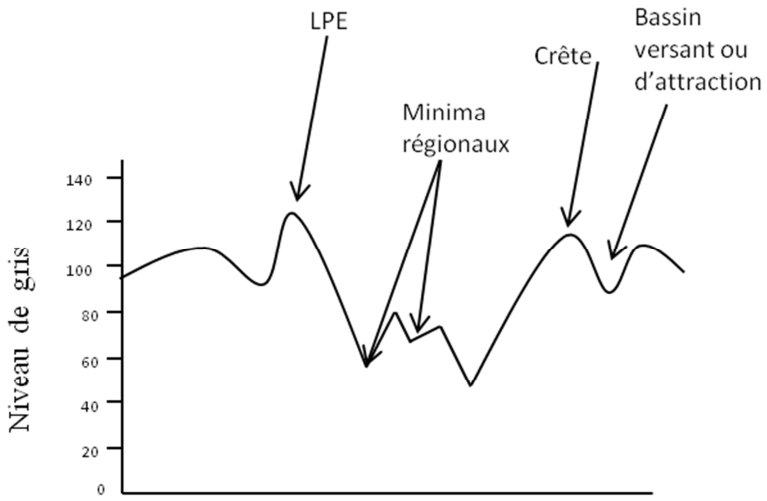


Figure 1. 3. Représentation d'une image en niveau de gris sous forme de relief topographique

5. Définition et classification des Lignes de Partage des Eaux

Depuis son apparition, plusieurs définitions ont été proposées dans la littérature pour calculer la LPE. Ces définitions peuvent être classées selon le phénomène physique qui les régit en trois principales catégories qui sont: la LPE par immersion, la LPE par ruissellement et la LPE topologique. Nous détaillons dans ce qui suit le principe général de chacune de ces LPE.

5.1 La LPE par immersion

La LPE par immersion nommée aussi LPE par inondation est la méthode par laquelle la LPE a été introduite en tant qu'outil de Morphologie Mathématique en 1979 [6]. Elle consiste à simuler une montée progressive du niveau d'eau à partir des minima du relief. La montée des eaux consiste à immerger ce relief dans de l'eau et ce en imaginant que ce relief est percé d'un trou dans chacun de ses minima régionaux. L'eau pénètre ainsi à travers les trous en

commençant par immerger les minima et remplir les bassins versants. Lorsque les eaux provenant de différents trous se rencontrent, des digues sont créées. Ces digues constituent la LPE.

5.2 La LPE par ruissellement

La notion de la LPE par ruissellement nommée aussi LPE par distance topographique a été introduite en 1994 par Fernand Meyer [11] (voir également [12, 21]). Elle vise principalement à déterminer les bassins versant qu'une LPE sépare. Chaque bassin versant étant associé à un minimum régional. Cette détermination est basée sur la simulation du phénomène de l'écoulement d'une goutte d'eau tombant sur un relief topographique. En effet une telle goutte suit un chemin descendant jusqu'à atteindre un minimum régional correspondant à un bassin versant donné. Les points du relief où une goutte peut être attirée par plus qu'un bassin versant correspondent ainsi à la LPE.

5.3 La LPE topologique

Les résultats fournis par les méthodes de segmentation par les LPE par immersion et par ruissellement présentent certaines limites surtout lorsque la segmentation est utilisée comme une étape préliminaire d'une phase d'analyse plus approfondie de l'image [4,15]. En effet ces deux méthodes produisent un résultat binaire dans lequel aucune information concernant l'altitude des régions associées aux bassins versants n'est disponible. Une telle information peut s'avérer utile pour certains post-traitements tels que la fusion des régions qui s'impose souvent du fait de la sur-segmentation que produit la LPE en sortie [7]. Dans de telles fusions, l'altitude des bassins versants et des crêtes qui les séparent est une information importante requise pour prendre la décision concernant les composantes à fusionner.

Pour remédier à ces limites, la LPE topologique appelée également LPE par ravinement, a été introduite par Michel Couprie et Gilles Bertrand en 1997 [4]. Elle a été ensuite développée dans d'autres travaux tels que [15, 18, 19]. Elle consiste à raviner le relief topographique en partant des minima régionaux [23] tout en préservant certaines propriétés topologiques entre les régions. Ainsi, les zones qui ne forment pas de col entre deux minima sont ravinées en diminuant leurs altitudes. Seules les cloisons dont l'altitude est celle du col séparant les deux minima sont conservées. Les crêtes qui persistent constituent la LPE topologique.

Contrairement aux LPE par immersion et par ruissellement, la LPE topologique ne produit pas un résultat binaire mais plutôt un résultat qui contient une information relative au contraste entre les différentes régions de l'image. Le contraste étant défini ici comme l'altitude minimale à laquelle on est obligé de monter pour passer d'une région à l'autre [20]. La séparation induite par la LPE topologique vérifie la propriété que pour chaque paire de minima séparés par une crête, les bassins versants correspondants sont séparés par une crête de même altitude.

6. Ligne de Partage des Eaux et sur-segmentation

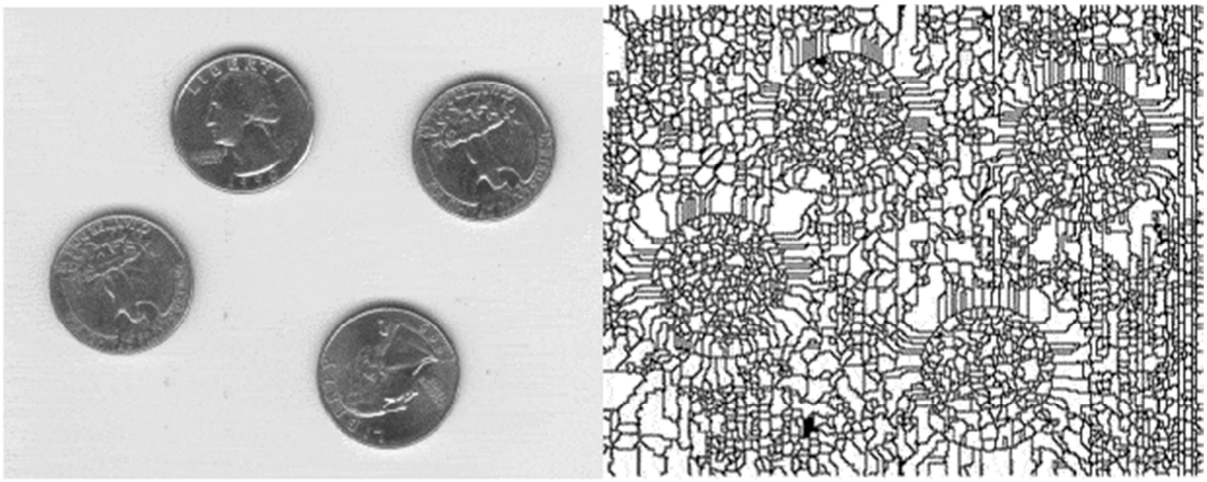
La LPE, dans ses différentes variantes, est une technique très utilisée en segmentation d'images en niveaux de gris vu la précision des frontières qu'elle produit et la distinction parfaite entre les régions [17, 21, 23]. Toutefois cette technique présente un inconvénient majeur qui est la sur-segmentation. Cette dernière est due au fait que les images contiennent généralement un grand nombre de minima et que chaque minimum génère un bassin versant. La figure 1.4 illustre ce phénomène. La figure 1.4 (a) représente une image originale qui visuellement apparaît homogène et ne représente pas trop de détails. La figure 1.4 (b) représente le résultat de l'application de la LPE sur le gradient (morphologique) de l'image de (a). L'image résultat obtenue est sur-segmentée. Comme nous l'avons mentionné, ceci est dû au grand nombre de minima que contiennent les gradients des images naturelles. Ceci peut provenir ou bien du système de prise de vue ou des variations locales non significatives des niveaux de gris ou encore de la texture des régions.

Il existe plusieurs méthodes qui permettent de remédier à ce problème ou du moins l'atténuer :

- Une première solution consiste à filtrer l'image avant d'appliquer la LPE. Le but de ce filtrage est de supprimer les minima non-significatifs [17, 90, 91, 92]. Les filtres les plus communément utilisés dans ce cadre sont le filtrage par diffusion anisotropique et le filtre alterné séquentiel [100, 101, 102].
- Une deuxième solution consiste à appliquer un post-traitement de fusion des régions produites par la LPE en se basant sur des critères de similarité [17, 93].
- Une troisième solution consiste à fixer le nombre de minima régionaux et donc le nombre de zones que l'on souhaite obtenir en appliquant la LPE. Dans ce cas, les minima dans des positions spécifiques sont considérés comme marqueurs, le reste des

minima est ignoré [74, 75]. Ces positions sont définies explicitement par l'utilisateur ou bien déterminées automatiquement entre autres par des opérateurs morphologiques. On parle dans ce cas de segmentation par LPE à base de marqueurs [74, 75].

Il est à noter qu'une approche commune en segmentation à base de LPE consiste à considérer le résultat obtenu en calculant une première LPE comme marqueur pour une seconde LPE. Ce processus itératif est appelé segmentation hiérarchique [17, 76].



(a) (b)

Figure 1. 4 Illustration de la sur-segmentation (a) image originale et (b) image résultat après application de la LPE par immersion sur le gradient morphologique de l'image (a) [99]

7. LPE, image et graphe

L'objectif de la segmentation est de détecter les contours des régions constituant l'image. De ce fait, la LPE n'est généralement pas appliquée directement sur le niveau de gris des pixels mais plutôt sur leurs gradients (gradient morphologique par exemple). En effet, en assimilant l'image à un relief topographique, le gradient produit de fortes valeurs et donc des hautes crêtes au niveau des contours entre les régions où l'altitude des pixels voisins (dans l'image) varient brusquement. Cette variation d'altitude devient au contraire minime dans les zones situées à l'intérieur des régions qui sont de nature homogènes. Ainsi, la LPE doit détecter ces crêtes pour en déduire les contours de l'image.

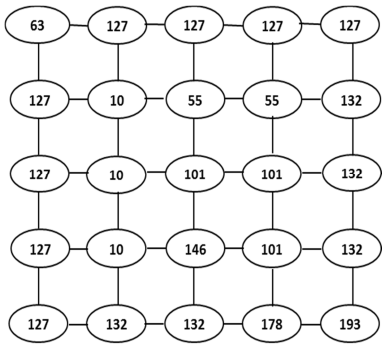
La majorité des algorithmes qui ont été proposés pour le calcul de la LPE se sont ainsi basés sur l'information intrinsèque aux pixels et ont utilisé naturellement comme domaine d'application des Graphes à Sommets Pondérés. En effet, la définition de voisinage entre les pixels d'une image s'apparente à celle d'adjacence entre les sommets d'un graphe. Ainsi, une image *I* associée à un voisinage est un graphe non-orienté *G* pondéré sur les sommets : chaque

sommet correspond à un pixel de l'image et sa pondération correspond à une grandeur liée au pixel. De tels graphes sont bien adaptés à la définition des plusieurs notions que requiert le calcul de la LPE telles que les notions de voisinage, d'adjacence, et de parcours des chemins, etc. Plus récemment, des travaux ont essayé d'adapter ces algorithmes et de les définir dans le cadre des Graphes à Arêtes Valuées. A cet égard, différentes fonctions de valuation des arêtes ont été utilisées qui dépendent du contexte de l'application dans laquelle est considérée la LPE. Ainsi si $u = \{x, y\}$ désigne une arête d'un graphe à arêtes valuées, parmi les valuations usuelles de u il est possible de citer : $F(u) = min\ \{F(x), F(y)\}$ ou également $F(u) = |F(x) - F(y)|$. Les valuations obtenues par cette dernière fonction représentent le gradient et permettent de calculer directement la LPE sur les arêtes du graphe.

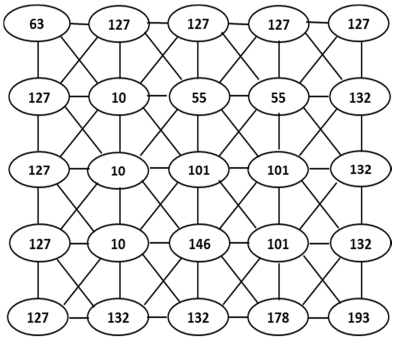
(a)

63	127	127	127	127
127	10	55	55	132
127	10	101	101	132
127	10	146	101	132
127	132	132	178	193

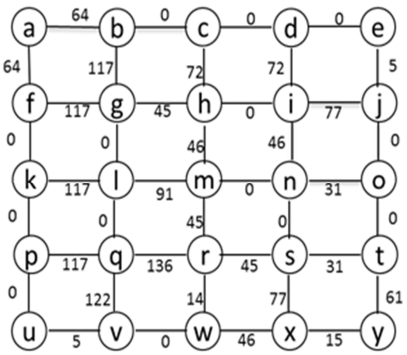
(b)



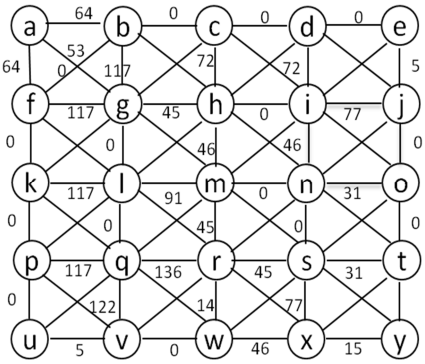
(c)



(d)



(e)



(f)

Figure 1. 5 (a) et (b) une image en niveau de gris (c) et (d) graphe à sommets pondérés correspondant à l'image (a) associée au 4-voisinage et au 8-voisinage (e) et (f) graphe à arêtes valuées correspondant à l'image (a) associée au 4-voisinage et au 8-voisinage

Un exemple de représentation d'une image bidimensionnelle en niveau de gris de taille 5×5 sous forme de graphe en tenant compte de différents voisinages est donné dans la figure 1.5. La figure 1.5 (a) (respectivement figure 1.5 (b)) représente l'image initiale (respectivement sa matrice correspondante). La figure 1.5 (c) (respectivement figure 1.5(d)) représente le graphe à sommets pondérés correspondant à l'image de (a) associé au 4-voisinage respectivement 8-voisinage. Dans ce cas, les sommets sont pondérés en leurs attribuant la valeur de niveau de gris des pixels associés. La figure 1.5 (e) (respectivement figure 1.5 (f)) représente le graphe à arêtes valuées correspondant à l'image de (a) associé au 4-voisinage respectivement 8-voisinage. Dans ce cas, les arêtes sont valuées en leurs attribuant la différence de niveau de gris entre deux pixels voisins.

Dans ce qui suit, les images seront représentées par des graphes. Nous empruntons de ce fait pour décrire leurs éléments la terminologie utilisée par les graphes. Ainsi, un pixel sera également désigné par le terme sommet, un niveau de gris par altitude, etc.

8. Algorithmes de calcul de la LPE

Dans le cas discret, plusieurs algorithmes ont été proposés dans la littérature pour calculer la LPE. Ces algorithmes consistent ou bien à calculer les bassins versants associés à chaque minimum régional de l'image, ou bien à calculer un ensemble de lignes de partage des eaux qui séparent ces bassins. Dans les deux cas, la majorité de ces algorithmes est basée sur l'information relative aux pixels et est ainsi définie dans le cadre des Graphes à Sommets Pondérés (GSP). Des travaux plus récents ont essayé d'adapter ces algorithmes et à les définir dans le cadre des Graphes à Arêtes Valuées (GAV). La motivation principale derrière cela est la vérification de certaines propriétés principalement théoriques [17, 22, 23] qui ne sont pas satisfaites dans le cadre des graphes à sommets pondérés. Les auteurs ont démontré que les résultats obtenus sont meilleurs notamment en ce qui concerne la qualité et la précision des contours des régions produites.

Dans ce qui suit, nous allons passer en revue une sélection des principaux algorithmes de calcul de la LPE. La présentation de ces algorithmes sera faite selon leur cadre d'application (GAV ou GSP). Nous commençons à cet égard par le cas des Graphes à Sommets Pondérés puis nous passons au cas des Graphes à Arêtes Valuées.

8.1 Algorithmes de calcul de la LPE dans le cadre des Graphes à Sommets Pondérés

La présentation des algorithmes dans la cadre des graphes à sommets pondérés est faite selon le principe physique qu'ils simulent. Nous traitons ainsi dans l'ordre la LPE par immersion, par ruissellement et la LPE topologique. Une variante de la LPE topologique nommée LPE par érosion est aussi présentée et un algorithme de calcul de cette variante est décrit.

8.1.1 Algorithmes de calcul de la LPE par immersion

Plusieurs algorithmes de calcul de la LPE par immersion existent dans la littérature. Les deux les plus connus parmi eux sont celui de Vincent et Soille [9] et celui de Fernand Meyer [10].

L'algorithme proposé par Vincent et Soille [9] (Algorithme 1) est basé sur le calcul des *zones d'influence géodésiques*. Cette notion, repose sur la définition de la *distance géodésique* entre deux pixels *x* et *y* dans un espace discret *A* (notée $d_A(x,y)$). Cette distance est définie comme étant la longueur du plus court chemin, totalement inclus dans *A*, qui lie *x* et *y*. Elle est donnée par:

$$d_A(x,y) = \min\{l(\Pi) \mid \Pi \; est \; l'ensemble \; des \; chemins \; entre \; x \; et \; y \; inclus \; dans \; A\} \quad (1.7)$$

Si *B* est un sous-ensemble de *A* alors la distance d'un pixel *x* au sous-ensemble *B* dans l'espace *A* est donnée par:

$$d_A(x,B) = min_{z \epsilon B}\big(d_A(x,z)\big) \quad (1.8)$$

Soit $B \subseteq A$ avec *B* composé de *k* composantes connexes B_i tel que $i=1\ldots k$. Dans ce cas, la *zone d'influence géodésique* de l'ensemble B_i dans *A* est donnée par:

$$iz_A(B_i) = \{p \epsilon A | \forall \; j \epsilon [1 \ldots k] \backslash \{i\}, \; d_A(p,B_i) < d_A(p,B_j)\} \quad (1.9)$$

L'algorithme de Vincent-Soille (Algorithme 1) consiste à propager les zones d'influence des pixels minima initialement étiquetés par des étiquettes différentes. Cet algorithme peut être résumé en deux principales étapes:

(i) étape de tri : initialement, les pixels sont triés dans l'ordre croissant selon leurs niveaux de gris;

(ii) étape d'immersion : cette phase se fait niveau par niveau en commençant par les minima qui sont labélisés initialement en leurs attribuant des étiquettes différentes. Elle consiste à propager les étiquettes uniques attribuées initialement aux minima, à leurs bassins versants associés. Cette propagation est réalisée par immersion itérative de l'image considérée comme relief topographique. Le parcours de chaque niveau se fait en utilisant l'approche de parcours en largeur d'abord et en utilisant une file d'attente FIFO. A un niveau *h* donné, tous les pixels d'altitude *h* qui ont des voisins déjà étiquetés à l'itération précédente sont insérés dans une file d'attente (FIFO). A partir de chacun de ces pixels des zones d'influence géodésiques sont propagées dans l'ensemble des pixels du niveau *h*.

Algorithme 1 : Vincent-Soille [9]

Data : Image I de taille $N * M$ et P le pas d'altitude (tolérance).
Result : Tableau d'entiers T de taille $N * M$ contenant les étiquettes des pixels tel que l'étiquette "Wshd" est assignée aux pixels appartenant à la LPE.

```
// Initialisation
Min:=le minimum global de l'image;
Max:=le maximum global de l'image;
L:= φ; // FIFO
// Étape 1
Trier les pixels par ordre croissant de leurs altitudes; // tous les pixels
 dont l'altitude est entre h et h + P sont considérés de même altitude ainsi
 les pixels sont traités par série.
// Étape 2
Chaque minimum régional de l'image possède une étiquette différente.
for (h = Min, h <= Max, h++) do
    foreach (pixel p d'altitude ∈ [h, h + E]) do
        if (p touche un bassin versant) then
            L:=L ∪ {p};
    foreach ( x ∈ L) do
        Parcourir les 4 voisins de x;
        Étiqueter x en tant que "Wshd" ou "voisin";
        Les pixels adjacents à x(+1) et de niveau [h,h+E] sont ajoutés à
         L;
    Propagation des étiquettes des bassins vers les pixels qui sont
     étiquetés comme "voisins";
```

Cet algorithme a été mis à jour en 2001 par Chen and Shi [70] dans l'objectif de résoudre quelques limites de l'algorithme original parmi lesquelles il est possible de citer :

- un étiquetage incorrect lorsqu'un point *p* est à la même distance d'au moins trois bassins versants adjacents (c'est-à-dire qu'il sera étiqueté comme un point appartenant à un bassin versant au lieu d'un point de la ligne qui les sépare);

- des calculs inutiles de la distance géodésique entre les points qui appartiennent à une zone étiquetée;
- la consommation de mémoire;
- l'absence d'informations (étiquettes) sur les bassins versants (niveau d'immersion) pendant l'exécution de l'algorithme.

L'algorithme modifié proposé introduit une troisième étape appelée «étape de jaillissement», qui, associée à l'étape d'immersion (étape (ii) de l'algorithme de Vincent-Soille original), permet la reconnaissance rapide des étiquettes des pixels et permet aussi de savoir le niveau d'immersion du bassin versant au cours de l'exécution de l'algorithme.

Par ailleurs, au début des années 90, Meyer [10] a proposé un algorithme, plus rapide que celui de Vincent-Soille, pour le calcul de la LPE par immersion pour les images en niveau de gris [9]. Dans cet algorithme (Algorithme 2), l'ensemble des pixels à partir desquels l'immersion commence est initialement prédéfini. Les pixels de ces ensembles sont appelés marqueurs et possèdent des étiquettes différentes. L'algorithme d'immersion consiste à étendre au maximum cet ensemble de minima en préservant les composantes connexes des minima. Pour cela, les pixels voisins de chaque région marquée sont insérés dans une file de priorité composée de *N* files d'attente de type FIFO tel que *N* désigne le nombre de niveau de gris dans l'image. Le niveau de priorité de chaque FIFO correspond au niveau de gris des pixels appartenant à cette file de priorité. L'algorithme consiste à parcourir les pixels de l'image par ordre croissant de leur niveau de gris. Ainsi, les pixels ayant le niveau de priorité le plus petit sont extraits de la file de priorité (plus précisément de la file d'attente ayant le niveau de priorité le plus petit). Si les voisins d'un pixel extrait ont tous la même étiquette, alors, cette dernière est attribuée à ce pixel. Sinon, tous les voisins non-étiquetés qui n'appartiennent pas à la file de priorité sont insérés dans cette dernière. Ce processus est répété jusqu'à vider la file de priorité. Enfin, les pixels non-étiquetés représentent la LPE.

L'algorithme de Meyer peut être résumé en trois principales étapes:

(i) Initialement, étiqueter les minima régionaux en attribuant des étiquettes distinctes aux différents minima;

(ii) A partir de cet ensemble des minima régionaux, marquer un sous-ensemble de minima (nommé S_m) à partir desquels l'immersion commence;

(iii) Insérer les pixels minima marqués dans une file de priorité *Q;*

(iv) extraire de Q les pixels ayant le niveau de priorité le plus petit, les stocker dans l'ensemble Ex_p et tester pour chaque pixel p de Ex_p, si ses voisins ont le même label *lab*, dans ce cas, *lab* est attribué à p et les voisins non encore étiquetés sont insérés dans Q;

(v) Répéter l'étape (iv) tant que la file de priorité Q n'est pas vide.

Après l'exécution de cet algorithme, la ligne de partage des eaux est donnée par le complément de l'ensemble des pixels étiquetés.

Algorithme 2 : F. Meyer [10]

Data : Image I et S_m l'ensemble des minima marqués à partir des quel l'inondation commence.
Result : image *lab*.

```
// Initialisation
foreach (pixel p ∈ S_m) do
    Insérer le pixel dans une file de priorité Q.
while (Q ≠ φ) do
    Exp:= extraire le pixel ayant le niveau de priorité le plus élevé;
    for (p ∈ Exp) do
        if (les voisins de p possèdent le même label label) then
            lab(p):=label;
            Q:= tout point voisins non encore marqué;
```

Il est à noter que cet algorithme n'est pas monotone. En effet, si un pixel *y* d'altitude *k* est extrait de la file *Q*, alors il est possible de trouver un pixel voisin *z* de *y* non encore étiqueté et qui a une altitude *h* inférieure à celle de *y (h<k)*. Le pixel *z* est inséré dans *Q* et il sera traité dans l'itération suivante. Ainsi, l'algorithme ne traite pas toujours les pixels dans un ordre croissant selon leur niveau de gris.

8.1.2 Algorithmes de calcul de la LPE par ruissellement

Le principe des algorithmes de calcul de la LPE par ruissellement (appelée aussi LPE par distance topographique ou LPE par écoulement) consiste à calculer la zone d'influence d'un minimum régional en utilisant une fonction de coût. Cette fonction est généralement basée sur le calcul d'une distance. Dans ce cadre, différentes distances peuvent être envisagées parmi lesquelles il est possible de citer la distance topographique [11]. Cette dernière exprime la distance entre deux points d'une image en fonction de la somme des distances entre les points intermédiaires et de l'inclinaison de la pente entre ces deux points. La zone d'influence d'un minimum régional est ainsi définie comme étant l'ensemble de points qui ont le coût le plus

faible par rapport à ce minimum que par rapport à tout autre minimum. Cette zone d'influence représente le bassin versant associé à ce minimum. Les points qui appartiennent à plus qu'un bassin versant forment la ligne de partage des eaux. Ce concept constitue la base de plusieurs algorithmes [11, 21, 31, 110].

Dans ce qui suit nous présentons dans un premier lieu les notions utiles pour définir la LPE par ruissellement et dans un second lieu les algorithmes qui permettent de la calculer.

Soient : *f* une fonction numérique de $\mathbb{Z}^n$ dans $\mathbb{Z}$ associée à une image en niveau de gris, *supp(f)* son support associé (c'est la partie du domaine où *f* n'est pas nulle) et *G* sa grille associée. A la grille *G* est associé un *graphe de voisinage* nommé *U* tel que *U* est un ensemble de $\mathbb{Z}^n \times \mathbb{Z}^n$ défini par :

$(x,y) \in U$ si x et y sont voisins (1.10)

Rappelons qu'un chemin π entre deux points *x* et *y* dans *G* est l'ensemble des n-uplets $\{x_1=x,\ldots,x_k=y\}$ de cardinal *k* tel que $\forall i \in [1,k\text{-}1]$, $(x_i , x_{i+1}) \in G$.

La longueur d'un chemin nommée $L(\pi)$ est donnée par :

$$L(\pi) = \sum_i dist(x_i, x_{i+1}) \quad (1.11)$$

La pente entre deux pixels *x* et *y* tel que $f(y) < f(x)$ est donnée par:

$$Slope(x,y) = \frac{f(x)-f(y)}{dist(x,y)} \quad (1.12)$$

L'ensemble des *voisins inférieurs* d'un pixel *x* pour lesquels la pente est maximale est nommé $\Gamma(x)$. La valeur maximale de la pente descendante nommée *pente inférieure* de la fonction *f* au point *x* est donnée par:

$$LS(x) = \max\left(\frac{f(x)-f(y)}{dist(x,y)}\right) \quad (1.13)$$

Il est à noter que Γ permet de définir un graphe orienté *V* comme sous-graphe du graphe d'adjacence *U*. *V* est nommé *graphe d'adjacence inférieur* :

$$(x,y) \in V \Leftrightarrow y \in \Gamma(x) \quad (1.14)$$

Le *coût de déplacement* sur une surface topographique d'un pixel à un autre (de $f(x_{i-1})$ à $f(x_i)$) est donné par:

$$Cost(x_{i-1}, x_i) = \begin{cases} LS(x_{i-1}) \times dist(x_{i-1}, x_i) si\ f(x_{i-1}) > f(x_i) \\ LS(x_i) \times dist(x_{i-1}, x_i) si\ f(x_{i-1}) < f(x_i) \\ \left(\frac{LS(x_{i-1}) + LS(x_i)}{2}\right) \times dist(x_{i-1}, x_i) si\ f(x_{i-1}) = f(x_i) \end{cases} \quad (1.15)$$

La distance *π-topographique* entre deux points *x* et *y* sur un chemin π est donnée par:

$$T_f^{\pi}(x, y) = \sum_{i>1} Cost(x_{i-1}, x_i) \quad (1.16)$$

Ainsi, la *distance topographique* entre deux pixels *x* et *y* est définie comme étant la distance *π-topographique* minimale entre ces deux pixels parmi tous les chemins π entre *x* et *y* dans *supp(f)*. Elle est donc donnée par:

$$T_f(x, y) = \min(T_f^{\pi}(x, y)) \quad (1.17)$$

Un bassin d'attraction associé à un minimum régional m_i , noté $CB(m_i)$ est défini comme étant l'ensemble des points $x \in supp(f)$ qui sont plus proches de m_i que de tout autre minimum régional selon la distance topographique définie par (1.17), i.e. $T_f(x, m_i) < T_f(x, m_j)$, $\forall\ i \neq j$.

Par conséquent, la LPE de *f* est l'ensemble des points de *supp(f)* qui n'appartiennent à aucun bassin d'attraction. Ainsi elle est donnée par:

$$Wshd(f) = supp(f) \cap [\cup_i (CB(m_i))] \quad (1.18)$$

Les nœuds minima du *graphe d'adjacence U* (défini par (1.10)) sont pondérés en leur attribuant un coût initial qui est leur altitude (niveau de gris).

Rappelons que la construction des bassins d'attraction consiste à trouver, pour chaque pixel, un chemin à coût minimal qui le lie à un minimum régional. Il est à noter que tous les pixels le long d'un chemin à coût minimal auront la même étiquette que le minimum régional appartenant à ce chemin.

Le problème de trouver le plus court chemin dans un graphe valué est résolu dans la littérature et différents algorithmes sont introduits. F. Meyer a exploité les algorithmes existants pour calculer la Ligne de Partage des Eaux par ruissellement. Ainsi, il a proposé trois algorithmes de calcul de ce type de LPE [11], qui ont été ensuite mis à jour par Roerdink et Meijster dans [21].

Le premier algorithme calcule une forêt des plus courts chemins (Shortest Path Forest) en utilisant l'algorithme de Moore [68] modifié. En effet, ce dernier est utilisé pour calculer le chemin le plus court d'un nœud au reste des nœuds dans un graphe. F. Meyer a proposé d'adapter cet algorithme et de dériver de ce dernier un algorithme de calcul des bassins d'attraction [11] et déduire ainsi les lignes de partage des eaux qui les séparent.

L'algorithme proposé par Meyer consiste à attribuer à chaque minimum régional une étiquette unique qui va être étendue à tous les pixels appartenant au bassin d'attraction associé à ce minimum. Cet algorithme peut être résumé en quatre principales étapes :

(vi) Phase d'initialisation : initialement, pour tout pixel appartenant à un plateau de minimum régional, la distance minimale est égale à l'altitude de ce dernier, i.e. $\forall\, x \in m_i$, $\pi(x)=f(x)$. Pour le reste des pixels z, $\pi(z)=\infty$. Les pixels internes des plateaux minima ne doivent pas être étendus et ils sont stockés dans un ensemble S. Les pixels appartenant aux limites des plateaux des minima régionaux doivent être étendus. Ces pixels ainsi que tous les pixels en dehors des minima régionaux appartiennent à un ensemble $\overline{S}$.

(vii) Sélectionner un pixel $x \in \bar{S}$ pour lequel $\pi(x) = min_{z\in\bar{S}}\ \pi(z)$ et le supprimer de l'ensemble $\bar{S}$.

(viii) Pour chaque voisin z de x dans l'ensemble $\bar{S}$ faire:
Si $\pi(z) < \pi(x) + Cost(x, z)$ alors $\pi(z)=\pi(x)+Cost(x, z)$ et $label(z)=label(x)$.

(ix) Répéter (iii) jusqu'à ce que $\bar{S}=\varnothing$.

Le second algorithme est appelé algorithme d'ascension de collines (*Hill Climbing Algorithm*) [21] exploite la fonction Γ (l'ensemble des voisins inférieurs d'un pixel pour lesquels la pente est maximale) définie ci-dessus. Cet algorithme (Algorithme 3) est basé sur l'idée que chaque pixel z est étiqueté par son voisin appartenant à $\Gamma(z)$: au cours de l'extension de x, tous ses voisins appartenant à l'inverse de la fonction Γ nommé Γ^{-1} et qui ne sont pas encore étiquetés sont étiquetés en leur attribuant l'étiquette de x. L'algorithme consiste donc à :

(i) Phase d'initialisation : initialement, pour tout pixel appartenant à un minimum régional, la distance minimale est égale à l'altitude de ce dernier. Les pixels internes des plateaux minima ne doivent pas être étendus et sont stockés dans un ensemble S. Les pixels appartenant aux limites des plateaux des minima régionaux

doivent être étendus. Ces pixels ainsi que tous les pixels en dehors des minima régionaux appartiennent à un ensemble $\bar{S}$.

(ii) Sélectionner un pixel $x \in \bar{S}$ pour lequel $f(x) = min_{z \in \bar{S}} f(z)$ et le supprimer de l'ensemble $\bar{S}$, i.e. $\bar{S} := \bar{S} \backslash \{x\}$.

(iii) Pour chaque pixel z de $\Gamma^{-1}(z) \cap \bar{S}$ qui n'est pas encore étiqueté faire: $label(z)=label(x)$.

(iv) Répéter l'étape (iii) jusqu'à ce que $\bar{S}=\emptyset$.

Il est à noter que l'étape (ii) consiste à sélectionner le pixel ayant la valeur $f(x)$ la plus petite. Dans le cas où plusieurs pixels ont la même valeur (cas d'un plateau de la fonction f), un problème de choix du pixel à sélectionner se présente. En effet, dans ces plateaux, aucun principe n'est appliqué pour guider la progression de l'écoulement de l'eau. Ceci entraine que la LPE produite n'est pas unique et plusieurs résultats peuvent être obtenus.

Pour remédier à cela, trois solutions peuvent être envisagées :

- La première solution consiste à utiliser une file d'attente hiérarchique pour stocker les pixels étiquetés et extraire le pixel ayant la valeur la plus petite [10]. Pour chaque niveau de gris h, une FIFO est utilisée pour stocker les pixels d'altitude h. Dans ce cas, les pixels frontières des plateaux sont ajoutés à la FIFO (associée au niveau de gris du plateau) lorsque leurs voisins inférieurs sont étendus. Ces pixels seront les premiers à être retirés de la FIFO et durant leur extension les pixels à distance 2 de la frontière sont ajoutés à la file.
- La deuxième solution consiste à modifier le *graphe d'adjacence inférieur V* définie par (1.14) à l'intérieur des plateaux, i.e. pour toute paire de pixels (x,y) pour laquelle $Cost(x,y)=0$. Pour de telles paires de pixels, un arc orienté est créé de x à y si la distance géodésique de x à la frontière inférieure du plateau est supérieure à celle de y à la même frontière. Un graphe nommé *graphe complété* (*completed graph*) est ainsi construit.
- La troisième solution proposée pour résoudre le problème de calcul de la LPE sur les plateaux, conduit à modifier l'algorithme de calcul de la LPE par ascension de collines. L'algorithme modifié traite en entrée une image dans laquelle il n'existe aucun plateau non minimum, i.e. pour chaque pixel non minimum x, il existe un voisin de x qui a une altitude inférieure. Une telle image est appelée "*Lower Completed*

Image" notée dans le reste du manuscrit LCI. Le principe de l'algorithme reste le même sauf que l'image en entrée est prétraitée et est transformée en une LCI.

Algorithme 3 : Ascension de colline [21]

Data : I une image LCI.
Result : lab une image étiquetée.

// Initialisation
L'image lab est initialisée comme suit:
(i) les pixels non minima sont étiquetés comme $MASK$;
//Rappelons qu'il n'existe pas de plateau non minima dans l'image I.
(ii) les pixels minima ont des étiquettes distinctes.
// les pixels intérieurs des minima sont exclus du traitement
$\overline{S}$:= { l'ensemble des pixel p tel que $\exists$ q in $N(p)$ avec $I(p) = I(q)$};
// $N(p)$ désigne l'ensemble des voisins de p.
while $(\overline{S} \neq \phi)$ **do**
    Sélectionner un point $p \in \overline{S}$ ayant le niveau de gris le plus petit;
    $\overline{S}$:=$\overline{S} \setminus \{p\}$;
    //L'ensemble des pixels q_i voisin d'un pixel p tel que p et q_i sont liés
    //par un chemin descendant de plus grande pente forment un ensemble
    //η. Ainsi, l'ensemble $\eta^{-1}(p)$ désigne l'ensemble des pixels q_i
    //pour les quels $p \in \eta(p)$.
    for $(q_i \in \eta^{-1}(p) \cap \overline{S}\ )$ **do**
        if $(\ lab(q_i)=MASK;)$ **then**
            $lab(q_i)$:=$lab(p)$;
        else if $(\ (lab(q_i) \neq WSHD)\ et\ (lab(q_i) \neq lab(p))\ )$ **then**
            $lab(q_i)$:=$WSHD$; //q_i est étiqueté comme étant un point appartenant à la LPE.

Le troisième algorithme proposé par Meyer est une modification de l'algorithme de Berge pour les Forêts des plus courts chemins (SPF) [69]. En effet, comme pour l'algorithme de Moore, l'algorithme de Berge est utilisé dans la littérature pour calculer la distance d'un nœud aux restes des nœuds d'un graphe valué. Cependant le principe de l'algorithme de Berge diffère de celui de Moore en ce qui concerne le calcul du plus court chemin. Il exploite la propriété suivante:

Soit $\pi(i)$ le chemin le plus court entre 1 et *i* et (*i*, *j*) l'arc de longueur l_{ij} liant les nœuds *i* et *j*. Le plus court chemin de 1 à *j* qui passe par le nœud *i*, vérifie:

$$\pi(j) \leq \pi(i) + l_{ij} \qquad (1.19)$$

L'algorithme de Berge est donc le suivant :

(i) Initialisation : $\pi(1)=0$ et $\forall i \neq 1$, $\pi(i)=0$.

(ii) Trouver un arc (i,j) pour lequel $\pi(j)-\pi(i) > l_{ij}$.

(iii) $\pi(j)= \pi(i)+ l_{ij}$.

(iv) Répéter les étapes (ii) et (iii) tant qu'il existe un arc qui vérifie la condition (ii).

Ainsi, le troisième algorithme proposé par Meyer est le suivant:

(i) Initialisation : pour tout pixel x qui n'est pas un minimum régional, $lab(x)=\infty$.

(ii) Balayer l'image ligne par ligne du haut vers le bas (*raster scanning*) puis du bas vers le haut (*inverse raster scanning*) et appliquer à chaque pixel les traitements suivants :
Pour chaque voisin y de x (y est le pixel à traiter après avoir terminé le traitement de x), si $lab(y) \neq lab(x)$ et $y \in \Gamma(x)$ alors $lab(y)=lab(x)$.

(iii) Répéter l'étape (ii) jusqu'à la stabilité, i.e. ne plus avoir de changements durant le balayage dans les deux sens.

8.1.3 Algorithme de calcul de la LPE topologique

Dans [4], un algorithme de calcul de la LPE topologique basé sur le calcul d'arbre des composantes a été introduit. Cet algorithme consiste à déformer le relief topographique progressivement jusqu'à obtenir une structure fine qui constitue la LPE. La déformation du relief est réalisée en préservant la notion de contraste. Avant de présenter cet algorithme, nous introduisons quelques notions nécessaires pour appréhender son cadre d'application. Ces notions sont celles de *section supérieure* et *inférieure* d'une fonction dans les graphes valués ainsi que la notion de *point W-destructible* [20]. Dans ce cadre, soient $G = (V, E, F)$ un graphe à sommets pondérés et $k \in \mathbb{Z}_+$:

- La *section supérieure* de F au niveau k notée F_k, est donnée par l'ensemble des sommets de V dont le niveau est supérieur ou égal à k.

- La *section inférieure* de F au niveau k, notée $\overline{F_k}$, est donnée par le complémentaire de F_k.

- Un *point simple* pour F est un élément e de V d'altitude k qui est adjacent à une seule composante connexe de $\overline{F_k}$.

- Un *point destructible* pour F est un élément e de V d'altitude k qui représente un point simple pour $\overline{F_k}$.

- Un point *W-destructible* (*Watershed-destructible*) pour *F* est un point destructible dont la valeur peut être abaissée de 1 sans changer le nombre de composantes connexes de $\overline{F_k}$.

Une LPE topologique de *F* (nommée $W \in \mathcal{F}(E)$) peut être dérivée de *F* en abaissant itérativement l'altitude des *points W-destructibles* jusqu'à stabilité (c'est-à-dire jusqu'à ce que tous les points de *V* soient non destructibles pour *W*) [20]. Les bassins d'attraction de la LPE sont les minima de *W*, et les lignes de la LPE topologique représentent les non-minima de *W*. En se basant sur cette définition, un algorithme quasi-linéaire de calcul de la LPE topologique a été proposé dans [4]. Cet algorithme, appelé algorithme de calcul de noyau homotopique supérieur (*Upper Homotopic Kernel Algorithm*), nécessite le calcul d'une structure appelée arbre des composantes [26]. Il consiste à élever l'altitude de certains points de l'image de telle sorte que la connectivité de chaque section inférieure soit conservée. Il peut être résumé en trois principales étapes :

(i) Construire l'arbre des composantes *T;*
(ii) sélectionner itérativement un point *W-destructible*, l'élever grâce à l'information contenue dans (*T, F*) et mettre à jour la fonction *F*;
(iii) répéter l'étape (ii) jusqu'à atteindre la stabilité.

A partir de cet algorithme La LPE topologique d'une fonction *F* est donnée par le complément du noyau homotopique supérieur de $\bar{F}$.

Plus tard en 2005, M. Couprie et al [18] ont proposé un autre algorithme linéaire de calcul de la LPE topologique en se basant sur la même définition de point destructible. En effet, l'algorithme présenté précédemment est un algorithme naïf de calcul de la LPE topologique dans lequel un point W-destructible abaissé à sa valeur minimale peut redevenir W-destructible de nouveau suite à l'abaissement de ses voisins. Pour obtenir une complexité linéaire, il faut éviter de sélectionner le même point plusieurs fois durant l'exécution de l'algorithme. Pour ce faire, des critères de sélection des points à abaisser ont été imposés par les auteurs pour garantir qu'un point abaissé une fois ne sera plus un point W-destructible au cours de l'exécution de l'algorithme.

Le premier critère concerne les points qui peuvent être abaissés à la valeur de leur voisin appartenant à un minimum régional : les *M-points*. L'action est nommée *M-abaissement* (*M-lowering* en Anglais). Dans certains cas, ce critère peut ne pas produire une LPE topologique (même après avoir abaissé itérativement les M-points jusqu'à la stabilité, il reste des points

W-destructibles qu'il faut traiter pour obtenir la LPE topologique de *F*) ou une LPE épaisse. Ainsi, les auteurs dans [18] ont proposé un deuxième critère de sélection des points W-destructibles à abaisser pour éviter la sélection multiple d'un même point. L'idée est d'attribuer la priorité la plus élevée au point W-destructible qui peut être abaissé jusqu'à sa valeur la plus petite possible. Les auteurs ont prouvé qu'un algorithme basé sur une telle stratégie ne sélectionne jamais le même point plus qu'une fois. Une file de priorité peut être utilisée pour sélectionner les points W-destructibles dans l'ordre approprié. En se basant sur cette idée, un algorithme linéaire de calcul de la LPE topologique a été proposé (Algorithme 4).

Algorithme 4 : Couprie et al [18]

Data : (V, E, F) un graphe à sommets pondérés, ψ le mapping de composantes et $C(\overline{F})$: un arbre de composantes de $\overline{F}$.
Result : une LPE topologique F

```
// Initialisation
for (k = K_min, k++, k < k_max) do
    L_k := φ;
foreach (p ∈ V) do
    [i, c]:=W-destructible(F,p,C(F̄),ψ);
    if (i ≠ ∞) then
        L_{i-1}:= L_{i-1} ∪ {p};
        (K(p):=i − 1;
        H(p):=pointer sur [i, c];
for (k = K_min, k++, k < k_max) do
    while (∃ p ∈ L_k) do
        L_k:= L_k \ {p};
        if (K(p) = k) then
            F(p) := k;
            ψ (p):= H(p);
            foreach (q ∈ V voisin de p tel que k < F(q)) do
                [i, c]:=W-destructible(F,p,C(F̄),ψ);
                if (i = ∞) then
                    K(q):=∞;
                    else if (K(q) ≠ (i − 1)) then
                        L_{i-1}:=L_{i-1} ∪ {q};
                        K(q):=i − 1;
                        H(q):=pointer sur [i, c];
```

Function W-destructible [18]($F, C(\overline{F}), \Psi$):
V:=l'ensemble des éléments de $C(\overline{F})$ pointés par $\psi(q)$ pour tout $q \in N(p)$;
if $(V = \phi)$ **then**
  retourner $([\infty, \phi])$;
$[k_m, c_m]$=HighestFork$(C(\overline{F}), V)$;
if $([k_m, c_m] == [\infty, \phi])$ **then**
  retourner $(Min(V))$;
if $(k_m <= F(p))$ **then**
  retourner $([k_m, c_m])$;
else
  retourner $([\infty, \phi])$;

Dans l'algorithme 4, Ψ désigne la fonction des composantes qui associe à chaque point *p*, un pointeur $\psi(p)$ vers l'élément $[\bar{F}(p), C(p)]$ de l'arbre de composantes $C(\bar{F})$. De plus, cet algorithme fait recours à la fonction W-destructible qui retourne si un point est W-destructible ou non. Dans le cas où un point *p* est W-destructible, la fonction retourne aussi la *composante d'altitude la plus basse* (*the lowest component*) à laquelle le point *p* peut être ajouté. Sinon, la valeur $[\infty, \emptyset]$ est retournée.

8.1.4 Algorithme de calcul de la LPE par érosion

La LPE par érosion a été introduite en 2006 dans [22,7] comme variante de la LPE topologique dans le cadre des *graphes de fusion parfaits*. Dans ce type de graphe, deux régions séparées par un ensemble de sommets formant une frontière peuvent être fusionnées en supprimant un sommet de cet ensemble de sommets frontières sans ajouter une troisième région. En effet, dans les graphes de fusion parfaits, tout sommet d'un ensemble frontière est strictement adjacent à deux régions. Ainsi, cette catégorie de graphes présente un cadre idéal à la fusion de graphe puisque qu'il n'y a plus à vérifier si le sommet à retirer est adjacent à une troisième région [7, 22].

Cette variante de la LPE peut être définie comme suit : pour une image représentée sous forme d'un relief topographique, les zones situées autour des minima régionaux sont érodées progressivement, en commençant par les plus basses, et ce en diminuant leurs altitudes jusqu'à celle du minimum régional. L'ensemble des séparations entre deux minima, de hauteur le col le plus bas reliant ces deux minima, représente la LPE par érosion. Tout comme le processus de ravinement (défini dans la section 5.3 de ce chapitre), l'érosion étend les minima régionaux, elle n'en crée pas de nouveaux et elle n'en diminue pas l'altitude. En

revanche, à la différence d'un ravinement, une érosion ne peut diminuer l'altitude que des éléments adjacents à un unique minimum régional.

Il est à rappeler que pour calculer la LPE par ravinement, les auteurs dans [18] ont proposé un algorithme qui procède par abaissement itérative de la valeur des points W-destructibles de 1 (section 8.1.3 de ce chapitre). Cette transformation est appelée W-amincissement. L'étude d'une famille des W-amincissements qui procède par abaissement des altitudes de certains points particuliers que les auteurs appellent points *M-falaise* a conduit à définir l'algorithme de calcul de la LPE par érosion dans le cadre des graphes de fusion parfaits. Il est à noter qu'un point est dit un point *falaise* pour une fonction *F* s'il appartient à $\overline{Min\,(F)}$ et il n'est adjacent qu'à un seul minimum de *F*. Un point *p* est dit *M-falaise* (ou *Minimum-falaise*) pour *F* si :

- *p* est un point falaise pour *F*;
- il n'existe aucun autre point appartenant à $\overline{Min\,(F)}$ qui est adjacent à un unique minimum et dont l'altitude est strictement inférieure à celle de *p*.

L'algorithme de calcul de la LPE par érosion est donc l'algorithme 5:

Algorithme 5 : LPE par érosion [17]

Data : G un graphe de fusion parfait et une fonction $F \in \mathcal{F}(V)$
Result : F une LPE par érosion de la fonction d'entrée

```
// Initialisation
L:=φ;
K:=φ;
Attribuer des étiquettes distinctes à tous les minima de F et marquer
  les points de Min(F) avec les étiquettes correspondantes;
foreach ( p ∈ V ) do
    if ( p ∈ Min(F) ) then
        K := K ∪ {p};
    else if ( p est adjacent à Min(F) ) then
        L := L ∪ {p};
        K := K ∪ {p};
while ( L ≠ φ ) do
    p:=un élément de L d'altitude minimale pour F;
    L := L \ {p};
    if ( p est adjacent à exactement un minimum de F ) then
        Attribuer à F[p] l'altitude du seul minimum de F adjacent à p;
        Marquer p avec l'étiquette correspondante;
        foreach ( q ∈ N(p) ∩ K̄ ) do
            L := L ∪ {q};
            K := K ∪ {q};
```

L'algorithme 5 [17] prend comme entrée un graphe de fusion parfait *G* et une fonction *F* et produit une LPE par érosion de la fonction d'entrée. A chaque itération de la boucle principale (ligne 11), l'ensemble *L* contient les points adjacents à un minimum de la fonction *F*. Le traitement des points appartenant à l'ensemble *L* consiste à leur attribuer l'altitude du seul minimum de *F* qui lui est adjacent (ligne 15) et leur marquer par l'étiquette correspondante (ligne 16). Lorsque l'ensemble *L* est vide, l'algorithme se termine et il n'y a plus de points adjacents à un unique minimum de *F*. A l'issue de l'algorithme 5, la fonction *F* obtenue est une LPE par érosion de la fonction d'entrée.

Il est à noter que cet algorithme est monotone. En effet, lorsqu'un point *p* d'altitude *k* est traité, tous les points insérés ultérieurement dans *L* ont une altitude supérieure ou égale à *k*. L'ensemble *L* peut donc être implémenté par une file de priorité monotone qui permet de traiter les points par altitude croissante. Ceci permet également d'éviter la réinsertion d'un même point dans l'ensemble *L*.

8.2 Algorithmes de la LPE dans le cadre des Graphes à Arêtes Valuées

Il a été mis en évidence dans [17, 23] que les Graphes à Arêtes Valuées représentent un cadre convenable et particulièrement adapté à la définition et au calcul de la LPE. Dans ce cadre, une variante de la LPE nommée *LPE d'arêtes* ou *coupures par LPE* a été introduite [17]. Cette variante de la LPE peut être définie de manières équivalentes par ses bassins d'attraction (à travers une propriété de plus grande pente) ou par les lignes qui séparent ces bassins d'attraction (à travers le principe de la goutte d'eau). Les algorithmes séquentiels de calcul de cette variante sont présentés dans ce qui suit.

8.2.1 Algorithme de calcul de la LPE d'arêtes

Dans [8, 17, 30], J. Cousty et al ont proposé deux algorithmes linéaires qui permettent de calculer les coupures par LPE (ou LPE d'arêtes) : l'algorithme de calcul de noyau par M-bord et l'algorithme de coupures par flux.

Le premier algorithme est basé sur une transformation nommée *amincissement par bord* qui consiste à abaisser jusqu'à l'idempotence la valeur de certaines arêtes du graphe de départ. Les minima de la fonction transformée induisent ce que les auteurs appellent une *coupure par M-bord* qui constitue la LPE d'arêtes.

Le deuxième algorithme est basé sur l'extraction itérative des flux. Il permet de calculer une partition par flux à partir de laquelle est déduite ce que les auteurs appellent une *coupure par flux* qui constitue la LPE d'arêtes.

Les graphes considérés par les auteurs dans ce cadre sont des Graphes à Arêtes Valuées (GAV) munis d'une fonction de valuation *F* appliquée sur les arêtes. Cette fonction attribue à une arête $u=\{x, y\}$ une valeur indiquant la dissimilarité entre les deux sommets qui la composent. Ainsi, pour un graphe $G= (V, E, F)$ la fonction *F* est donnée par:

$$\forall\, u = \{x, y\} \in E,\ F(u) = |I(x) - I(y)| \qquad (1.20)$$

où $I(x)$ (respectivement $I(y)$) représente le niveau de gris du pixel x (respectivement (y)).

Avec cette valuation, les contours saillants de l'image sont localisés sur les arêtes les plus élevées.

La valuation des sommets du graphe est donnée par une fonction *VF* qui pour chaque sommet assigne le minimum des valuations des arêtes auxquelles il appartient. La fonction *VF* est ainsi donnée par :

$$\forall\, x \in V, VF(x) = \min\{F(u_i)\ telque\ u_i \in E\ et\ u_i \cap x \neq \emptyset\} \qquad (1.21)$$

Dans un tel graphe *G*, un chemin $\Pi = \langle x_0, \ldots, x_k \rangle$ est dit *chemin de plus grande pente* pour *F* si, pour tout $i \in [1, k]$, $F(\{x_{i-1}, x_i\}) = F(x_{i-1})$. Il est dit *chemin descendant* (pour *F*) si pour tout $i \in [1, k-1]$, $F(\{x_{i-1}, x_i\}) \geq F(\{x_i, x_{i+1}\})$.

Pour les graphes ainsi définis, J. Cousty et al distinguent différents types d'arêtes qui peuvent être classifiées en se basant sur des propriétés purement locales. Dans ce contexte :

- Une arête *u* est dite *localement séparante* (pour *F*) si $F(u) > max\,(VF(x); VF(y))$.
- Une arête *u* est dite *arête de bord* (pour *F*) si $F(u) = max\,(VF(x); VF(y))$ et $F(u) > min\,(VF(x); VF(y))$.
- une arête *u* est dite une arête *minimum-border* ou *M-border* (pour *F*), si *u* est une *arête de bord* pour *F* et si exactement un sommet de $u \in Min$ où $Min = (V_{min}, E_{min})$ désigne le graphe dont les ensembles de sommets et d'arêtes sont, respectivement, l'union des ensembles de sommets et ensembles d'arêtes de tous les minima de *F* et *VF*.
- Une arête *u* est dite une *arête interne* (pour *F*) si $VF(x) = VF(y) = F(u)$.

La figure 1.6 illustre les différents types d'arêtes dans un *GAV*.

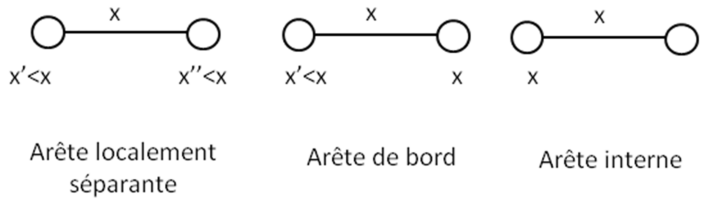


Figure 1. 6 Types d'arêtes dans un Graphe à Arêtes Valuées

8.2.1.1 Algorithme de calcul de noyau par M-bord

Dans [30], Cousty et al ont proposé une transformation d'amincissement qui consiste à abaisser les valeurs de certaines arêtes, jusqu'à l'idempotence. La fonction *d'abaissement* utilisée dans ce contexte est définie comme suit :

Soit $u \in E$. *L'abaissement* de F en u est la fonction F' telle que:

- $F'(u) = min_{x \in u} \{VF(x)\}$;
- $F'(v) = F(v)$ pour tout arête $v \in E \setminus \{u\}$.

En se basant sur cette définition, deux notions importantes pour le calcul de la LPE d'arêtes ont été introduites en l'occurrence la notion d'*amincissement par bord* et celle de *coupure par bord*. Dans ce cadre, *soit* $H \in \mathcal{F}(E)$. H est dit *amincissement par bord* de F si:

(i) $H=F$ ou si

(ii) il existe $I \in \mathcal{F}(E)$ un *amincissement par bord* de F tel que H est l'abaissement de I en une arête de bord pour I.

S'il n'existe aucune arête de bord pour H, H est dit *un noyau par bord*. Si H est un amincissement par bord de F et que c'est un noyau par bord, alors H est un *noyau par bord de* F.

Si H est un noyau par bord de F, toute coupure relative au sous-graphe *Min(H)* (voir section 3.2.2 de ce chapitre) est appelée *coupure par bord* de F.

L'algorithme qui permet de calculer un noyau par M-bord d'une fonction d'entrée, et par conséquent de calculer une LPE d'arêtes consiste à abaisser les valeurs des arêtes de bord adjacentes aux minima et répéter ce traitement jusqu'à ne plus avoir d'arêtes M-bord. Il est à

noter que dans le cas des arêtes M-bord, l'abaissement est réalisé au plus une seule fois. L'algorithme consiste donc à:

i) Détecter les arêtes sortantes du sous-graphe des minima et les stocker dans un ensemble *L*;
ii) Parmi les arêtes stockées dans *L*, détecter une arête M-bord et l'abaisser;
iii) Mettre à jour le sous-graphe des minima en lui ajoutant l'arête abaissée;
iv) Ajouter les nouvelles arêtes sortantes à l'ensemble *L*;
v) Répéter les étapes (ii), (iii), et (iv) jusqu'à ne plus avoir de nouvelles arêtes sortantes.

Une fois le noyau par M-bord de la fonction d'entrée est obtenu, il est possible de déduire la coupure par M-bord de *F*. Cette dernière constitue la LPE d'arêtes à calculer.

Il est à noter que cet algorithme ne requiert ni tri ni file de priorité. Par ailleurs, la propriété d'arête M-bord peut être testée localement ce qui rend cet algorithme adapté à la parallélisation. Nous reviendrons avec plus de détails sur ce point dans le chapitre 3 dans lequel nous présenterons notre contribution à cet égard.

8.2.1.2 Algorithme à base d'extraction de flux

Un deuxième algorithme de calcul de la LPE d'arêtes appelé algorithme de calcul de coupures par flux a été proposé dans [8]. Avant de présenter ce dernier, nous commençons d'abord par introduire quelques notions sur lesquels il est basé. Dans ce cadre, étant donné un graphe G (V, E, F), soit $L \subseteq V$. L est dit un *flux* de G si, pour toute paire de points x et y de L, il existe, dans L, soit un chemin de x à y soit de y à x qui est un chemin de plus grande pente pour F. x est dit une *source* (respectivement *puits*) de L si l'altitude de x est supérieure (respectivement inférieure) ou égale à l'altitude de tout autre sommet y de L.

Par ailleurs, soient L_1 et L_2 deux flux distincts (i.e., $L_1 \cap L_2 = \emptyset$;) et soit $L = L_1 \cup L_2$. On dit que L_1 est *sous* L_2, et on note $L_1 \prec L_2$, s'il existe une source x de L_1 et un puits y de L_2, tels qu'il existe, dans L, un chemin de plus grande pente de y à x. Il est à noter que si $L_1 \prec L_2$, alors L est également un flux. L est dit un *inf-flux* noté $\prec$*-flux* s'il n'existe aucun flux sous L.

Soient £ une famille de flux et $M_1, \ldots, M_n$ les minima de F. L'application Ψ de V dans $\{1,\ldots,n\}$ est dite une *partition par flux* de F si elle associe à chaque sommet x de V, l'index (étiquette) i tel que M_i est le seul minimum de F inclus dans un $\prec$-flux de £ qui contient x.

Si Ψ est une partition par flux de F, alors la *coupure par flux* de F est donnée par l'ensemble $C = \{\{x, y\} \in E \mid \Psi(x) \neq \Psi(y)\}$.

L'ensemble d'arêtes qui constitue la coupure par flux de F représente la LPE d'arêtes de F.

Algorithme 6 : Coupures par flux [8]

```
   Data : (V, E, F) un graphe à arête valuées
   Result : Ψ une partition par flux de F.
1  // Initialisation
2  foreach ( x ∈ V ) do
3  |  Ψ(x) := NO_LABEL;
4  nb_labs:=0;  // Le nombre de minima déjà trouvés
5  foreach ( x ∈ V tel que Ψ(x) = NO_LABEL) do
6  |  [L, lab]:= Flux(V, E, F, Ψ, x);
7  |  if (lab = -1 )  // L est un ≺ − flux then
8  |  |  nb_labs + +;
9  |  |  foreach ( y ∈ L ) do
10 |  |  |  Ψ(y) := nb_labs;
11 |  else
12 |  |  foreach (y ∈ L) do
13 |  |  |  Ψ(y) := lab;
```

L'algorithme 6 repose sur l'extraction de flux. Pour cela, il utilise la fonction *flux* qui permet d'extraire un flux en explorant les chemins de plus grande pente. L'algorithme assigne une étiquette à chaque sommet du graphe. Dans ce but, pour chaque point $x \in V$, il extrait un flux L, composé de points non encore étiquetés et dont la source est x (ligne 4). Si L est un $\prec$-flux (ligne 5), une nouvelle étiquette est assignée à tous les points de L. Sinon (ligne 8), il existe un $\prec$-flux $L1$ sous L qui est déjà étiqueté. Dans ce cas les éléments de L reçoivent l'étiquette de L_1 (ligne 9). La fonction *Flux*, invoquée à la ligne 4, permet d'extraire L. Intuitivement, elle explore les chemins de plus grande pente, en mélangeant des itérations en largeur et en profondeur. Les invariants principaux de cette fonction sont :

i) l'ensemble L est, à chaque itération, un flux;
ii) l'ensemble L_0 regroupe tous les puits de L non encore explorés.

La fonction s'arrête à la ligne 17, lorsque tous les puits de L ont été explorés ou, à la ligne 9, si un point z déjà étiqueté, est découvert lors de l'exploration : Dans le premier cas, le flux L retourné, est un $\prec$-flux. Dans le second cas, l'étiquette *lab* du sommet z est également retournée. Il est à noter que dans ce cas, il existe un puits y de L tel que $<y, z>$ est un chemin

de plus grande pente. Pour cette raison, il est garanti qu'il existe un $\prec$-flux L_1, sous L, qui est inclus dans l'ensemble des sommets étiquetés *lab*. En conclusion, à l'issue de l'algorithme 6, Ψ est une *partition par flux* de F à partir de laquelle une coupure par flux de F qui constitue la LPE d'arêtes de F est déduite.

Tout comme l'algorithme de calcul de noyau par M-bord, cet algorithme ne nécessite ni tri ni file de priorité. De plus, il ne nécessite pas le pré-calcul des minima.

```
Function Flux [8] ( V, E, F, Ψ, x ):
    Data : (V, E, F) un graphe à arête valuées; Ψ un étiquetage de V et
           x un point de V.
    Result : [L, lab]avec L est un flux tel que x est une source de L et
             lab est soit l'étiquette d'un ≺ − flux sous L soit -1.
1   L := {x};
2   L' := {x};// l'ensemble des sources non encore explorée de L
3   while ( ∃ y ∈ L' ) do
4       L' := L' \ {y};
5       breadth_first:=TRUE;
6       while (breadth_first et ∃ {y, z} ∈ E tel que z ∉ L et
        F({y, z}) = F(y) ) do
7           if (Ψ(z) ≠ NO_LABEL) then
8               // il existe un ≺ − flux sous L déjà étiqueté
9               retourner[L, Ψ(z)];
10          else if (F(z) < F(y)) then
11              L := L ∪ {z};//z est maintenant le seul puits de L
12              L' := {z};// basculer dans une phase d'exploration en
                profondeur d'abord
13              breadth_first:= FALSE;
14          else
15              L := L ∪ {z};//F(z) = F(y) donc z est aussi un puits de L
16              L' := L' ∪ {z};//continuer l'exploration en largeur d'abord
17  retourner[L,-1];
```

8.3 Bilan sur les algorithmes séquentiels de calcul de la LPE

Nous avons passé en revue les algorithmes séquentiels permettant de calculer les principales variantes de la LPE. Ces algorithmes peuvent être analysés sous plusieurs angles. Dans ce travail nous dressons une comparaison entre ces algorithmes en se basant sur un ensemble de critères. Ces derniers peuvent être divisés en deux catégories. Les critères qui sont en relation avec la nature algorithmique et les critères qui concernent la qualité du résultat produit.

Du côté algorithmique, nous avons retenu les propriétés qui peuvent avoir une influence sur une implémentation parallèle des algorithmes séquentiels. Dans ce cadre, nous nous sommes intéressés :

- à la nature itérative ou récursive de l'algorithme,
- au cadre d'application de l'algorithme en termes de graphes à sommets pondérés ou graphes à arêtes valuées,
- aux structures de données utilisées notamment les files d'attente, et les files d'attentes hiérarchiques,
- à la linéarité de la complexité,
- à la présence d'étapes supplémentaires au niveau fonctionnel telles que l'étape de tri et de recherche des minima qui sont des étapes coûteuses.

Du côté de la qualité des résultats ce sont essentiellement la préservation de la topologie et l'épaisseur des contours qui sont examinés.

Le tableau 1.1 résume toutes ces propriétés pour les algorithmes étudiés dans ce chapitre.

Le point de départ est la nature de l'algorithme. En effet, nous remarquons que les différents algorithmes présentés sont itératifs à l'exception de l'algorithme de calcul de la LPE d'arêtes basé sur l'extraction des flux [8] (Algorithme 6) et de l'algorithme de calcul de la LPE par immersion proposé par Meyer [10] (Algorithme 2) qui sont récursifs.

De point de vue structure de données utilisées, les algorithmes de calcul de la LPE d'arêtes proposés par Cousty et al [8, 30] ainsi que l'algorithme de calcul de la LPE par immersion proposé par Vincent-Soille [9] (Algorithme 1) ne nécessitent pas l'utilisation d'une File d'Attente Hiérarchique. Cependant, ce dernier algorithme requiert une étape de tri des pixels de l'image contrairement aux deux premiers algorithmes. D'autre part, seuls les algorithmes de calcul de la LPE d'arêtes à base de flux [8] (Algorithme 6) et de la LPE topologique [18] (Algorithme 4) ne nécessitent pas le pré-calcul des minima régionaux.

Du coté complexité, trois types de LPE peuvent être calculés en temps linéaire. En effet, l'algorithme 1 est linéaire par rapport au nombre de pixels de l'image en entrée. L'algorithme de calcul de la LPE topologique [18] qui est appliqué dans le cadre des Graphes à Sommets Pondérés est linéaire par rapport à (N+M) où N désigne le nombre de sommets et M le nombre d'arêtes du graphe associé à l'image de départ. Il est à noter que cette complexité est

calculée dans le cas où la plage de variation du niveau de gris de l'image traitée est faible et le nombre de voisins d'un sommet est constant. D'autre part, la complexité au pire des cas des algorithmes de calcul de la LPE d'arêtes [8,30] (qui sont appliqués dans le cadre des GAV) est de l'ordre de *O(M)* tel que *M* désigne le nombre d'arêtes du graphe de départ.

De point de vue de la qualité du résultat de segmentation, les algorithmes de calcul de la LPE par ascension de collines [21] (Algorithme 3) de la LPE par érosion [17] (Algorithme 5) et de la LPE d'arêtes [8, 30] produisent une LPE mince. Mais, seuls les algorithmes de calcul de la LPE topologique et de la LPE d'arêtes permettent de préserver la topologie de l'image. D'autre part, le cadre d'application des algorithmes est un critère très important qui affecte la qualité de résultat obtenu. En effet, seuls les algorithmes de calcul de la LPE d'arêtes et de la LPE par érosion sont appliqués dans le cadre des graphes de fusion parfaite. Ce cadre garantit certaines propriétés morphologiques (telle que la résolution du problème de fusion de région, etc.) détaillées dans [7, 17, 22, 23]. Ce qui rend ces algorithmes efficaces dans ce cadre.

	LPE par immersion		**LPE par ruissellement**	**LPE Topologique**	**LPE par érosion**	**LPE d'arêtes**	
	Vincent-Soille [9]	*F. Meyer [10]*	*Hill-Climbing[21]*	*Couprie et al [18]*	*Cousty et al [17]*	*M-bord [30]*	*Flux [8]*
	Algorithme 1	Algorithme 2	Algorithme 3	Algorithme 4	Algorithme 5		Algorithme 6
Type d'algorithme	Itératif	Récursif	--	Itératif	Itératif	Itératif	Récursif
Etape de tri	oui	Oui	Non	Non	Non	Non	Non
Utilisation d'une FAH	Non	Oui	Oui	Oui	Oui	Non	Non
Cadre d'application	Matrice(1)	GSP	GSP	GSP(2)	GAV(3)	GAV(3)	GAV(3)
Calcul des minima	Oui	Oui	Oui	Non	Oui	Oui	Non
Algorithme linéaire	Oui	Non	--	Oui(4)	Non	Oui	Oui
Préservation de la topologie	Non	Non	Non	Oui	--	Oui	Oui
LPE mince ou épaisse	épaisse	épaisse	Mince	--	Mince	Mince	Mince

(1)Cet algorithme peut être modifié pour être appliqué sur une image représentée sous forme de graphe.
(2)Graphe à sommets pondérés : GSP.
(3)Graphe à arêtes valuées : GAV.
(4)Il s'agit d'un algorithme linéaire lorsque la plage de variation du niveau de gris de l'image traitée est faible.

Tableau 1. 1. Comparaison des algorithmes séquentiels de calcul de la LPE

9. Conclusion

Depuis son apparition et son adaptation au contexte de la segmentation d'images la LPE a fait l'objet de plusieurs travaux qui l'ont défini en simulant trois principaux phénomènes physiques qui sont l'immersion d'un relief dans l'eau, le ruissellement des gouttes d'eau sur un relief et enfin le ravinement du relief. Pour chacun de ces principes, plusieurs algorithmes ont été proposés pour le calcul de la LPE. Dans ce chapitre, nous avons passé en revue les algorithmes les plus connus parmi ceux proposés dans la littérature et nous avons dégagé leurs propriétés entre autres en termes d'aspect algorithmique, de structure de données et de qualité du résultat produit.

Dans ce cadre, nous avons souligné que les algorithmes de la LPE d'arêtes, notamment l'algorithme de calcul de noyau par M-bord, se distinguent par leur préservation des propriétés topologiques et par la précision des contours des régions qu'ils produisent. Ce dernier point a été confirmé par Cousty et al [1, 17] qui a prouvé l'efficacité de cette catégorie de LPE dans la délinéation du myocarde ventriculaire gauche dans des images de type IRM. Outre la qualité, l'algorithme de calcul de noyau par M-bord a la particularité d'être linaire et donc parmi les plus rapides relativement aux restes des algorithmes de calcul de la LPE. Toutefois, malgré cette efficacité, cet algorithme, à l'image de la majorité des algorithmes de calcul de la LPE, reste d'une manière absolue un processus assez coûteux en termes de temps d'exécution. En effet, il fait intervenir des traitements qui peuvent s'avérer assez lourds lorsque le volume des données considérées devient important tels que les opérations de manipulation de files d'attente, de recherche des minima régionaux et de parcours de graphes en largeur et/ou en profondeur. Ce problème de performance devient plus accru lors de la manipulation d'images de grandes tailles ou des séries d'images ce qui est le cas par exemple des images produites par les modalités médicales.

Comme contribution à l'optimisation globale de toute la chaîne de segmentation et de visualisation 3D du myocarde ventriculaire gauche du cœur proposée et validée dans [1], nous proposons l'optimisation en termes de temps d'exécution de l'algorithme [30]. Pour ce faire, la parallélisation représente l'une des solutions les plus intuitives à étudier. Ainsi, nous consacrons le chapitre suivant pour explorer les différents travaux qui se sont intéressés à la parallélisation de la LPE avec ses différentes catégories et ce, afin de dégager les problématiques que soulèvent cette tâche et les solutions qui peuvent s'avérer adéquates à

notre contexte. Ceci nous sera utile pour proposer une solution spécifique à l'algorithme qui fait l'objet de notre étude.

Chapitre 2 : Calcul parallèle et Lignes de Partage des Eaux

1. **Introduction**
2. **Traitement parallèle et modèles des stratégies de parallélisation**
3. **Méthodologie de conception des programmes parallèles**
   - **3.1. Le partitionnement**
   - **3.2. La communication**
   - **3.3. L'agglomération**
   - **3.4. L'ordonnancement**
4. **Types de parallélisation**
   - **4.1. Parallélisme de données**
   - **4.2. Parallélisme de contrôle**
5. **Architectures parallèles**
   - **5.1. Machines à mémoire partagée**
   - **5.2. Machines à mémoire distribuée**
   - **5.3. Machines à mémoire hybride**
6. **Modèles de programmation parallèle**
   - **6.1. Le modèle de programmation pour mémoire partagée**
   - **6.2. Le modèle de programmation à base de threads pour mémoire partagée**
   - **6.3. Le modèle de programmation par échange de message**
   - **6.4. Le modèle de programmation parallèle hybride**
7. **Mesure des performances**
   - **7.1. Temps d'exécution**
   - **7.2. Accélération**
   - **7.3. Efficacité**
8. **Parallélisation de la Ligne de Partages des Eaux**
   - **8.1. Algorithmes parallèles de calcul de la LPE par immersion**
   - **8.2. Algorithmes parallèles de calcul de la LPE par ruissellement**
   - **8.3. Algorithmes parallèles de calcul de la LPE topologique**
   - **8.4. Algorithmes parallèles de calcul de la LPE d'arêtes**
   - **8.5. bilan sur les algorithmes parallèles de calcul de la LPE**
9. **Conclusion**

1. Introduction

L'objectif principal des travaux menés dans le cadre de cette thèse est l'optimisation des performances de l'algorithme de calcul de la LPE d'arêtes par M-bord à travers l'exploration de différentes stratégies de sa parallélisation, pour en déduire la plus pertinente. Pour ce faire, nous présentons dans ce chapitre les éléments et les concepts nécessaires pour bien appréhender cette problématique. La première partie est dédiée à rappeler les fondements de base relatifs au calcul parallèle notamment les architectures et les modèles de programmation qui permettent de le mettre en œuvre. La deuxième partie de ce chapitre relate l'application de ces concepts à la parallélisation de la Ligne de Partage des Eaux. Dans ce cadre, un état de l'art des principaux travaux de parallélisation de la Ligne de Partage des Eaux est rapporté. Ces travaux sont classés selon le phénomène physique utilisé pour la formulation du calcul de cette transformation. L'accent est mis sur la stratégie de parallélisation adoptée et l'architecture matérielle visée.

2. Traitement parallèle et modèles des stratégies de parallélisation

Le traitement parallèle peut être défini comme étant la division d'une quantité donnée de travail sur différents processeurs qui s'exécutent simultanément. Le principal avantage de tels traitements vient de leurs capacités à gérer des grands volumes de tâches ou de données avec une latence et une cadence raisonnables. L'objectif étant d'aboutir à des performances meilleures, entre autres, à des temps de calcul réduits en comparaison avec des implémentations sur un seul processeur.

La stratégie de parallélisation consiste à utiliser un ensemble de processeurs capables de communiquer et de coopérer dans le but d'accélérer la résolution de problèmes. Il existe dans la littérature plusieurs stratégies de parallélisation. Certaines de ces stratégies sont communes et s'appliquent à un grand nombre de problèmes et constituent par conséquent des modèles de parallélisation. L'adaptation d'un modèle à un problème dépend des caractéristiques de ce dernier. Les principaux modèles utilisés dans ce cadre sont :

(i) Le modèle de parallélisation des boucles : Il s'agit de la parallélisation des structures itératives dans les programmes. Dans ce cas, si plusieurs itérations ou groupes d'itérations sont indépendantes, alors leur exécution peut être lancée en parallèle.

(ii) Le modèle Fork/Join : ce modèle peut être appliqué dans les programmes qui comportent des sections séquentielles et des sections parallèles. Le programme s'exécute d'abord sous forme d'une tâche principale. Ensuite, lorsqu'une section parallèle est rencontrée, un groupe de threads est créé. Chaque thread traite une tâche de la partie parallèle. Cette action est appelée « Fork ». Une fois la section parallèle terminée, les threads du groupe sont fusionnés au niveau d'un point de synchronisation. Cette action est appelée « Join ».

(iii) Le modèle de décomposition en domaines : ce modèle consiste à diviser l'espace des données en des parties appelées domaines. Chaque domaine est traité par un ou plusieurs threads. Le résultat final est composé par l'ensemble des résultats obtenus dans les différents domaines. Dans certains cas, une phase de fusion des résultats locaux des différents processeurs est requise afin d'obtenir le résultat global adéquat. Notons de plus que pour ce modèle de stratégie, la division de l'espace des données se fait d'une façon statique.

Certaines de ces stratégies seront exploitées et adaptées à certains algorithmes dans les chapitres suivants de ce manuscrit.

3. Méthodologie de conception des programmes parallèles

La diversité des problématiques posées par la parallélisation engendre une diversité d'approches qui permettent de les résoudre. Ces approches doivent s'adapter aux spécificités du programme à paralléliser en termes de données manipulées et natures des tâches à réaliser. L'élaboration de la majorité de ces approches passe le plus souvent par quatre étapes communes qui sont le partitionnement, la communication, l'agglomération et l'ordonnancement. Selon [37], ces étapes constituent une méthodologie de conception de programmes parallèles nommée PCAM (Partitionning, Communication, Agglomeration, Mapping).

3.1 Le partitionnement

Le partitionnement consiste à décomposer le problème initial en plusieurs tâches de calcul. Le partitionnement peut être fait selon le domaine ou d'une manière fonctionnelle. Dans le cas de partitionnement selon le domaine, l'accent est mis sur la répartition des données sur différents processeurs. Chaque processeur va traiter une partie des données du problème initial. Le partitionnement fonctionnel touche un autre aspect qui est celui des

traitements à faire. Dans ce cas le partitionnement produit des sous-tâches disjointes qui peuvent être exécutées en parallèle. Cette étape doit être précédée d'abord par une analyse des dépendances entre les sous-tâches avant de les dissocier. Le partitionnement selon le domaine et le partitionnement fonctionnel ne sont pas nécessairement exclusives. L'un ou l'autre ou les deux en même temps peuvent être appliqués à un problème de parallélisation selon la nature de ce dernier. Selon Foster [37], pour que le partitionnement soit efficace, les points suivants doivent être vérifiés :

(i) Le nombre de tâches produites par le partitionnement ne doit pas être inférieur à celui des processeurs car dans un tel cas certains processeurs restent oisifs. En même temps, le souci d'exploiter tous les processeurs ne doit pas engendrer des tâches de tailles très réduites en termes de charge, car ceci impliquerait un coût de communication qui dépasserait le coût de traitement. Un partitionnement efficace doit établir un compromis entre granularité de la décomposition et charge de traitement.

(ii) Les tâches produites doivent être de tailles comparables en termes de charge. Ceci permet d'assigner aux différents processeurs des volumes de travail équilibrés et d'éviter les temps d'attente entre processeurs.

(iii) Le partitionnement doit éviter la redondance des traitements et des stockages des données.

(iv) Le partitionnement doit être évolutif (c'est-à-dire scalable) par rapport au nombre de tâches. Ainsi une augmentation de la taille du problème initial doit engendrer une augmentation du nombre de tâches plutôt qu'une augmentation de la taille de ces dernières.

3.2 La communication

La communication est une conséquence du partitionnement qui engendre naturellement un besoin de coordination et d'échanges de données entre les sous-tâches de calcul qui s'exécutent en parallèle. Dans cette étape, les besoins en communication sont spécifiques à la nature de données à échanger, aux structures et aux protocoles permettant de les échanger.

Dans le cas d'un partitionnement selon le domaine, les tâches produites travaillent généralement sur des sous-ensembles de données différents spécifiques à chaque tâche. Toutefois, dans certaines situations, des opérations peuvent demander des données provenant d'autres tâches. Dans ce cas, la communication est sensée gérer le transfert des données

nécessaires entre les tâches. Organiser cette communication d'une façon efficace reste difficile. Même les décompositions simples peuvent engendrer des structures de communications complexes.

Contrairement à cela, les besoins en communication entre les tâches issues d'un partitionnement fonctionnel sont plus simples en termes de structures et de gestion. La communication dans ce cas se réduit à des flux de données entre une tâche productrice et une tâche consommatrice de données.

Pour satisfaire la contrainte d'efficacité de l'étape de communication, les caractéristiques suivantes doivent être respectées [37] : (i) l'étape de partitionnement produit des tâches de calcul qui communiquent avec le nombre le plus réduit possible des autres tâches, (ii) l'étape de partitionnement produit des tâches différentes ayant un même volume de communication (iii) les opérations de communication se déroulent d'une manière parallèle.

3.3 L'agglomération

Les étapes de partitionnement et de communication produisent une approche de parallélisation qui se situe au niveau conceptuel sans aucune adaptation par rapport à l'architecture qui sera réellement utilisée. Cette adaptation est nécessaire pour assurer l'efficacité de l'approche proposée. Par exemple si le partitionnement produit un nombre de tâches qui dépasse de loin le nombre de processeurs réellement disponibles sur la machine d'exécution, la performance du programme parallèle sera réduite. L'objectif de l'agglomération est de garantir cette adaptation, et ce, en évaluant les tâches produites par le partitionnement et les structures de communication afin de fournir un programme adapté à un type d'architecture donné. Elle procède généralement par combinaison des petites tâches en des tâches de tailles plus importantes et par réplication des traitements et des données et ce afin d'améliorer les performances et diminuer les coûts de communication.

3.4 L'ordonnancement

L'ordonnancement est la dernière étape dans la conception des programmes parallèles. Elle consiste à assigner concrètement chaque tâche à un processeur de manière à optimiser l'exploitation des processeurs et à diminuer les coûts de communication et de synchronisation entre processeurs. L'ordonnancement peut être statique ou dynamique déterminé lors de l'exécution à l'aide des algorithmes d'équilibrage de charge et d'ordonnancement de tâches.

La parallélisation des programmes est à l'origine une tâche manuelle: c'est le programmeur qui est responsable de la spécification explicite des différentes étapes de la parallélisation. Toutefois, il y a eu des propositions d'outils qui offrent une assistance au programmeur dans la réalisation de cette tâche et qui le décharge de la spécification explicite de certaines étapes à travers une parallélisation automatique et implicite. Ces outils sont généralement intégrés dans certains compilateurs.

4. Types de parallélisation

Le parallélisme peut être classé selon plusieurs critères parmi lesquels la source qui est à l'origine de ce parallélisme. Selon ce critère, deux principaux types peuvent être distingués : le parallélisme des données et le parallélisme des tâches.

4.1 Parallélisme de données

Ce type de parallélisme désigne le cas où plusieurs unités de traitement exécutent en parallèle la même opération sur un ensemble de données. Ces données sont généralement organisées dans une structure de données commune se présentant dans la majorité des cas sous forme d'un vecteur. Le principal souci de ce type de parallélisme est comment distribuer les données sur les différentes unités de traitement tout en les gardant indépendantes. Le programme parallèle se présente alors comme une succession de phases de traitement et de redistribution des données.

Grâce à son principe, le parallélisme de données est facile à exploiter. Il reste toutefois limité à un certain type d'applications simples où les données sont organisées sous forme de vecteurs. Cependant, avec la complexité croissante des applications modernes ce type de parallélisme se trouve souvent utilisé conjointement avec d'autres types de parallélisme [45].

4.2 Parallélisme de contrôle

Ce type de parallélisme également connu sous le nom de parallélisme de tâches ou parallélisme fonctionnel repose sur la décomposition du programme informatique sous la forme d'un graphe orienté sans cycle. Chaque nœud dans ce graphe représente une tâche composée d'une suite d'opérations exécutée d'une manière séquentielle. Les arcs du graphe représentent l'ordre de précédence auquel doit obéir l'exécution des différentes tâches. Le graphe ainsi obtenu est appelé graphe de précédence et définit les dépendances qui peuvent exister entre les tâches en termes de données produites et données consommées. Les tâches de

ce graphe qui ne présentent pas de contraintes de dépendance peuvent être exécutées en parallèle.

5. Architectures parallèles

Une machine parallèle est une machine qui dispose d'un ensemble de ressources en termes de processeurs et de mémoires capables de coopérer afin d'effectuer des calculs complexes. Cette définition s'applique aussi bien aux machines multiprocesseurs qu'aux réseaux de stations de travail ou encore aux supercalculateurs parallèles comportant des milliers de processeurs.

Les architectures des machines parallèles peuvent être classées de différentes manières. Toutefois, la classification proposée par Flynn [41], connue sous le nom de « Flynn Taxonomy » continue à être utilisée, notamment par rapport à sa simplicité. Dans cette classification quatre types de machines parallèles sont distinguées selon deux paramètres indépendants qui sont les instructions et les données : les machines SISD (Single Instruction Single Data), les machines MISD (Multiple Instruction Single Data), les machines SIMD (Single Instruction Multiple Data) et les machines MIMD (Multiple Instruction Multiple Data).

Les machines SISD (figure 2.1) sont des architectures matérielles dans laquelle un seul processeur exécute un seul flot d'instruction sur des données résidant dans une seule mémoire. De ce fait, il n'y a aucune parallélisation. On retrouve là l'architecture de Von-Neumann.

Les machines MISD (figure 2.1 (b)) sont des machines dans lesquels la même donnée est traitée par plusieurs processeurs en parallèle. Il existe peu d'implémentations en pratique.

Les machines SIMD (figure 2.1 (c)) sont capables d'exécuter un même flot d'instructions simultanément sur des données multiples. Elles sont moins complexes à utiliser mais ne permettent pas cependant de traiter efficacement tous les types de problèmes. Elles sont plus adaptées aux applications orientées données; par exemple, dans le domaine de traitement d'images où les mêmes traitements sont appliquées de la même manière sur les différents pixels de l'image.

Les machines MIMD (figure 2.1 (d)) sont des machines qui sont capables d'exécuter différents flots d'instructions sur des données multiples. Elles offrent plus de flexibilité pour

paralléliser une large gamme d'applications. Ces machines peuvent être réalisées par essentiellement deux différentes configurations matérielles qui se différencient par la localisation de leur mémoire : les multiprocesseurs et les multi-ordinateurs. Les multiprocesseurs sont des machines disposant de plusieurs processeurs partageant un même espace mémoire. Elles sont connues sous le nom de machines MIMD à mémoire partagée. Les multi-ordinateurs se présentent sous forme d'un réseau où chaque nœud est un ordinateur indépendant disposant de son propre processeur et de sa propre mémoire locale. Ces machines sont appelées des machines à mémoire distribuée. Il est à noter que des architectures hybrides entre les deux types de machines MIMD peuvent exister. Ces sont les multi-ordinateurs de multiprocesseurs.

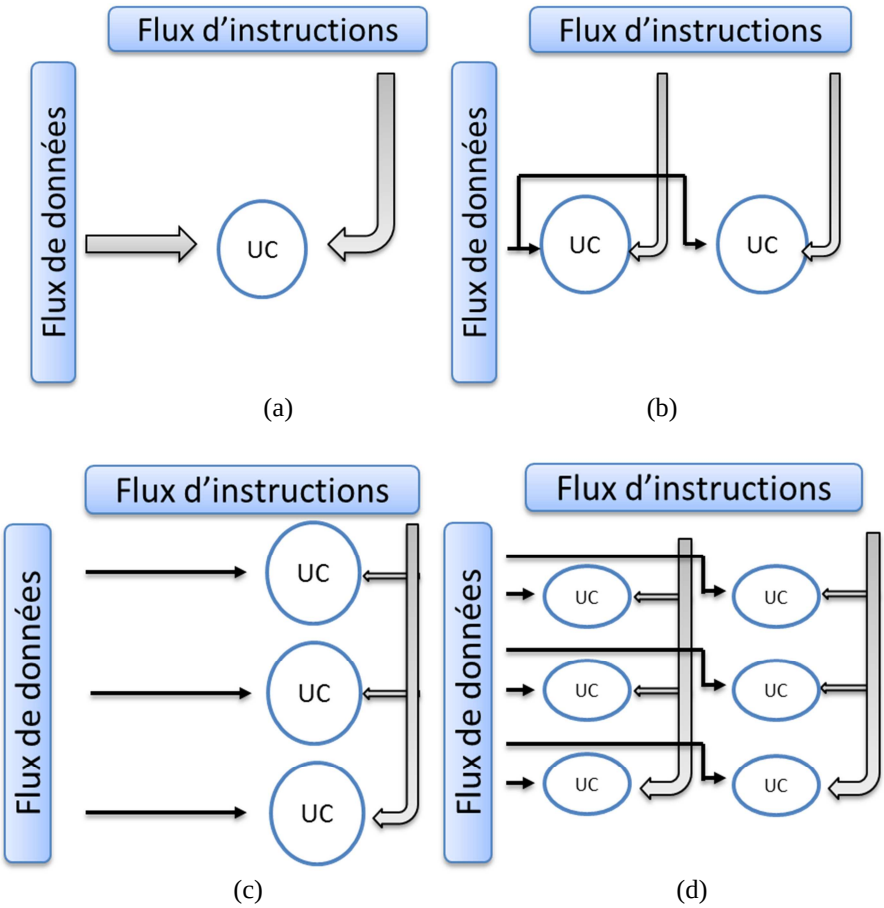


Figure 2. 1 Machine (a) SISD (b) MISD (c) SIMD et (d) MIMD avec UC désigne l'unité de calcul qui peut être d'un processeur uni-cœur ou multi-cœur

Vue du côté de l'organisation de la mémoire, trois principales architectures parallèles de type MIMD peuvent être distinguées : les architectures à mémoire partagée, les architectures à mémoires distribuée et les architectures hybrides. Ces architectures sont intéressantes à étudier car chacune va induire un modèle de programmation parallèle qui lui est adaptée.

5.1 Machines à mémoire partagée

Les architectures parallèles à mémoire partagée présentent une mémoire unique accessible directement par l'ensemble de leurs processeurs via un bus ou une hiérarchie de bus (figure 2.2). La plupart des machines à mémoire partagée sont des architectures symétriques nommées SMP (Symmetric MultiProcessor) où tous les processeurs ont les mêmes fonctions en occurrence l'accès aux ressources du système de façon uniforme.

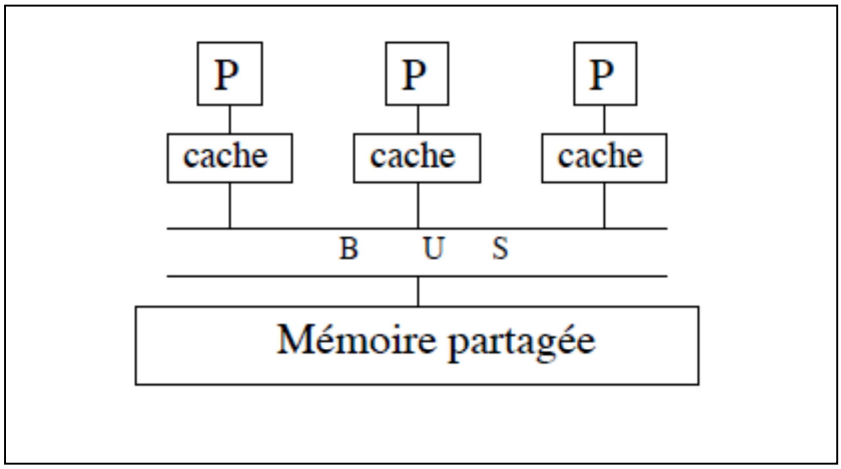


Figure 2. 2 Machine multiprocesseurs à mémoire partagée

L'existence d'une mémoire commune entre les processeurs simplifie la tâche de parallélisation des algorithmes. La communication entre les processeurs est facilitée par le fait que les communications se font par le biais de lectures et d'écritures de zone de mémoire partagée. Ceci rapproche la logique de construction de programmes de celle de la programmation séquentielle traditionnelle. D'autre part, l'existence d'une mémoire commune peut également poser un problème car tous les processeurs doivent y accéder. Dans le cas du modèle PRAM (Parallel Random Access Machine), souvent utilisé pour étudier d'un point de vue théorique des algorithmes parallèles [43], tout processeur accède en une même durée de temps à la mémoire. En pratique, le fait d'augmenter le nombre de processeurs introduit généralement une forme de hiérarchie mémoire. En particulier, la contention d'accès à la mémoire partagée peut être réduite en conservant dans le cache de chaque processeur des copies des données fréquemment accédées. En effet, les accès dans les caches des processeurs sont beaucoup plus rapides que ceux effectués dans la mémoire partagée, mais apparaissent alors des problèmes de cohérence et de synchronisation.

5.2 Machines à mémoire distribuée

Les machines à mémoire distribuée (figure 2.3) se présentent comme un réseau où chaque nœud est composé d'un processeur et d'une mémoire locale qui lui est associée. La communication entre les mémoires locales et par conséquent l'interaction entre les processeurs sont assurées par le réseau. Chaque processeur dispose de sa propre mémoire privée dans laquelle il travaille d'une manière indépendante ce qui permet d'éviter les problèmes de cohérence de cache rencontrés dans les architectures à mémoire partagée.

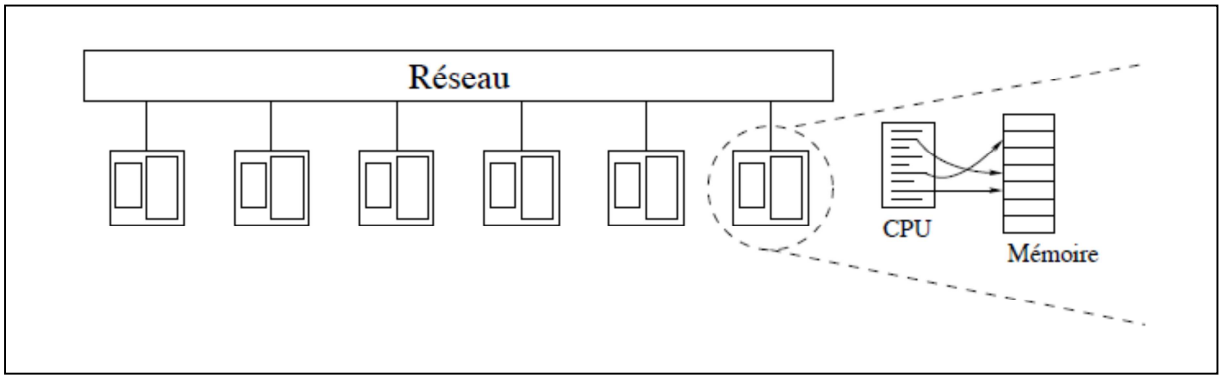


Figure 2. 3 Modélisation d'une machine parallèle MIMD à mémoire distribuée. Chaque nœud est composé d'un processeur et d'une mémoire locale et communique avec ses voisins par envoi de messages via le réseau

Ces architectures peuvent atteindre des tailles importantes en termes de processeurs et de mémoires puisque le problème de contention entre les processeurs pour accéder à la mémoire ne se pose pas. Elles sont de ce fait plus adaptées à un parallélisme massif. Toutefois la distribution de la mémoire engendre un besoin en synchronisation et un coût supplémentaire en communication entre les processeurs pour échanger leurs résultats.

5.3 Machine à mémoire hybride

L'architecture des machines de ce type se présente comme une combinaison des architectures des deux modèles précédents (mémoire partagée et mémoire distribuée). La mémoire dans ce cas est globalement distribuée et localement partagée. Les systèmes ayant cette architecture sont généralement des grappes de calculateurs ou chaque nœud est une machine ayant une architecture à mémoire partagée. Ils peuvent se manifester ainsi sous forme d'un ensemble d'ordinateurs indépendants interconnectés par un réseau haut débit et gérés par un middleware spécialisé permettant de les faire fonctionner en parallèle ou également sous forme d'un système composé de plusieurs machines de type SMP interconnectés via un bus de communication et d'entrée/sortie. La mémoire est de ce fait

globalement distribuée entre les différents nœuds et localement partagée à l'intérieur de chaque nœud.

Les machines hybrides bénéficient à la fois des avantages offerts par la mémoire partagée en termes de facilité d'accès aux données et de ceux offerts par la mémoire distribuée en termes d'extensibilité. Ces machines sont de plus en plus utilisées aujourd'hui car elles permettent d'obtenir des performances de calcul assez élevées. En effet, ce type d'architectures peut être rencontré dans plusieurs systèmes comme les clusters de machines hétérogènes, dans les systèmes Multi-Cores ou dans les systèmes Many-Cores en particulier les systèmes de type GPU (Graphics Processing Units).

6. Modèles de programmation parallèle

Avec la multitude et la complexité des architectures parallèles, le développement de programmes sur ces architectures devient une tâche assez complexe. Afin de réduire cette complexité, des modèles de programmation parallèles ont été établis. Ces modèles définissent une sorte de couche d'abstraction entre le développement d'une application et les détails techniques des architectures. Ils permettent ainsi de cacher des détails concernant l'exécution parallèle comme le transfert et le routage des messages, l'allocation des tâches et la variabilité des plateformes tout en exposant d'une manière haut niveau les fonctionnalités de la programmation parallèle. L'implémentation des modèles de programmation parallèles peut prendre différentes formes notamment en tant qu'un ensemble de langages, d'extensions de langages existants ou de bibliothèques et outils qui permettent de mapper les architectures parallèles. Nous présentons dans ce qui suit les principaux modèles utilisés dans ce cadre.

6.1 Le modèle de programmation pour mémoire partagée

Dans ce modèle de programmation, les tâches partagent un même espace mémoire comportant un adressage commun et global. La communication est assurée par des opérations de lecture et d'écriture de données qui sont effectuées dans cet espace d'une manière asynchrone. Le contrôle de l'accès à la mémoire partagée est assuré par des mécanismes de synchronisation tels que les verrous et les sémaphores. Ce modèle est considéré comme étant le modèle de programmation parallèle le plus simple. Sa principale limitation vient du surcoût qu'engendrent la synchronisation et le rafraichissement de cache ce qui affecte la performance. Ce modèle est implémenté d'une manière native sur la plupart des systèmes d'exploitation des machines ayant des architectures à mémoire partagée. Il est essentiellement

adapté aux architectures parallèles de type SMP et ne supporte pas les architectures hétérogènes [47, 65].

6.2 Le modèle de programmation à base de threads pour mémoire partagée

Ce modèle de programmation parallèle peut être considéré comme un cas particulier du modèle à mémoire partagée. Dans ce cas, le programme à paralléliser se présente comme un programme principal qui s'exécute d'une manière séquentielle mais qui comporte certaines parties pouvant être exécutées d'une manière parallèle. Ces parties sont alors placées dans des threads ordonnancés et exécutés d'une manière concurrente par le système d'exploitation. Le nombre de threads peut être déterminé d'une manière dynamique en fonction de la taille du problème et du nombre de tâches. Ces threads ont une durée de vie variable et sont créés et détruits d'une manière explicite.

La tâche exécutée par un thread peut être décrite comme une sous-routine du programme principale. Chaque thread possède ses propres données locales mais tous les threads peuvent partager également des données globales situées dans l'espace d'adressage du programme principale. La communication entre les threads est ainsi assurée par la mémoire partagée ce qui engendre un besoin en opérations de synchronisation entre les threads afin de garantir l'exclusivité de l'accès à une information donnée.

Le modèle de programmation par thread est adapté aux architectures à mémoire partagée. Les implémentations de ce modèle se présentent souvent sous forme de bibliothèques de fonctions ou des directives de compilation. Plusieurs implémentations de ce modèle de programmation parallèle existent parmi lesquelles deux se sont imposées comme des standards : les threads POSIX et OpenMP [64, 65].

6.3 Le modèle de programmation par échange de messages

Dans ce modèle de programmation, les tâches de calcul parallèle utilisent chacune sa propre mémoire locale. La communication est assurée par envoi et réception de messages entre ces différentes tâches. Les messages peuvent être échangés en mode synchrone ou asynchrone. Les implémentations de ce modèle de programmation se présentent sous forme de bibliothèques de sous-routines offrant les outils nécessaires de communication. Le parallélisme est ainsi explicite et se fait à travers l'appel de ces sous-routines. Les

bibliothèques MPI et PVM se sont imposées comme standards pour ce modèle de programmation parallèle.

Ce modèle de programmation est adapté aux architectures à mémoires distribuées et aux architectures hétérogènes. Il a l'avantage d'être extensible (scalable) par rapport au nombre croissant de tâches de calcul.

6.4 Le modèle de programmation parallèle hybride

Le modèle de programmation parallèle hybride combine deux ou plusieurs modèles de programmation parmi ceux cités dans les paragraphes précédents. Un exemple classique de modèle hybride est celui qui se base sur la combinaison entre le modèle par échange de messages (MPI) et le modèle à base de threads (OpenMP). Dans ce modèle les threads assurent les traitements intensifs sur des données locales aux différents nœuds. Les nœuds échangent leurs résultats via le réseau par envoi de messages. Ce modèle se prête bien aux architectures de type clusters de machines Multi/Many-Cores.

7. Mesure des performances

L'objectif principal du parallélisme est d'améliorer la performance des programmes. Cette performance est évaluée à travers des critères qui permettent de juger d'une manière quantitative l'apport d'une stratégie de parallélisation. Les critères en question sont principalement exprimés en fonction du volume des données en entrée, cependant, d'autres facteurs peuvent être considérés. Nous présentons dans ce qui suit ces principaux critères.

7.1 Temps d'exécution

Un algorithme séquentiel est généralement évalué en termes de temps d'exécution exprimé en fonction de la taille de ses entrées pour une architecture donnée. Le temps d'exécution séquentiel (noté T_{seq}) d'un programme correspond au temps qui s'écoule entre le début et la fin de son exécution sur une machine séquentielle. Le temps d'exécution d'un algorithme parallèle (noté T_P) dépend non seulement de la taille de ses entrées mais également du nombre de processeurs disponibles. Ce temps correspond au temps qui s'écoule entre le moment de début du calcul parallèle et le moment où le dernier processeur termine son exécution.

7.2 Accélération

Le temps d'exécution ne constitue pas la métrique la plus efficace pour évaluer la performance d'un algorithme parallèle puisque ce temps varie en fonction de la taille du problème. Il est plus approprié d'utiliser une métrique relative comme le gain en termes de performance obtenu entre une exécution séquentielle et une exécution parallèle du même problème. Ce gain est exprimé à l'aide de l'accélération qui est définie comme étant le rapport entre le temps d'exécution séquentielle et le temps d'exécution parallèle. La formulation usuelle de l'accélération notée A_P est donnée par:

$$A_P = \frac{T_{seq}}{T_P} \qquad (2.1)$$

Selon le type du temps séquentiel considéré, deux types d'accélérations peuvent être distinguées :

(i) *l'accélération absolue* donnée par le rapport entre le temps d'exécution du meilleur algorithme séquentiel qui résout le problème divisé par le temps d'exécution de l'algorithme parallèle qui résout ce même problème sur une machine à *P* processeurs;

(ii) *l'accélération relative* qui est donnée par le rapport entre le temps d'exécution de l'algorithme parallèle qui résout le problème sur un unique processeur divisé par le temps d'exécution de ce même algorithme parallèle avec *P* processeurs.

Il est à noter que les *P* processeurs utilisés dans l'exécution parallèle sont supposés être identiques au processeur utilisé dans l'exécution séquentielle.

Un système parallèle contenant *P* processeurs est considéré comme idéal s'il permet d'obtenir une accélération relative égale à *P*. Toutefois sur un plan pratique, le comportement idéal n'est presque jamais atteint car dans une exécution parallèle, en plus des traitements, les processeurs passent également du temps à communiquer et à synchroniser leurs résultats.

Dans le reste de ce manuscrit, et en raison de simplification, l'accélération relative est le type d'accélération utilisé pour évaluer les différents algorithmes parallèles présentés dans ce manuscrit. Elle est nommée accélération tout court.

7.3 Efficacité

L'efficacité est une autre métrique de mesure de performance similaire à l'accélération qui par rapport à cette dernière permet de fournir une mesure normalisée (théoriquement comprise entre 0 et 1). L'efficacité notée E_p est définie comme étant le rapport entre l'accélération et le nombre de processeurs (2.2).

$$E_P = \frac{A_P}{P} \qquad (2.2)$$

L'efficacité donne une indication sur le degré d'exploitation des processeurs durant l'exécution parallèle d'un programme. Tout comme l'accélération elle peut être définie d'une manière relative ou absolue.

Si on considère le cas où on dispose d'un nombre infini de processeur ($P \to \infty$), l'accélération maximale va tendre vers 1. La loi d'Amdahl limite alors l'accélération maximale à la fraction non parallélisable du programme. Même avec un nombre infini de processeurs, l'accélération ne pourra pas dépasser la limite définie par le temps séquentiel. Ceci est une limitation importante car la majorité des algorithmes ne peuvent être que partiellement parallélisés.

Après avoir présenté les différents concepts de base liés au parallélisme, les différentes architectures parallèles et les modèles de programmation associés et les mesures de performance de parallélisation des algorithmes, nous passons dans la deuxième partie de ce chapitre à présenter les algorithmes parallèles les plus connus dans la littérature de calcul des différentes variantes de la LPE.

8. Parallélisation de la Ligne de Partage des Eaux

Les différentes variantes de la LPE (présentées dans le chapitre 1) ont fait l'objet de nombreux travaux qui ont visé leurs parallélisations. Nous nous intéressons dans la suite de ce chapitre aux principaux travaux effectués dans ce cadre tout en essayant de dégager les stratégies de parallélisation qui ont été adoptées à cet égard.

8.1 Algorithmes parallèles de calcul de la LPE par immersion

Dans la section 4.1 du chapitre 1 deux algorithmes séquentiels de calcul de la LPE par immersion ont été présentés à savoir l'algorithme de Vincent-Soille et l'algorithme de Meyer.

Nous présentons dans ce qui suit les principaux travaux qui se sont intéressés à leur parallélisation.

8.1.1 Parallélisation de l'algorithme de Vincent-Soille

La difficulté de parallélisation de l'algorithme de calcul de la LPE par immersion de Vincent-Soille [9] réside dans le fait que ce dernier ne soit pas basé sur des conditions locales. Dans [56, 81], les auteurs ont proposé une stratégie de parallélisation de cet algorithme: Ils ont exploité le fait que l'algorithme séquentiel [9] peut être étendu pour être appliqué sur les graphes et ils ont dérivé un algorithme alternatif qui permet une implémentation parallèle en se basant sur l'approche de parallélisation par décomposition en domaines.

L'idée principale de l'algorithme alternatif est de diviser le calcul de la LPE en des étapes qui peuvent être exécutées chacune en parallèle. Ainsi, trois étapes sont distinguées:

(i) Transformer l'image à segmenter en un graphe orienté valué appelé *graphe de composantes*;
(ii) Calculer la LPE du graphe obtenu;
(iii) Transformer le graphe étiqueté en une image binaire.

La première étape consiste à transformer une image initiale *I* en un graphe $G^*=(V^*, E^*)$ dans lequel chaque sommet représente une composante connexe associée à un certain niveau de gris. Dans ce graphe, V^* désigne l'ensemble des sommets et E^* l'ensemble des arêtes. Les sommets du graphe représentent les ensembles de composantes connexes maximales qui ont le même niveau de gris. Ces ensembles sont appelés *des composantes de niveaux*. Pour une image discrète *I*, sa grille associée *G* et un pixel $p \in D(I)$, la *composante du niveau h* notée L_h est définie par:

$$L_h = \{C \subseteq T_h \setminus T_{h-1} \; avec \; C \; est \; une \; composante \; connexe \; de \; T_h \setminus T_{h-1}\} \qquad (2.3)$$

Où T_h désigne la *composante inférieure* de l'image *I* à un niveau *h* définie par:

$$T_h = \{p \in D(I) \mid I(p) \leq h\} \qquad (2.4)$$

En se basant sur (2.3), l'ensemble des sommets du graphe G^* est définie comme étant l'ensemble des composantes de niveaux de *I*. Il est donné par:

$$V^* = \bigcup_{h=hmin}^{hmax} L_h \qquad (2.5)$$

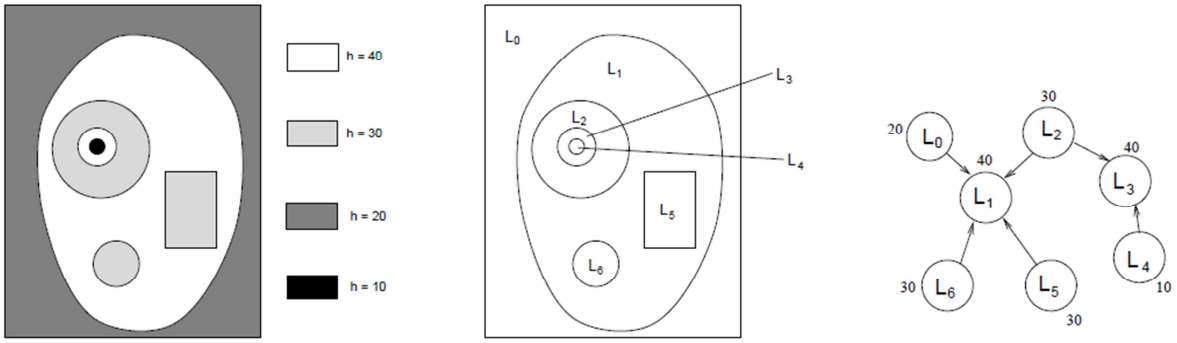
où h_{min} et h_{max} désignent respectivement le niveau de gris minimum et maximum de l'image digitale. Dans le cas des images en niveau de gris, h_{min} =0 et h_{max} =255.

Une paire de composantes de niveaux (v,w) constitue une arête du graphe G^* si il existe un pixel p appartenant à la composante v et un pixel q appartenant à la composante w tel que l'altitude de p est inférieure à celle de q. Cet ensemble est ainsi donné par:

$$(v,w) \in E^* \text{ si et seulement si } \exists\, (p \in v,\, q \in w) \text{ tel que } G \wedge (I(p) < I(q)) \qquad (2.6)$$

La figure 2.4 illustre cette étape de transformation de l'image en un graphe de composantes.

Après avoir exécuté cette étape, l'image représentée sous forme de graphe de composante ne contient plus de plateaux. En effet, un plateau de l'image I est réduit à exactement un nœud du graphe G^*.



(a) (b) (c)

Figure 2. 4 Transformation d'une image de synthèse en graphe de composantes (a) image originale (b) les composantes des niveaux sont étiquetés et (c) graphe de composantes associé à l'image originale [56, 81]

La deuxième étape consiste à calculer la LPE du graphe obtenu par l'étape (i). Elle consiste à attribuer une étiquette à chaque sommet minimum du graphe ainsi qu'à son bassin d'attraction associé, et ce en immergeant itérativement le graphe en utilisant un algorithme de parcours en largeur d'abord [28]. Si un sommet v peut avoir deux ou plusieurs étiquettes différentes, il est étiqueté comme LPE car il est atteint par deux ou plusieurs bassins d'attraction par un chemin descendant. Sinon, si v ne peut être atteint que par des sommets ayant la même étiquette alors cette dernière est attribuée à v. Dans ce cas, le nœud v est dit immergé avec le bassin correspondant.

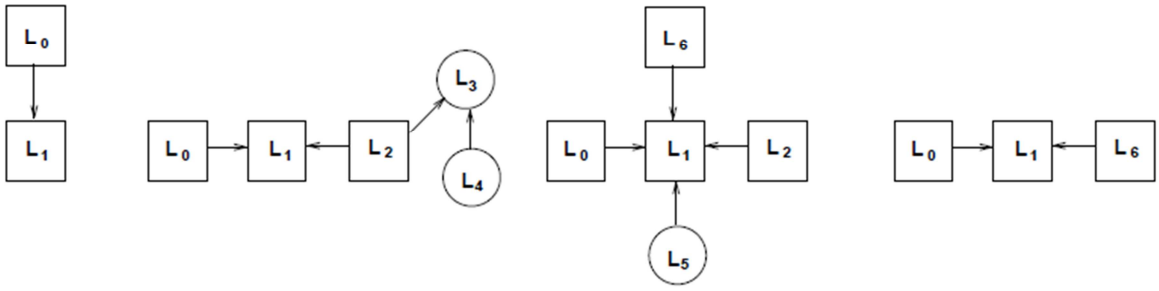
La troisième étape (étape (iii)) consiste à transformer le graphe étiqueté en une image binaire. Dans cette image, les pixels étiquetés comme LPE sont représentés en blanc tandis que les pixels étiquetés comme non LPE sont représentés en noir.

Cet algorithme alternatif [56, 81] peut être parallélisé contrairement à l'algorithme original de calcul de la LPE par immersion [9]. Ceci est dû au fait que les pixels appartenant à une même composante de niveau sont regroupés en un seul nœud du graphe de composantes ce qui permet de décider si un nœud est une LPE en se basant sur des conditions purement locales (seulement les voisins de ce nœud qui ont une valeur inférieure sont examinés). L'algorithme alternatif proposé est parallélisé et implémenté sur une architecture parallèle hybride en anneau (ayant une mémoire distribuée et une partie de mémoire partagée) en utilisant l'approche SPMD. Dans une telle architecture, chaque processeur peut communiquer avec ses deux voisins en utilisant une interface asynchrone de passage par message et possède sa propre mémoire locale pour stocker les parties du code à exécuter et les données locales utilisées.

Dans l'algorithme parallèle, l'étape d'étiquetage des composantes des niveaux est effectuée par un seul processeur car il s'agit d'une étape rapide et sa parallélisation n'entraîne pas un gain significatif en termes de temps (la parallélisation entraîne un gain en termes de temps qui est négligeable par rapport au coût de la communication entre les processeurs).

Ensuite, l'image en entrée ainsi que l'image étiquetée sont distribuées sur P processeurs. La distribution se fait en attribuant, d'une façon statique, à chaque processeur une partie de l'image (appelée *bande*). Les bandes adjacentes sont attribuées à des processeurs voisins. Les bandes sont constituées de lignes consécutives de l'image en entrée et sont chevauchées (chaque bande contient une ligne qui se chevauche avec les bandes associées aux processeurs voisins).

Après, chaque processeur construit un *graphe de composantes local* qui correspond à sa bande associée. Du fait que certaines *composantes de niveaux* sont partagées par plusieurs bandes, les graphes locaux ne sont pas disjoints. La figure 2.5 illustre cette étape. Dans cette figure, l'image représentée dans la figure 2.4 (a) est subdivisée en 4 bandes distribuées sur 4 processeurs. Les sommets partagés sont représentés sous forme rectangulaire. Alors que, les sommets non-partagés sont représentés sous forme de nœuds circulaires.



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

Figure 2. 5 Les graphes de composantes locaux associés aux processeurs (a) P_0 (b) P_1 (c) P_2 et (d) P_3 [81]

Après avoir construit les graphes de composantes locaux, chaque processeur exécute une version modifiée de l'algorithme de Vincent-Soille sur son graphe local. Un nouveau minimum partagé par deux ou plusieurs processeurs doit avoir la même nouvelle étiquette (les processeurs qui partagent ce nœud doivent assigner la même étiquette à ce nœud). Ceci est résolu grâce au partage du tableau contenant les étiquettes attribuées aux nœuds du graphe G^*. i. e, pour chaque $v \in V^*$, si v est étiqueté comme minimum par un processeur P_i, alors cette étiquette est stockée dans le tableau partagé à travers la zone de mémoire partagée. D'autre part, l'extension des bassins est réalisée de la façon suivante : après avoir effectué l'immersion locale à un niveau h, chaque processeur attribue une étiquette locale aux sommets partagés qui est stockée dans la zone de mémoire partagée. Ensuite, chaque processeur récupère ces étiquettes et les compare. Un nœud peut être étiqueté comme LPE ou par un entier positif. Si tous les processeurs ont attribué le même entier à un nœud partagé alors cette étiquette est maintenue. Sinon ce nœud est étiqueté comme LPE.

Enfin, chaque processeur transforme son graphe de composantes correspondant en sous-image binaire. Le résultat de chaque processeur est une bande de la LPE de l'image en entrée.

8.1.2 Parallélisation de l'algorithme de Meyer

La parallélisation de l'algorithme séquentiel de calcul de la LPE par immersion proposé par Meyer [10] a été discuté dans [52] et une implémentation parallèle a été proposée.

Dans la version parallèle proposée [52], l'approche de parallélisation par décomposition du domaine est utilisée. Dans ce cadre, l'image est divisée en des sous-images (ensemble de blocs rectangulaires) non chevauchées de taille quelconque. Le nombre de ces sous-images correspond au nombre de processeurs disponibles. Cette implémentation parallèle est conçue pour une architecture à mémoire distribuée et utilise l'approche SPMD.

L'algorithme parallèle comporte trois principales étapes: la détection des minima; l'étiquetage des minima et l'immersion. Ces étapes sont appliquées sur des sous-images associées aux différents processeurs.

La première étape consiste à parcourir l'image dans ligne par ligne du haut vers le bas et à détecter les pixels non minima (qui possèdent au moins un pixel voisin ayant un niveau de gris inférieur). Ces pixels sont appelés des pixels NARM (*Not A Regional Minimum*). Si un pixel *p* étiqueté comme pixel NARM est détecté, un parcours en largeur d'abord est effectué (*Breadth First Search* noté *BFS*), et ce, afin de propager le label NARM aux pixels voisins ayant le même niveau de gris que *p*. Il est à noter qu'un plateau peut être étendu sur plus qu'une sous-image et que le parcours en largeur d'abord se fait localement par chaque processeur du fait qu'il est limité par les frontières de la sous-image traitée. Une communication entre les processeurs est donc nécessaire afin d'obtenir le résultat adéquat. Ainsi, après avoir détecté les pixels NARM, chaque processeur doit parcourir les pixels situés au niveau des frontières entre les sous-images et une phase de fusion est effectuée. Cette phase consiste à tester s'il existe des pixels voisins appartenant à une sous-image adjacente qui sont de type NARM. Dans ce cas, l'étiquette de ces pixels NARM doit être propagée dans la sous-image adjacente en continuant la recherche en largeur d'abord.

La deuxième étape est basée sur le même principe de parcours de l'image ligne par ligne de haut vers le bas en utilisant l'approche de recherche en largeur d'abord. Elle consiste à attribuer une étiquette unique à chaque minimum régional dans les sous-images locales. Dans cette phase, chaque processeur utilise une plage d'étiquettes disjointes. Comme pour l'étape de détection des minima, l'étiquetage ne peut pas être effectué en se basant sur des conditions purement locales. Ainsi, la communication entre les processeurs voisins est requise sinon des parties (ensemble de pixels) d'un plateau minimum étendu sur plus qu'une sous-image auront des étiquettes différentes. En conséquence, un parcours des frontières entre les sous-images est nécessaire afin de mettre à jour les étiquettes des pixels de ces frontières. La propagation de la mise à jour d'une étiquette dans une sous-image voisine se fait en faisant recourt à une file d'attente.

Enfin, la parallélisation de l'étape de simulation de l'immersion n'est pas triviale vû qu'elle utilise une file d'attente hiérarchique constituée de plusieurs files d'attente. Dans cette étape, chaque processeur P_i maintient une file de priorité locale qui est initialisée avec les pixels minima qui ont un pixel voisin non-minima. Dans l'implémentation parallèle

l'immersion se fait localement dans les sous-images. Elle est ainsi limitée par les frontières de ces dernières. Il est à noter qu'à chaque fois que la propagation des étiquettes atteint une frontière, l'immersion doit se poursuivre dans la sous-image voisine. Pour cela, les auteurs ont proposé qu'à chaque niveau d'altitude, l'inondation est d'abord effectuée à l'intérieur de chaque sous-image et est ensuite suivie par une phase de communication par passage de messages entre processeurs pour propager l'immersion à travers les frontières. Ce cycle d'immersion et de communication est itéré jusqu'à atteindre la stabilité.

L'algorithme parallèle proposé produit un résultat de segmentation très proche de celui produit par l'algorithme séquentiel [10]. Cependant, quelques différences peuvent apparaître et ce, à cause de l'ordre de balayage des pixels qui a changé par rapport à la version séquentielle de l'algorithme. De plus, l'utilisation des files de priorité distribuées entraîne la modification de l'ordre d'immersion des plateaux au niveau des frontières des sous-images.

L'implémentation de cet algorithme parallèle est réalisée sur une architecture à mémoire distribuée (réseau de stations de travail). Dans ce cadre, l'évaluation des temps a montré que le temps d'inactivité présente une partie significative du temps d'exécution total. Ceci est dû entre autres au nombre insuffisant de processeurs disponibles et à la phase de communication qui est coûteuse en termes de temps.

En résumé l'algorithme parallèle de calcul de la LPE proposé dans [52] n'est pas efficace. En effet, la stratégie de parallélisation proposée consiste à diviser la FAH globale en des FAH locales. Ainsi, les pixels appartenant à une file d'attente de la FAH globale sont distribués sur différentes FAH locales, une forte synchronisation entre les processeurs traitant ces files d'attente locales est requise. De plus, l'approche de parallélisation par décomposition du domaine peut entraîner le re-étiquetage d'un même pixel plusieurs fois et ce afin d'attribuer des étiquettes correctes aux parties des bassins versants qui ne se trouvent pas dans le même sous-domaine que leurs minima régionaux associés.

8.1.3 Autres Algorithmes parallèles de calcul de la LPE par immersion

D'autres algorithmes parallèles de calcul de la LPE par immersion ont été proposés dans la littérature parmi lesquels on peut citer ceux de Trieu et al [82,83], André Kôrbes et al [35,86], Bieniek et al [54] et Beucher et al [84], Moga et al [51,55].

L'approche parallèle de calcul de la LPE par parcours séquentiels de l'image proposé dans [55] s'intéresse à la parallélisation de l'algorithme présenté dans [51]. Bien que la version séquentielle de cet algorithme soit assez lente, la méthode présente des propriétés intéressantes dans une implémentation parallèle. Dans cette implémentation chaque processeur parcourt son sous-domaine ligne par ligne du haut vers le bas puis du bas vers le haut. Ces parcours se font d'une manière répétée avec une synchronisation à travers des envois de messages entre les processeurs jusqu'à stabilisation. Les principales étapes de cet algorithme sont : (i) la détection des minima, (ii) l'étiquetage des minima (iii) l'immersion par intégration de l'image.

Les étiquettes au niveau des frontières entre les sous-domaines ont besoin d'être communiqués entre les processeurs. Même si le calcul s'est stabilisé dans un sous domaine, un nouveau parcours peut être nécessaire à cause des changements au niveau des frontières causés par les autres processeurs. Un thread master supervise le tout et détecte quand la stabilisation globale est obtenue.

D'après les expérimentations menées par les auteurs, une accélération approximativement linéaire proche de la courbe idéale est obtenue dans plusieurs cas. Mais l'efficacité diminue lorsque le nombre de processeurs devient grand. Les images de grandes tailles conduisent à une accélération plus importante pour le même nombre de processeurs.

Un autre algorithme parallèle de calcul de la LPE par immersion a été proposé dans [83, 94]. Cet algorithme est basé sur le calcul de composantes connexes [82].

Dans cet algorithme, les pixels d'une image sont scannés à plusieurs reprises en partant du coin haut-gauche vers le coin bas-droit de l'image et ensuite à partir du coin bas-droit vers le coin haut-gauche pour propager la valeur de chaque pixel à ses voisins. Il peut être résumé en trois principales étapes :

(i) Marquer chaque pixel de l'image qui possède un voisin ayant une altitude inférieure.

(ii) Pour chaque pixel qui n'est pas marqué (i.e. qui appartient à un plateau), la distance minimale qui le sépare du plateau d'altitude inférieure est calculée.

(iii) Trois cas sont possibles :

- Pour chaque pixel non marqué et n'ayant pas de distance calculée (ce pixel appartient à un plateau correspondant à un minimum régional), si tous ses

voisins dans le même plateau ne sont pas encore étiquetés, une étiquette unique est attribuée à ce pixel. Sinon, une des étiquettes est copiée et est attribuée à ce pixel.

- Pour chaque pixel marqué, l'étiquette de son voisin à altitude minimale est copiée.
- Pour chaque pixel ayant une distance calculée, l'étiquette de l'un de ses voisins est copiée.

Ces trois étapes peuvent être exécutées d'une façon continue lors d'un même parcours de l'image. Ainsi, des fausses étiquettes peuvent être attribuées aux pixels mais ces dernières vont être écrasées et rectifiées par des étiquettes correctes lors des parcours suivants.

Il est à noter que cet algorithme est basé sur l'algorithme de SunYang et al [85] mais qui substitue l'utilisation des files d'attente (utilisées pour la synchronisation) par le processus de calcul de distance géodésique et d'étiquetage.

L'algorithme parallèle proposé dans [82, 83] est implémenté sur carte FPGA. Une telle implémentation sur un circuit standard (ADM-XRC-II par Alpha Data) avec une carte Xilinx XC2V6000 garantit un gain en termes de temps qui rend cet algorithme adapté au calcul temps réel. En effet, la performance du circuit est plus rapide que 30 frames/s même dans le cas où une seule ligne de l'image est traitée à la fois. Dans le cas où 4 lignes sont traitées en parallèle, la performance peut être améliorée de 2.5 à 3 fois. Cette performance est environ 10 fois meilleure qu'une implémentation software [106] sur un Pentium à 4 processeurs avec 2.4 GHz et est suffisamment rapide pour la segmentation des images en temps réel [82].

Un autre algorithme parallèle de calcul de la LPE a été proposé par André Kôrbes et al dans [35, 86]. Cet algorithme peut être résumé en quatre principales étapes :

(i) Pour chaque pixel, trouver le voisin d'altitude la plus petite (pour pouvoir calculer le chemin descendant de plus grande pente) ;
(ii) Pour les pixels internes des plateaux, trouver la frontière la plus proche ;
(iii) Etiquetage des minima ;
(iv) Etiquetage des pixels par inondation à partir des minima.

8.2 Algorithmes parallèles de calcul de la LPE par ruissellement

8.2.1 Algorithme parallèle de calcul de la LPE par ascension de collines

Dans [107], Zhou et al ont proposé la parallélisation de l'algorithme de calcul de la LPE par distance topographique en utilisant l'approche par décomposition de domaine. L'algorithme séquentiel de calcul de la LPE par distance topographique concerné par cette parallélisation peut être résumé en deux principales étapes:

- Une première étape de calcul des minima régionaux de l'image en entrée. Les pixels minima sont nommés des *pixels germes*;
- Une deuxième étape de croissance de régions basée sur la notion de *distance inférieure* définie comme suit: Pour une image numérique en niveau de gris *I*, définie dans un domaine $D \subseteq \mathbb{Z}^2$, à la quelle est associée une grille *G*, la distance inférieure nommée *d* est donnée par:

$$d(p) = \begin{cases} 0, \ \ si\ p\ est\ un\ minimum\ régional \\ longueur\ du\ plus\ court\ chemin\ de\ p_0\ \text{à}\ p_s | \forall\ i \in \{1,2,\dots,s\}(p_{i-1},p_i) \in G, I(p_s) < I(p_0) \\ si\ s > 1, \forall i \in \{1,2,\dots,s-1\}, I(p_i) = I(p_{i-1}) \end{cases} \quad (2.7)$$

Chaque pixel non minima est inséré dans un bassin d'attraction. Les pixels sont traités dans l'ordre croissant de leurs niveaux de gris. La phase de propagation de l'étiquetage nommée phase d'immersion se fait d'une manière récursive. Pour ce faire, une relation d'ordonnancement est définie pour pouvoir l'exploiter dans la parallélisation de cette phase. Cette relation satisfait deux conditions:

- condition 1: si $I(p_s) = min_{\,r \in N_{G(p)}}\{I(r) \setminus I(r) < I(p_0)$ alors p_s est le pixel antérieur de p_0. Dans ce cas, p_0 est dit immergé par p_s et $L(p_0) = L(p_s)$.
- condition 2: si $I(p_0) = I(p_s)$ et $d(p_0) = d(p_s) + 1$ alors $L(p_0) = L(p_s)$.

Où *L* désigne l'image de sortie qui associe, à chaque pixel, une étiquette.

Une étiquette unique est assignée à chaque pixel situé au niveau des frontières de l'image et ayant un pixel antérieur. Ainsi, ces pixels sont considérés comme des pixels pseudo-germes. En se basant sur la relation d'ordonnancement définie ci-dessus, chaque processeur peut délimiter l'étendue des régions dans leurs sous-domaines locaux indépendamment des autres processeurs.

De plus, les auteurs ont défini un *graphe de composantes frontières* (désigné dans ce qui suit par BCG (*Boundary Component Graph*). Ce graphe assure l'archivage de la relation d'ordonnancement entre les pixels pseudo-germes et leurs pixels antérieurs pour la phase d'immersion et de fusion dans les étapes suivantes. En effet, si une image *I* est considérée comme un graphe valué direct *G(V,E,F)* où *V* désigne l'ensemble des pixels et *E* l'ensemble des arêtes qui correspondent à la relation d'adjacence utilisée, un BCG nommé *G* (V*, E*, F*)* peut être défini comme suit :

- *F*=F* ;
- si $v \in V \wedge L(v) = H$ ou $L(v) \neq H \wedge (\exists\, p, p \in V \wedge p \in N_G(v) \wedge L(p) = H)$ alors $v \in V^*$.
- $\forall\, u \in V^* et\ v \in V^*$, si $L(u) \neq H \wedge L(v) = H$ et *u* et *v* satisfont la condition 1 ou la condition 2 de la relation d'ordonnancement alors $\{u, v\} \in E^*$.
- $\forall\, u \in V * et\ v \in V *$, si $L(u) \neq H \wedge L(v) = H \wedge F^*(u) = F^*(v) \wedge d(u) = d(v) = \infty$ alors $\{u, v\} \in E^*$.

Il est à noter que le BCG est lié uniquement aux pixels situés au niveau des frontières de chaque sous-domaine et non pas à tous les pixels de l'image d'entrée. Ainsi, la taille du graphe est réduite.

L'algorithme parallèle proposé dans [107] consiste donc à diviser le domaine global *D* de taille *M*×*N* sur *P* processeurs en des sous-domaines D_i. Cet algorithme peut être résumé en 4 principales étapes :

i) Détection des pixels germes et des pixels pseudo-germes et construction des BCG locaux.
ii) Immersion locale : dans cette étape une file de priorité est utilisée pour calculer la LPE locale.
iii) Fusion globale : dans cette étape, la même approche utilisée dans [56,57] pour la fusion des graphes de composantes est considérée.
iv) Diffusion des résultats fusionnés à chaque processeur pour mettre à jour les résultats locaux

Cet algorithme parallèle est implémenté sur deux types d'architectures : un cluster avec 16 nœuds et un super-computer Yinhe avec 32 processeurs. Des images de différentes tailles sont utilisées pour les tests. Les expérimentations menées ont montré que les résultats obtenus

suite à l'implémentation de l'algorithme parallèle sur la première architecture sont meilleurs que ceux obtenus suite à son implémentation sur la deuxième architecture.

8.2.2 Algorithme parallèle de calcul de la LPE par ascension de collines combinée à un opérateur de composantes connexes

La parallélisation de l'algorithme par ascension de collines combiné avec un opérateur de composantes connexes a été considérée par Bieniek et al. [54] en utilisant une condition locale. La principale idée est de résoudre le problème de la LPE indépendamment dans tous les sous-domaines sans synchronisation. En effet au lieu de la synchronisation, des étiquettes temporaires sont assignées aux pixels qui seront inondés à partir des sous domaines adjacents. L'information relative à la connexité au niveau des frontières entre les sous domaines est stockée dans une table d'équivalence.

Les étiquettes globales sont calculées par une opération de réduction utilisant une étape de résolution inspirée de l'algorithme UNION-FIND. Les expérimentations menées par les auteurs dans [54] ont montré l'obtention d'une accélération linéaire pour un nombre de processeurs croissant. La saturation est obtenue à partir de 64 processeurs.

8.2.3 Algorithme parallèle de calcul de la LPE par ruissellement sur image complètement abaissée

Dans [57] les auteurs ont proposé un algorithme parallèle de calcul de la LPE par distance topographique en utilisant l'approche de décomposition fonctionnelle. L'algorithme proposé est adapté à une implémentation sur architecture à mémoire partagée. Il est basé sur une version parallèle et adaptée de l'algorithme de Djikstra [67] qui permet de calculer le plus court chemin d'un nœud au reste des nœuds dans un graphe indirect.

Rappelons que dans une première phase, l'image d'entrée est transformée en une image dans laquelle tout pixel non minimum possède au moins un voisin ayant un niveau de gris inférieur. Cette transformation est effectuée en utilisant des files FIFO. Les files sont réparties sur l'ensemble des processeurs. Chaque processeur initialise sa propre FIFO locale avec des points germes (pixels qui sont à la limite inférieure d'un plateau). Après cette initialisation, chaque processeur commence à propager les distances dans une image locale nommée '*dist*'. Lorsque tous les processeurs ont terminé cette opération, le minimum sur toutes les images '*dist*' est l'image complètement abaissée souhaitée. Le calcul de ce minimum peut être

efficacement effectué en parallèle en utilisant un opérateur de réduction. Après le calcul de l'image abaissée, la représentation de cette image sous forme d'un graphe indirect est effectuée en parcourant l'image une seule fois. Dans ce parcours, seuls les voisins de chaque pixel sont visités. Aucune dépendance entre les pixels n'existe en ce qui concerne l'ordre de calcul ce qui rend cette opération facilement parallélisable. Il est à noter qu'initialement, un processeur commence à travailler sur des pixels dans son domaine privé, mais pendant la phase de fusion il accède aux pixels dans les domaines d'autres processeurs.

8.2.4 Autres algorithmes parallèles de calcul de la LPE par ruissellement

La difficulté de parallélisation de la LPE est due à plusieurs facteurs parmi lesquels il est possible de citer le flux de traitement. En effet, dû à la nature de la LPE, l'ordre dans lequel les pixels sont examinés dépend uniquement des caractéristiques de l'image ce qui rend difficile la division arbitraire de l'image en des sous-images plus petites qui peuvent être traitées indépendamment. Une telle approche nécessite généralement une synchronisation complexe et des mécanismes de communication entre les processeurs. Dans [88] les auteurs ont proposé une solution à cette problématique. Ils ont proposé une nouvelle stratégie de parallélisation basée sur l'approche "*shared nothing*" appliquée sur un algorithme modifié de calcul de la LPE topographique. Cette stratégie permet de partitionner la majorité des tâches en des sous-tâches indépendantes qui peuvent être exécutées en parallèle sur différents processeurs. Cette approche élimine le besoin à utiliser des structures de données complexes pour assurer la synchronisation entre les processeurs.

Un autre algorithme parallèle de calcul de la LPE topographique a été proposé par Moga et al dans [51, 53]. Cet algorithme procède en deux grandes étapes : la détection des minima suivie d'un étiquetage des pixels avec une technique qui simule le ruissellement de l'eau. La détection des minima nécessite une communication répétitive jusqu'à la stabilisation pour aboutir au résultat global.

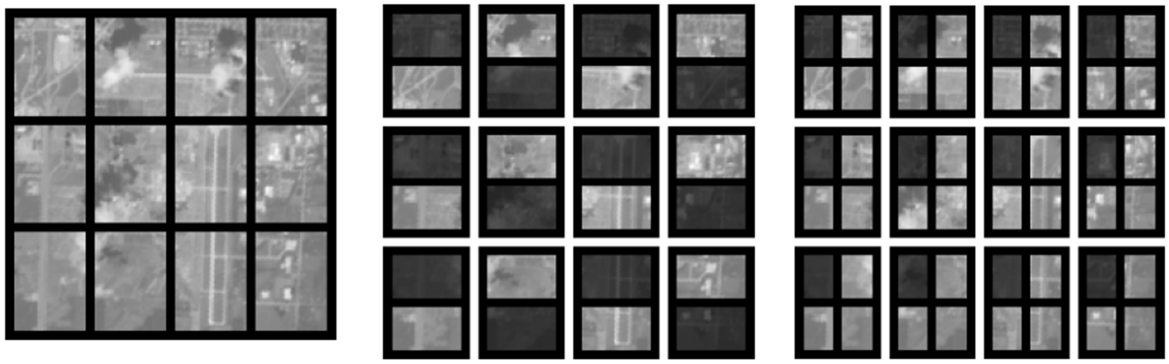
L'étape de ruissellement est considérée comme un étiquetage de chaque point par l'étiquette du minimum auquel il est connecté par un chemin descendant. Dans le cas de l'existence de plusieurs voisins équidistants en même temps le graphe orienté acyclique qui représente l'image est transformé en un ensemble de forêts disjointes. Ceci permet de réduire la non-localité du traitement mais introduit une dépendance par rapport à l'ordre du parcours. Cet algorithme est implémenté sur une architecture multi-cœurs à mémoire distribuée.

8.3 Algorithme parallèle de calcul de la LPE topologique

La parallélisation de l'algorithme séquentiel [18] de calcul de la LPE topologique a été discutée dans [98] et un algorithme parallèle a été proposé.

Pour rappel, l'algorithme séquentiel [18] consiste à abaisser itérativement les points W-destructibles autant que possible (à leurs valeurs minimales). La valeur minimale à laquelle un point peut être abaissé est calculée dans l'arbre de composantes (nommé $C(\overline{F})$) associé au complément de la fonction de valuation de l'image en entrée. De ce fait, la parallélisation de l'algorithme de calcul de la LPE topologique nécessite la parallélisation de l'étape de calcul de l'arbre de composante. Ce problème a été traité dans [95] et un algorithme parallèle de calcul de cet arbre a été proposé.

L'étape suivante consiste à abaisser tous les points W-destructible à la valeur de leurs bassins versants. L'algorithme séquentiel fait appel à une fonction clé nommée « W-Destructible » qui permet de déterminer à quelle altitude un pixel peut être abaissé. Pour cela, cette fonction utilise la valeur du plus bas ancêtre en commun (*Lowest Common Ancestor*) des composantes voisines dans l'arbre de composantes. Cette fonction a la particularité d'être locale. Ainsi, il est possible de paralléliser l'algorithme séquentiel de calcul de la LPE topologique par la division de l'image en entrée en N blocs traités par N processeurs tel que chaque bloc est assigné à un processeur. Un problème s'impose au niveau des pixels frontières car leurs voisins appartenant aux autres blocs (adjacents) peuvent être modifiés par leurs processeurs associés. Ceci peut produire un faux résultat si une phase de fusion n'est pas appliquée. La solution proposée par les auteurs pour résoudre ce problème est de permettre à chaque processeur de traiter son bloc associé sur différentes étapes. Pour cela, chaque bloc est divisé en des sous-blocs de telle sorte que les sous-blocs adjacents ne sont pas traités en parallèle. La figure 2.6 illustre le cas d'une image bidimensionnelle divisée en 12 blocs. Tenant compte de la 4-adjacence (respectivement 8-adjacence), chaque bloc est divisé en deux sous-blocs (respectivement 4 sous-blocs). A chaque étape, les sous-blocs non adjacents sont traités en parallèle. Dans la figure 2.6 (b) les sous-blocs en gris foncé peuvent être traités en parallèle car ils ne sont pas adjacents. Il est à noter que plusieurs itérations sont requises pour obtenir le résultat adéquat.



(a) (b) (c)

Figure 2. 6 Division d'une image 2D (a) en 12 blocs et chaque bloc est divisé en 2 sous-blocs dans le cas de la 4-adjacence (b) et en 4 sous-blocs dans le cas de la 8-adjacence (c) [98]

D'autre part, l'algorithme séquentiel [18] utilise une file de priorité pour stocker les pixels. Ainsi, les pixels sont traités selon leur ordre de priorité. Dans l'implémentation parallèle, cette étape est effectuée seulement dans la première itération. Ensuite, dans le reste des itérations, seuls les pixels qui ont changé de valeurs et voisins des pixels appartenant aux autres sous-blocs adjacents sont ajoutés à la file de priorité. Pour cela, une image binaire nommé « *pxchanged* » est utilisée. Dans cette image l'état de chaque pixel est enregistré (si sa valeur a changé récemment ou non). La figure 2.7 illustre l'utilisation de cette image.

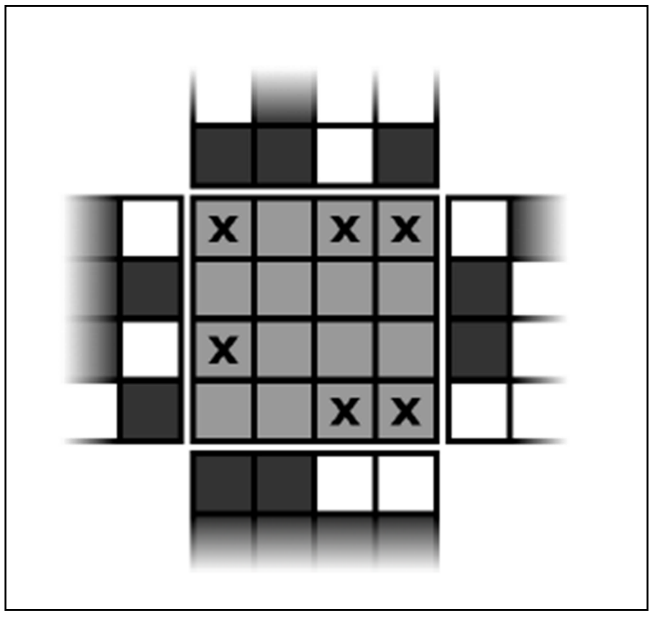


Figure 2. 7 Un exemple d'image « pxchanged » à une itération k avec k >1. Les pixels blancs dans les sous-blocs adjacents sont les pixels marqués comme « changed » dans « pxchanged ». Les pixels dans le sous-bloc courant qui sont adjacents à de tels pixels sont marqués avec un X et sont ajoutés à la file de priorité [98]

En résumé, l'algorithme parallèle peut être résumé en deux principales procédures : « InitialiserQueue » et « TopologicalWshdBloc ». Les algorithmes de ces deux procédures sont présentés et détaillés dans ce qui suit.

L'algorithme de la procédure « InitialiserQueue » prend comme entrée :

- l'image initiale nommée *F* (une image est représentée sous forme de GSP tel que *F* désigne la fonction de pondération des sommets qui correspond au niveau de gris des pixels),
- l'arbre de composantes $C(\overline{F})$ et sa fonction de composantes associée nommée ψ (qui associe à chaque point le nœud auquel il appartient dans $C(\overline{F})$),
- le sous-bloc *T* dans lequel l'algorithme doit être appliqué et le statut courant de la fonction « pxchanged ».

Algorithme 7 : InitialiserQueue [98]

```
   Data : (F, C(F̄), ψ, T, pxchanged)
   Result : (L, K, H, pxchanged)
 1 // Initialisation
 2 for (k de k_min à k_max − 1 do
 3 |  L_k:=φ;
 4 foreach (pixel p ∈ T) do
 5 |  pxchanged[p]:= FALSE;
 6 if ( première itération ) then
 7 |  foreach ( pixel p ∈ T) do
 8 |  |  c:=W-destructible(F,p,C(F̄), ψ); if (c ≠ φ) then
 9 |  |  |  i:=niveau de c; L_i:=L_i ∪ {p};
10 |  |  |  K(p):=i;; H(p):= pointer sur c;
11 |  |  else
12 |  |  |  foreach (pixels p appartenant aux frontières de T) do
13 |  |  |  |  addP:=FALSE; foreach ( voisins q de p) do
14 |  |  |  |  |  if (pxchanged[q] == TRUE) then
15 |  |  |  |  |  |  addP:=TRUE;
16 |  |  |  |  if (addP==TRUE) then
17 |  |  |  |  |  c:=W-destructible(F,p,C(F̄), ψ); if (c ≠ φ) then
18 |  |  |  |  |  |  i:=niveau de c; L_i:=L_i ∪ {p};
19 |  |  |  |  |  |  K(p):=i;; H(p):= pointer sur c;
```

Si un pixel est W-destructible il est ajouté à la file de priorité et son niveau de priorité est fixé au niveau auquel ce pixel peut être abaissé. Par ailleurs, ce nouveau niveau est enregistré dans la fonction *K* et un pointeur sur la composante à laquelle le pixel peut être ajouté est enregistré dans la fonction *H*. Le pixel lui-même n'est pas encore abaissé. La file *L* et les

fonctions *K* et *H* sont locales, cependant, « pxchanged » est globale et elle peut être lue et modifiée par différents processeurs au cours de l'exécution de la procédure. Cependant, chaque processeur modifie seulement la partie de l'image « pxchanged » qui correspond à son sous-bloc associé et n'accède qu'aux parties qui correspondent aux sous-blocs qui sont traités dans différentes étapes (sous-blocs indépendants) ce qui évite les conflits et résout le problème de synchronisation entre les processeurs.

L'algorithme appliqué à chaque sous-bloc de l'image commence par l'initialisation d'une part, de la file de priorité (ligne 1) et d'autre part, de l'image « pxchanged » (attribuer la valeur FALSE à chaque pixel de « pxchanged » dans le sous-bloc en cours de traitement). Dans la première itération, la file de priorité *L* est initialisée comme dans le cas de l'algorithme séquentiel [18] sauf que, seulement les pixels appartenant au bloc *T* sont considérés au lieu de traiter tous les pixels de l'image dans le cas séquentiel. Ensuite, pour le reste des itérations (ligne 9), tous les pixels appartenant aux frontières des sous-blocs sont parcourus un par un pour détecter si un ou plusieurs de leurs voisins ont changé de valeur. Dans ce cas, si le pixel de frontière en question est W-destructible alors, il est ajouté à la file de priorité et les fonctions *K* et *H* sont mises à jour comme expliqué précédemment.

Algorithme 8 : TopologicalWshdBloc [98]

```
   Data : (F, C(F̄), ψ, T, pxchanged)
   Result : (F, pxchanged, anychanges)
1  // Initialisation
2  anychanges:=FALSE;
3  InitialiserQueue(F, C(F̄), ψ, T, pxchanged);
4  for (k de k_min à k_max − 1) do
5      while (∃p ∈ L_k) do
6          L_k:=L_k \ {p};
7          if (K(p) == k) then
8              F(p):=k; ψ(p):=H(p);
9              pxchanged[p]:=TRUE; anychanges:=TRUE;
10             foreach (voisins q de p appartenant à T avec k < F(q)) do
11                 c:=W-destructible(F,p,C(F̄), ψ);
12                 if (c ≠ ϕ) then
13                     K(q):= ∞;
14                 else
15                     i:= niveau de c;
16                     if (K(q) ≠ i) then
17                         L_i:=L_i ∪ {q}; K(q):=i;
18                         H(q):=pointer sur c;
```

L'algorithme de la deuxième procédure « TopologicalWshdBloc » prend en entrée les même arguments que l'algorithme de la procédure « InitialiserQueue » et produit comme résultat : l'image *F* mise à jour, l'image « pxchanged » mise à jour et la variable binaire « anychanges ». Cette dernière est utilisée pour détecter rapidement si un changement a eu lieu dans le sous-bloc durant l'exécution de la procédure.

Cette procédure commence par initialiser la valeur de la variable « anychanges » ainsi que la file de priorité *L* (il est à noter que l'appel de la procédure « InitialiserQueue » à la ligne 1 conduit aussi à l'initialisation des fonctions *K* et *H*). Ensuite, le même traitement réalisé par l'algorithme séquentiel est effectué sauf qu'une mise à jour de la variable « anychanges » et l'image « pxchanged » est ajoutée (ligne 8). De plus, dans la ligne 9, seulement les voisins de *p* qui appartiennent au sous-bloc *T* doivent être ajoutés à la file de priorité.

Algorithme 9 : Algorithme parallèle de calcul de la LPE Topologique[98]

```
   Data : (F, C(F̄), ψ, pxchanged, id)
   Result : (F)
1  done:=FALSE;
2  while (not done) do
3      anychangesThrd[id]:=FALSE;
4      foreach (étape s) do
5          T:=sous-bloc courant en se basant sur id et s;
6          TopologicalWshdTile(F, C(F̄), ψ, T, pxchanged);
7          if (anychanges==TRUE) then
8              anychangesThrd[id]:=TRUE;
9          Barrière();
10     if (id = 0) then
11         anychangesAtAll:=FALSE;
12         foreach (thread t) do
13             if (anychangesThrd[t]==TRUE) then
14                 anychangesAtAll:=TRUE;
15     Barrière();
16     if (anychangesATAll==FALSE) then
17         done:=TRUE;
```

Après avoir présenté les algorithmes des deux procédures « TopologicalWshdBloc » et « InitialiserQueue », l'algorithme parallèle de calcul de la LPE topologique peut être donc présenté. Cet algorithme prend comme entrée l'image en entrée (représentée sous forme de fonction de pondération *F*), l'arbre de composantes $C(\bar{F})$ et sa fonction de composantes associée ψ, l'image « pxchanged » et l'identifiant de chaque processeur. Il est à noter que l'image « pxchanged » est globale et doit être fournie à tous les processeurs car chaque

processeur exécute cet algorithme indépendamment. L'algorithme s'exécute jusqu'à ce qu'une LPE topologique de l'image en entrée soit obtenue. A chaque itération, la boucle de la ligne 2 de l'algorithme 8 est exécutée par chaque processeur. La boucle de la ligne 4 de cet algorithme est exécutée à chaque étape. Il est à noter que le nombre d'étapes est égal au nombre de sous-blocs assignés à chaque processeur. Le sous-bloc à traiter par le processeur courant est déterminé (calcul de l'emplacement et de la dimension) à chaque étape (ligne 5). La LPE topologique de chaque sous-bloc est donc calculée à la ligne 6 en faisant appel à la procédure « TopologicalWshdBloc» détaillée ci-dessus. Si cette dernière indique qu'il y a des pixels changés pour le processeur courant alors cette information est stockée dans un tableau global nommé « anychangesThrd ». Ensuite, tous les processeurs sont synchronisés en utilisant barrière. Cette dernière est utilisée afin de garantir qu'aucun thread ne commence à traiter un autre sous-bloc (à l'étape suivante) avant que le reste des processeurs ne terminent le traitement du sous-bloc courant (à l'étape courante). Quand tous les processeurs finissent le traitement de tous leurs sous-blocs associés, le premier thread vérifie s'il y a des changements (retourné par n'importe quel processeur). Dans ce cas, chaque processeur passe à l'itération suivante et ré-exécute la boucle de la ligne 2. Sinon, s'il n'y a aucun changement détecté alors une LPE topologique est calculée.

L'algorithme séquentiel [18] et l'algorithme parallèle [98] de calcul de la LPE topologique sont implémentés sur une machine AMD Opteron. Les expérimentations ont montré que la parallélisation des étapes de construction de l'arbre de composantes et de calcul de la LPE topologique par blocs est efficace et a produit un facteur d'accélération entre 10 et 14 pour des images de différentes tailles en utilisant 24 threads. Par ailleurs, la totalité de l'algorithme parallèle de calcul de la LPE topologique possède un facteur d'accélération entre 10 et 12 pour des images de différentes tailles et en utilisant 24 threads.

8.4 Algorithme parallèle de calcul de la LPE d'arêtes

Comme nous l'avons mentionné dans la section 4.5 du chapitre 1, deux algorithmes linéaires de calcul de la LPE d'arêtes ou coupures par LPE sont présentés : le premier basé sur le calcul d'amincissement nommé algorithme de calcul de noyau par M-bord [30] et le second est basé sur le calcul de flux d'une façon récursive et est nommé algorithme de calcul de partition par flux [8]. A notre connaissance, la parallélisation de l'algorithme de calcul de noyau par M-bord n'a pas été traitée dans la littérature. La parallélisation du second

algorithme (algorithme de calcul de partition par flux [8]) a été discutée dans [96, 97] et un algorithme parallèle a été proposé.

L'algorithme séquentiel de calcul des coupures par LPE à base de calcul de flux [8] est appliqué dans le cadre des graphes à arêtes valuées et il repose sur l'extraction de flux. Cet algorithme est présenté dans la section 8.2.1.2 du chapitre 1. La difficulté de parallélisation de cet algorithme réside d'une part, dans le fait qu'il soit récursif et d'autre part du fait qu'il mélange des itérations en largeur et en profondeur. La version parallèle de cet algorithme (Algorithme 10) proposée dans [96, 97] consiste en deux principales étapes:

(i) Une étape de calcul en parallèle des flux des différents sommets du graphe;

(ii) Une étape de fusion des flux deux par deux. Dans ce contexte, deux cas sont possibles : ou bien il n'existe aucun puits en commun entre deux sources (pas de sommets en commun entre les deux flux à fusionner) ou bien il existe des puits en commun entre deux sources (les deux flux à fusionner contiennent un sommet en commun). Dans le second cas, il existe différentes configurations possibles.

Dans la première étape, une étiquette est attribuée, en parallèle, à chaque sommet x du graphe. En effet, à partir de chaque point non encore étiqueté, un flux L composé des points non étiquetés et ayant x comme source est calculé. Les sommets constituant un $\prec$-*flux* possèdent la même étiquette. Il est à noter que le calcul des flux est totalement indépendant. Ainsi le parallélisme massif est appliqué.

La deuxième étape consiste à fusionner les différents flux calculés. Cette étape présente un problème majeur qui réside dans le choix de l'étiquette à attribuer aux points appartenant à deux flux qui partagent le même puits. La solution proposée par les auteurs est inspiré du paradigme d'inondation et consiste à étudier les différents cas possibles de fusion de deux flux d'eaux jaillissant de différentes sources et identifier quel flux atteindra le puits le premier. C'est ce flux qui attribuera son étiquette au puits en question.

Le point de départ est l'approche de descente de plus grande pente avec les conditions suivantes : le débit d'écoulement d'eau est le même pour toutes les sources, la surface sur laquelle s'écoule l'eau est parfaitement lisse et la vitesse de ruissellement est uniforme pour chaque flux. Tenant compte de ces trois conditions, trois facteurs entrent en jeu pour déterminer la vitesse du flux : l'altitude de la source, la pente et la distance entre la source et

le puits. Ainsi, cinq configurations peuvent être distinguées. Cependant, dans la pratique, certaines configurations peuvent ne pas se produire lors de la fusion des flux :

- Si les deux flux à fusionner sont deux ↛-flux alors il faut créer une nouvelle étiquette et l'attribuer aux puits.
- Si un flux parmi les deux à fusionner contient un ↛-flux alors une étiquette qui existe déjà est attribuée aux puits communs.
- Si les deux flux à fusionner contiennent un ↛-flux alors il faut choisir quelle étiquette parmi les deux étiquettes qui existent déjà doit être conservée. Ainsi, il faut calculer approximativement l'accélération moyenne du flux en utilisant la formule de Chezy. Il s'agit d'une formulation mathématique de l'accélération moyenne du flux introduite en 1769 [103] donnée par:

$$V_c = c(h * S)^{1/2} \qquad (2.8)$$

Où *S* désigne la pente, *h* désigne l'altitude de la source et *c* désigne le coefficient de rugosité.

Il est à noter que le niveau de gris d'un pixel représente son altitude. Ainsi, la pente et la distance entre les sources et les puits peuvent être calculées en utilisant les coordonnées des pixels. D'autre part, le coefficient de rugosité dans ce cadre est égal à 1.

Les deux fonctions intitulées « f_labeling » et « s_labeling » résument les cas de fusion possibles.

Algorithme 10 : Algorithme parallèle de calcul des Coupures par flux [97]

Data : (V, E, F) un graphe à arête valuées
Result : Ψ une partition par flux de F.

```
1  foreach ( x ∈ V ) do in parallel
2      Ψ(x) := NO_LABEL; // pas de dépendance de données donc parallélisme massif
3  nb_labs:=0;  // variable globale et partagée
4  i:=0;  // flux traité
5  // lancer N process en parallèle
6  foreach ( x ∈ V tel que Ψ(x) = NO_LABEL) do in parallel
7      [L_i, lab_i]:= Flux(V, E, F, Ψ, x_i);  // calculer le flux associé à chaque x_i
8  nb_fusion:=i;
9  while (nb_fusion ≠ 1) do
10     // lancer nb_fusion/2 process en parallèle
11     for (j=0, j<nb_fusion, j+=2) do
12         if (L_j ∩ L_{j+1} = φ) then
13             s_labeling([L_j, lab_j], nb_labs);
14             s_labeling([L_{j+1}, lab_{j+1}], nb_labs);
15         else
16             f_labeling([L_j, lab_j], [L_{j+1}, lab_{j+1}], nb_labs);
17 nb_fusion:=nb_fusion/2;
```

Function *f_labeling* **[97]**($L_a, lab_a, L_b, lab_b, nb\_labs$):

```
//L_a et L_b sont deux inf-flux
if (lab_a=(-1) et lab_b=(-1) ) then
    nb_labs++;
    Attribuer_lab(L_a, L_b, nb_lab);
//L_a ou L_b contient un inf-flux déjà étiqueté
else if (lab_a ≠ (-1) et lab_b=(-1)) then
    Attribuer_lab(L_a, L_b, lab_a);
else if (lab_a = (-1) et lab_b ≠(-1)) then
    Attribuer_lab(L_a, L_b, lab_b);
//L_a et L_b contiennent deux inf-flux déjà étiquetés
else if (V_c(L_a) ¿ V_c(L_b))) then
    Attribuer_lab(L_a, L_b, lab_a);
else
    Attribuer_lab(L_a, L_b, lab_b);
retourner NULL;
```

```
Function Attribuer_lab [97]( (L1, L2, lab) ):
    foreach (z ∈ {L1 ∩ L2}) do
        ψ(z):=lab;
    foreach (x ∈ L1 tel que ψ(x)==NO_LABEL) do
        ψ(x):=lab;
    foreach (y ∈ L2 tel que ψ(y)==NO_LABEL) do
        ψ(y):=lab;
    retourner NULL;
```

Dans [97], les auteurs ont évalué l'algorithme parallèle de calcul de la LPE d'arêtes proposé et implémenté sur des architectures multi-cœurs à mémoire partagée en termes de temps, d'efficacité et de scalabilité. En effet, les auteurs ont utilisé un ensemble de processeurs Intel couramment utilisé ayant un nombre de processeurs qui varie de 1 à 8. Les expérimentations montrent que l'accélération augmente lorsque le nombre de cœurs augmente pour atteindre 6.11 pour 8 CPUs (et 8 threads). De plus, elle est améliorée lorsque la taille de données à traiter augmente. De plus, l'accélération augmente avec l'augmentation du nombre de threads. Cependant, quand le nombre de threads dépasse le nombre de cœurs disponibles, le temps cesse de diminuer.

L'efficacité diminue de 30% quand on passe d'une architecture mono-cœur à une architecture à deux cœurs. Pour 4 cœurs, l'efficacité augmente légèrement mais reste inférieure de 20% de celle mesurée pour 1 CPU. Ceci peut être dû, entre autres, à la décélération du calcul parallèle : les délais des I/O entraînés par la nécessité de distribuer les données à traiter en parallèle sur les éléments de traitement locaux et/ou les délais de communication dus au fait que les éléments de traitement peuvent demander l'accès à des données qui ne sont pas locales.

Par ailleurs, les expérimentations menées par les auteurs ont montré qu'un bon facteur de scalabilité de l'algorithme parallèle est noté pour les différentes architectures de test (ayant 2, 4 et 8 cœurs).

8.5 Bilan sur les algorithmes parallèles de calcul de la LPE

Afin d'évaluer les différents algorithmes présentés dans ce chapitre, nous avons sélectionné un ensemble de critères qui sont en relation avec, d'une part, la stratégie de parallélisation utilisée et d'autre part, l'implémentation sur architecture matérielle.

Ces critères sont :

- Approche de parallélisation basée sur la décomposition du domaine ou décomposition fonctionnelle,
- Type d'architecture sur laquelle est implémenté l'algorithme parallèle : dans ce bilan nous nous limitons à comparer les algorithmes parallèles implémentés sur architectures multi-cœurs à mémoire partagée, distribuée ou hybride,
- Nécessité de synchronisation et/ou communication entre les processeurs ou les threads,
- Nécessité d'effectuer une phase de fusion des résultats locaux,
- La localité des traitements : un traitement est dit local si on peut prendre la décision sur l'état d'un pixel en se basant uniquement sur l'état de ses voisins. En effet, la LPE est généralement un concept qui n'est pas local. Cependant, certains travaux ont proposé des algorithmes basés sur des conditions locales pour la calculer et ce pour pouvoir la paralléliser.

	LPE par immersion		**LPE par ruissellement**			**LPE topologique**	**LPE d'arêtes**
	Meijster et al [56]	**Moga et al [52]**	**Zhou et al [107]**	**Meijster et al [57]**	**Bieniek et al [54]**	**Neerbos et al [98]**	**Mahmoudi et al [97]**
Décomposition du domaine ou fonctionnelle	Décomposition du domaine	Décomposition du domaine	Décomposition du domaine	Décomposition fonctionnelle	Décomposition du domaine	Décomposition du domaine	Décomposition du domaine
Synchronisation	Non	Oui	Non	Oui	Non	Oui	Oui
Communication	Oui	Oui	Oui	Non	Oui	Non	Oui
Type d'architecture d'implémentation	Multi-cœur à mémoire distribuée avec une partie de mémoire partagée	Multi-cœur à mémoire distribuée	Multi-cœur à mémoire partagée et Multi-cœur à mémoire distribuée	Multi-cœur à mémoire partagée	Architecture hybride	Architecture multi-cœur à mémoire partagée	Architecture multi-cœur à mémoire partagée
Etape de fusion	Non	Oui	Oui	Oui	Oui	Oui	Oui
Type du traitement	Local	Non local	Non local	Local	Non Local	---	Non local

Tableau 2. 1. Évaluation des algorithmes parallèles de calcul de la LPE

Globalement, les travaux sur la parallélisation de la LPE présentés dans ce chapitre procèdent selon deux stratégies : la parallélisation par décomposition du domaine et la parallélisation par décomposition fonctionnelle. Dans les approches de parallélisation par décomposition du domaine, l'image est décomposée spatialement d'une manière régulière sur l'ensemble des processeurs. Pour chaque processeur, l'algorithme est exécuté d'une manière séquentielle. Des points de synchronisation et de communication sont insérés là où le résultat dépend des sous domaines voisins. La solution finale est obtenue par fusion des résultats partiels.

L'approche de parallélisation par décomposition fonctionnelle est utilisée essentiellement dans les algorithmes qui procèdent par inondation à partir des minima régionaux. Ces minima sont répartis sur les différents processeurs. Dans ce cas, l'efficacité de la stratégie de parallélisation dépend crucialement du nombre des minima régionaux et de la taille de leurs bassins correspondants. Des problèmes d'équilibrage de charge se manifestent souvent lorsque les tailles de ces bassins diffèrent d'une manière significative [21,57].

Il est à noter que la majorité des algorithmes présentés utilise des files ordonnées. Les algorithmes de cette catégorie permettent d'avoir des exécutions rapides mais présentent l'inconvénient d'être difficilement scalables sur des architectures à mémoire distribuée puisque dans ce cas les files locales doivent être ordonnées d'une manière globale à travers une synchronisation appropriée.

L'algorithme de calcul de la LPE par inondation est difficile à paralléliser en raison de sa nature séquentielle. Une implémentation parallèle de cet algorithme peut être basée sur une transformation en un graphe de composantes. De son côté, la définition de la LPE basée sur la distance permet plusieurs implémentations parallèles. Les principales propositions de parallélisation de cette catégorie de LPE sont basées sur des files d'attente (ordonnées), un balayage par lignes (*raster scan*) répété ou une combinaison de ceux-ci.

En résumé, toutes les techniques de parallélisation présentées, ont recours à une étape de fusion qui est une opération globale qui peut pénaliser le taux de parallélisme.

9. Conclusion

Dans ce chapitre, nous avons commencé par présenter les concepts de base du calcul parallèle. Dans ce cadre nous avons abordé, entre autres, les principales architectures utilisées par ce calcul ainsi que les modèles de programmation qui en découlent. Nous nous sommes ensuite intéressés à l'application des concepts du parallélisme au calcul de la Ligne de Partage des Eaux, dans un objectif d'améliorer les performances de son implémentation, en termes de temps d'exécution. Nous avons présenté les principaux travaux effectués dans ce cadre notamment ceux qui ont visé les architectures multi-cœurs. A travers cet état de l'art, nous avons pu identifier les difficultés liées à cette problématique et dégager les stratégies qui ont été élaborées pour les résoudre. Dans ce contexte, nous soulignons que les travaux de parallélisation ont touché un grand nombre de variantes de la LPE. Toutefois, et à notre connaissance, l'algorithme de calcul de noyau par M-bord [30] n'a pas fait l'objet de travaux similaires. Dans le chapitre suivant, nous allons adresser la parallélisation de cet algorithme.

Chapitre 3 : Implémentation parallèle d'un algorithme de calcul de la LPE d'arêtes sur architecture multi-cœurs

1. Introduction

Le calcul de la LPE, à travers ses différents algorithmes, est un processus qui nécessite souvent plusieurs balayages de l'image ou des explorations en largeur et/ou profondeur de cette dernière. Il conduit à la création et au traitement de structures de données complexes telles que les graphes et les files de priorité. Ceci rend ces algorithmes relativement consommateurs en termes de temps et d'espace mémoire [35] surtout avec la croissance de la taille des images traitées. L'algorithme de calcul de noyau par M-bord [30] que nous avons présenté dans la section 7.2.1.2 du chapitre 1, n'échappe pas à cette règle. Pour optimiser le temps d'exécution, la parallélisation représente une piste intéressante à explorer même si cette dernière reste problématique vue la nature séquentielle inhérente aux traitements effectués dans le cadre du calcul de la LPE.

Dans ce chapitre, nous abordons cette problématique de parallélisation. Nous étudions particulièrement la parallélisation de l'algorithme de calcul de la LPE d'arêtes dans le cadre des Graphes à Arêtes Valuées (GAV) basé sur le calcul de noyau par M-bord [17, 30]. Nous rappelons dans ce contexte que l'intérêt que nous avons porté à cet algorithme est motivé principalement par la validation de l'intégration de sa version séquentielle dans une chaîne de segmentation d'images médicales du ventricule gauche du cœur. La parallélisation de cet algorithme constitue une contribution partielle à la parallélisation de cette chaîne qui malgré les bons résultats qu'elle produit souffre d'un temps d'exécution assez lourd entravant son exploitation sur le plan pratique. Outre cette motivation applicative, notre intérêt à cet algorithme trouve également son origine dans les propriétés théoriques de ce dernier. En effet, l'algorithme de calcul de noyau par M-bord, contrairement à la majorité de ses homologues, s'exécute en un temps linéaire [30]. Il est basé par ailleurs sur une propriété locale qui le prête bien à la parallélisation. Toutefois, nous montrons dans ce chapitre que malgré cette propriété de localité, un problème de dépendance de données est soulevé ce qui entrave la parallélisation. Par rapport à cela, nous proposons trois stratégies qui permettent de résoudre cette problématique de dépendance de données:

- La première stratégie de parallélisation consiste à traiter le graphe de départ sous forme de partitions. Cette stratégie basée sur le modèle de décomposition du domaine est une stratégie commune qui a été adoptée dans plusieurs travaux portant sur la parallélisation d'autres variantes de la LPE [51, 53]. La difficulté de cette stratégie réside dans la modélisation de la phase de fusion des résultats

partiels en vue de l'élaboration du résultat global final. Cette fusion reste spécifique à chaque variante considérée.

- La deuxième stratégie consiste à traiter les arêtes du graphe correspondant à l'image à segmenter d'une façon qui permet de résoudre la problématique de dépendance de données que soulève cet algorithme. Cette stratégie a permis un gain plus important que la première stratégie en termes d'accélération.
- La troisième stratégie proposée consiste à examiner les sommets du graphe au lieu des arêtes. Cette stratégie a donné le meilleur facteur d'accélération parmi les trois stratégies proposées.

Dans la deuxième partie de ce chapitre, nous présentons et discutons les résultats obtenus suite à l'implémentation des différents algorithmes parallèles proposés sur une architecture multi-cœurs à mémoire partagée.

2. Algorithme séquentiel de calcul du noyau par M-bord

L'algorithme séquentiel de calcul de noyau par M-bord (Algorithme 11) permet de calculer la LPE d'arêtes comme présenté dans la section 7.2.2 du chapitre 1. Cet algorithme prend comme entrée un GAV (*V, E, F*), la fonction de pondération des sommets et le sous-graphe des minima régionaux associé (l'ensemble de sommets et d'arêtes minima) et produit en sortie un noyau par M-bord de la fonction d'entrée. Il est basé sur une transformation d'amincissement qui consiste à abaisser, itérativement, l'ensemble des arêtes qui vérifient une propriété locale : les arêtes de bord adjacentes au sous-graphe des minima. Rappelons que *VF* désigne la fonction de pondération des sommets qui consiste à attribuer à un sommet $x \in V$ la valeur minimale des arêtes auxquelles il appartient.

Initialement, les arêtes sortantes du sous-graphe des minima sont stockées dans une liste *L* comme c'est indiqué dans la boucle itérative de la ligne 3 de l'algorithme 11. Ensuite, pour toute arête *u* de cette liste, si *u* est une arête de bord, alors elle est abaissée (lignes 10-11) et ajoutée à l'ensemble des arêtes minima (lignes 12-13). L'abaissement est appliqué aussi sur le sommet non minimum de l'arête de bord abaissée. Ensuite, une mise à jour de la liste *L* est effectuée en ajoutant les nouvelles arêtes sortantes du sous-graphe des minima (ligne 14-15). Ce processus est répété jusqu'à l'idempotence. Cette dernière est atteinte lorsqu'il n'y a plus d'arêtes à traiter dans la liste *L* (*cf*. boucle de la ligne 5).

Algorithme 11 : Noyau par M-bord [30]

Data : (V, E, F) un graphe à arête valuées;
Min=(V_{min}, E_{min}) et $VF(x)$ pour chaque $x \in V$.
Result : F: noyau M-bord de la fonction d'entrée F et M, ses minima.

```
1  // Initialisation
2  L := φ;
3  foreach (u ∈ E sortante de (V_min, E_min)) do
4      L := L ∪ {u};
5  while ( ∃ u ∈ L) do
6      L:=L \ {u};
7      if ( u est une arête de bord pour F ) then
8          x :=sommet de u tel que VF(x) < F(u);
9          y :=sommet de u tel que VF(y) = F(u);
10         F(u):= VF(x);
11         VF(y):= F(u);
12         V_min:= V_min ∪ {y};
13         E_min:=E_min ∪ {u};
14         foreach ( v = {y', y} ∈ E tel que y' ∉ V_min ) do
15             L := L ∪ {v};
```

La figure 3.1 illustre l'application de cet algorithme sur un GAV de taille 4×4. Le sous-graphe des minima *Min*=(V_{min}, E_{min}) est représenté en gras (figure 3.1 (a)). Cet exemple montre 8 arêtes sortantes de *Min* qui sont: {*bc*}, {*dc*}, {*ae*}, {*bf*}, {*gc*}, {*gf*}, {*kj*} et {*on*}. Parmi ces arêtes 4 sont des arêtes de bord (représentées en rouge). Les figures 3.1 (b-h) représentent les différentes itérations de l'application de l'Algorithme 11 sur le graphe de la figure 3.1 (a). A l'itération 1 (figure 3.1(b)), l'arête {*bc*} qui est une arête M-bord est abaissée (ainsi que le sommet *c*). Donc, l'arête {*bc*} ainsi que le sommet *c* sont ajoutés au sous-graphe des minima. A l'itération 2, nous remarquons que l'arête {*dc*} n'est plus une arête de bord contrairement à l'arête {*ae*} qui est une arête M-bord, elle est donc, ainsi que le sommet *e*, abaissée et ajoutée à l'ensemble des arêtes minima. A l'itération 3 (respectivement 4, 5, 6 et 7), l'arête M-bord {*fg*} (respectivement {*fj*}, {*ei*}, {*im*} et {*mn*}) sont traitées de la même manière. Le noyau par M-bord présenté en gras dans la figure 3.2 (b) est ainsi obtenu.

L'ensemble des arêtes non minima composées de deux sommets appartenant à des minima différents sont des arêtes séparantes et composent la LPE d'arêtes.

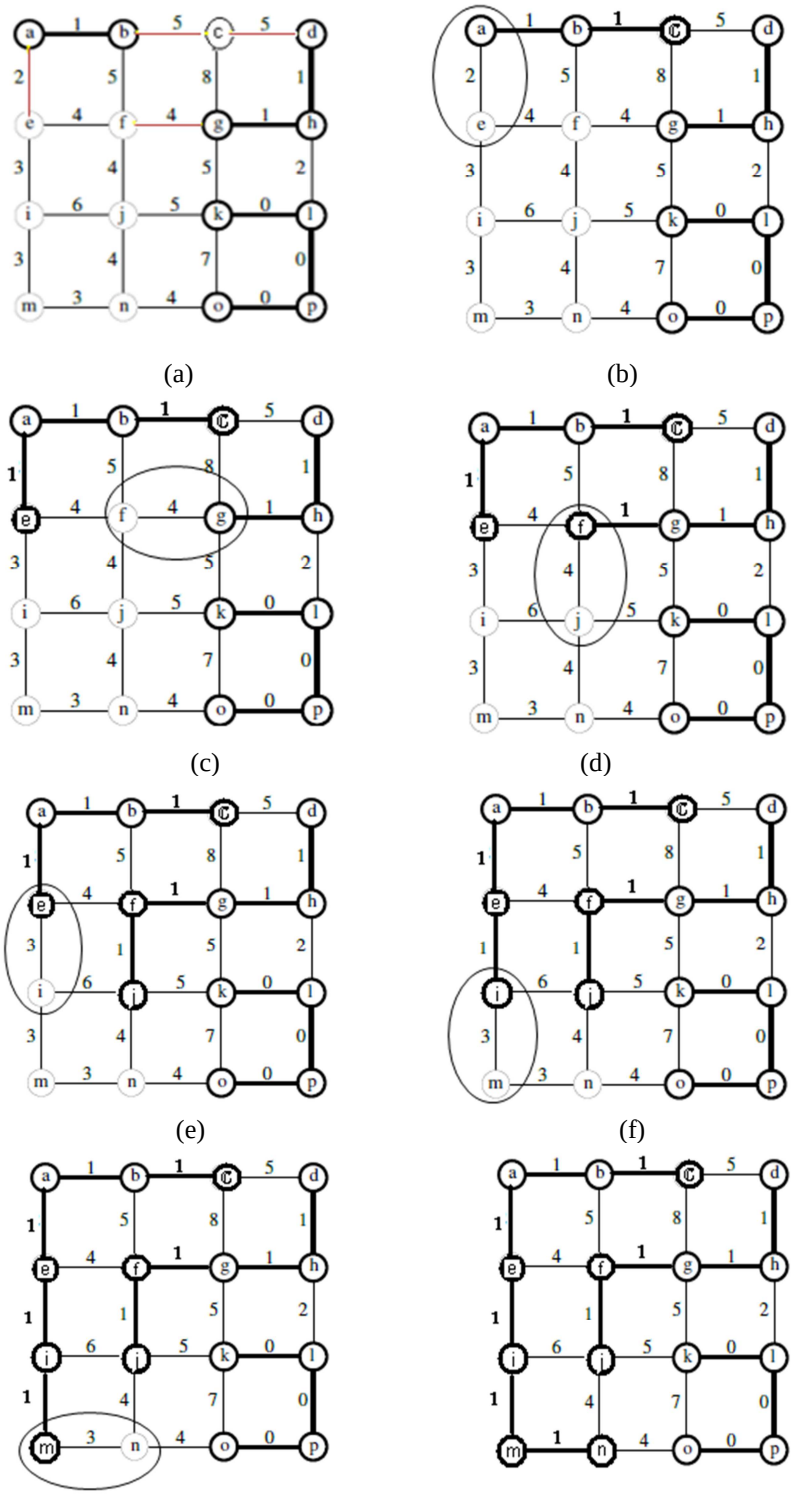
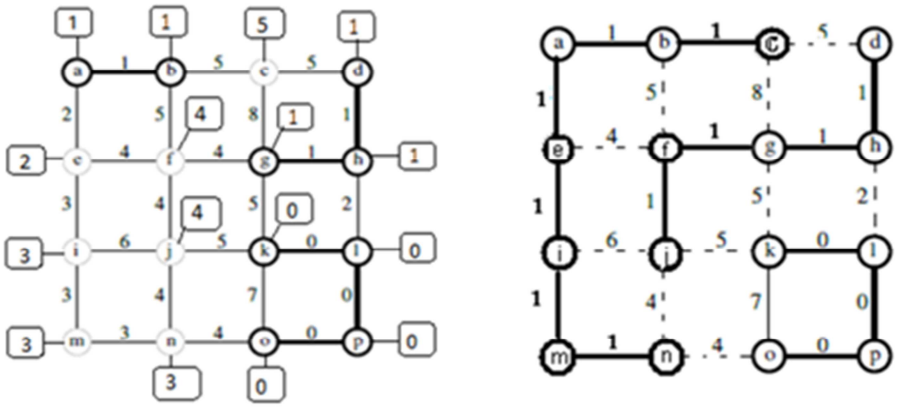


Figure 3. 1 Traitement séquentiel de l'algorithme de calcul de noyau par M-bord (a) un graphe à arêtes valuées (V, E, F); (b-g) les étapes de calcul de noyau par M-bord en appliquant l'Algorithme 11 sur le graphe de (a); (h) noyau par M-bord de la fonction d'entrée F



(a) (b)

Figure 3. 2 (a) Un graphe à arêtes valuées (b) Résultat de l'application de l'Algorithme 11 sur le graphe de (a) un noyau par M-bord de la fonction d'entrée (en gras) et la coupure induite par ce noyau (en pointillé)

3. Analyse de dépendance des données

La connaissance des dépendances entre les données est fondamentale dans l'implémentation d'algorithmes parallèles. Elle peut être classifiée selon l'ordre de lecture et d'écriture des variables utilisées. Dans ce contexte, trois types de dépendances peuvent être distingués:

- Dépendance de flot : une instruction écrit dans une variable, suivie d'une autre instruction qui la lit.
- Anti-dépendance : une instruction lit la variable avant qu'une instruction l'écrive.
- Dépendance de sortie : deux instructions écrivent dans la même variable.

Pour élaborer des stratégies adéquates de parallélisation de l'algorithme de calcul du noyau par M-bord, nous commençons d'abord par une analyse de la dépendance des données qui peut entraver cette tâche.

Pour ce faire, nous rappelons que l'algorithme de calcul du noyau par M-bord est basé principalement sur l'opération d'abaissement des arêtes M-bord comme cela a été présenté dans le chapitre 1. L'abaissement d'une arête M-bord *u*, consiste à attribuer la valeur du sommet minimum à l'arête *u* et au sommet non-minimum de *u*. En se basant sur cette définition, le processus d'abaissement s'avère un processus local puisqu'il ne requiert pour sa mise en œuvre aucune autre information supplémentaire concernant les autres arêtes du graphe. Par conséquent, un ensemble d'arêtes M-bord *indépendantes* (*mutuellement disjointes*) peut être abaissé simultanément. Cette propriété de localité rend cet algorithme adapté à une implémentation parallèle. Toutefois, nous montrons que certaines configurations

peuvent être problématiques par rapport à cette parallélisation. En effet, si nous considérons le cas de deux ou plusieurs arêtes M-bord adjacentes qui ont un sommet non-minimum commun. Dans cette configuration, une seule arête M-bord parmi les arêtes adjacentes doit être abaissée. L'abaissement de cette arête va entraîner le changement du type des autres arêtes qui ne seront plus des arêtes M-bord dans ce cas. Dans une exécution séquentielle, cette situation ne pose aucun problème. Cependant, dans une exécution parallèle, un problème de dépendance se manifeste. En effet, si chacune de ces arêtes est prise en charge par un processeur alors cette configuration va engendrer l'abaissement d'arêtes qui ne sont pas des arêtes M-bord. De ce fait, les arêtes qui peuvent être abaissées en parallèle doivent remplir une certaine condition. Nous appelons ces arêtes des arêtes M-disjointes. Elles sont définies comme suit:

Notion d'arêtes M-disjointes:

Soient G (V, E, F) un Graphe à Arêtes Valuées, Min=(E_{min}, V_{min}) le sous-graphe des minima associé, u et v deux arêtes sortantes de Min et x et y respectivement les sommets non minima de u et v.

u et v sont dites des *arêtes disjointes* si $x \neq y$. De plus, si ces arêtes sont des *arêtes de bord*, alors nous appelons ces arêtes des arêtes *M-disjointes.*

En conclusion, nous appelons des arêtes *M-disjointes* les arêtes M-bord qui n'ont aucun sommet non minimum en commun.

La figure 3.3 illustre le problème de dépendance de données (problème d'abaissement des arêtes qui ne sont pas *M-disjointes*). Dans le graphe à arêtes valuées G présenté dans la figure 3.3 (a), les arêtes {bc} et {cd} sont deux arêtes M-bord qui ont un sommet non minimum commun. Donc, ces deux arêtes ne sont pas M-disjointes et par la suite elles ne peuvent pas être traitées en parallèle car si on abaisse l'une d'entre elles, l'autre ne serait plus une arête M-bord (figure 3.3 (b)). Cependant, les arêtes {bc} et {gf} sont deux arêtes M-bord qui n'ont pas de sommet non-minimum commun. Donc, ces deux arêtes sont *M-disjointes* et par la suite elles peuvent être abaissées en parallèle.

En conclusion, seules les arêtes *M-disjointes* peuvent être abaissées en parallèle.

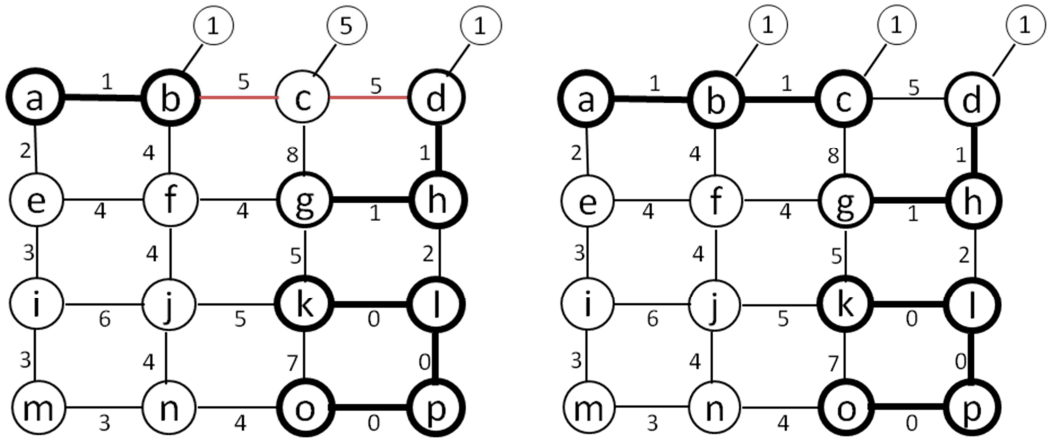


Figure 3. 3 Un graphe à arêtes valuées (a) les minima sont représentés en gras (b) l'abaissement de l'arête {bc} a entraîné que l'arête {cd} n'est plus une arête M-bord

En tenant compte de cette dépendance de données, nous proposons dans la suite, trois stratégies de parallélisation de l'algorithme 11, permettant de surmonter cette problématique. Ces stratégies sont : la stratégie de parallélisation par décomposition du domaine, la stratégie de parallélisation par traitement alterné des arêtes et enfin la stratégie de parallélisation par exploration des sommets du graphe.

4. Parallélisation par décomposition du domaine

La décomposition du domaine est une stratégie communément utilisée dans la parallélisation des algorithmes qui manipulent des données spatialement réparties. Ceci est le cas des algorithmes de traitement d'image où le domaine considéré correspond à la grille représentant l'image. Cette stratégie consiste alors à décomposer cette grille en un ensemble de partitions pouvant se chevaucher ou non et à appliquer la version séquentielle de l'algorithme parallèlement sur chacune de ces partitions. Des étapes de synchronisation et de fusion des résultats partiels obtenus dans chaque partition s'imposent souvent pour aboutir au résultat global.

Plusieurs travaux de parallélisation des algorithmes de calcul de la LPE ont adopté cette stratégie de décomposition du domaine [51, 53, 98]. Ces travaux même s'ils se ressemblent du côté des étapes de partitionnement et de l'application séquentielle des algorithmes étudiés, se distinguent par contre au niveau des étapes de synchronisation et de fusion. La formulation de ces dernières étapes étant étroitement liée aux spécificités de l'algorithme considéré et à ses besoins particuliers en données à échanger entre les processeurs.

Notre première contribution dans cette thèse consiste dans l'exploration des performances de cette stratégie dans la parallélisation du calcul de la LPE en utilisant l'algorithme de calcul du noyau par M-bord. Dans ce cadre, notre proposition peut être résumée par les étapes suivantes:

(i) Partitionnement du graphe représentant l'image en plusieurs partitions et leur attribution aux différents processeurs.
(ii) Application par chaque processeur de l'algorithme 11 à la partition qui lui est associée,
(iii) Répétition de l'étape (ii) jusqu'à la stabilité totale.

Nous détaillons dans ce qui suit chacune de ces étapes.

4.1 Approche de partitionnement utilisée

Le partitionnement du domaine d'une image peut se faire selon une seule dimension (les lignes ou les colonnes) ou selon deux dimensions (les lignes et les colonnes). Dans le premier cas, le partitionnement produit des partitions qui se présentent sous forme de bandes horizontales ou verticales ayant chacune une ou au plus deux bandes adjacentes. Dans le deuxième cas, les partitions sont plutôt de forme rectangulaire (appelées blocs) pouvant avoir deux, trois ou quatre partitions adjacents. Lorsque la nature de l'algorithme à paralléliser nécessite une communication et un échange de données entre les partitions, ce qui est le cas de l'algorithme du calcul du noyau par M-bord, il est préférable de limiter le nombre des zones frontalières. C'est pourquoi nous avons opté dans notre travail à un partitionnement en bandes. Ce partitionnement est réalisé par l'algorithme 13. Cet algorithme produit des bandes (ou partitions) horizontales non chevauchées (*cf.* figure 3.5). Chaque bande représente un sous-graphe de G et est traitée par un processeur p avec $p=\{1,\ldots,P\}$. Chaque processeur calcule les indices des frontières de sa partition correspondante. Ces frontières étant définies par les indices de la première et la dernière arête horizontale (respectivement verticale) comme c'est indiqué dans les lignes de 4 à 10 de l'algorithme 13.

Etant donné une grille de taille $rs \times cs$ où cs désigne la taille d'une colonne (le nombre de lignes) et rs désigne la taille d'une ligne (le nombre de colonnes). Soit P le nombre de processeurs, Q le quotient de cs par P et R le reste de cette division. Le partitionnement selon l'algorithme 13 produit :

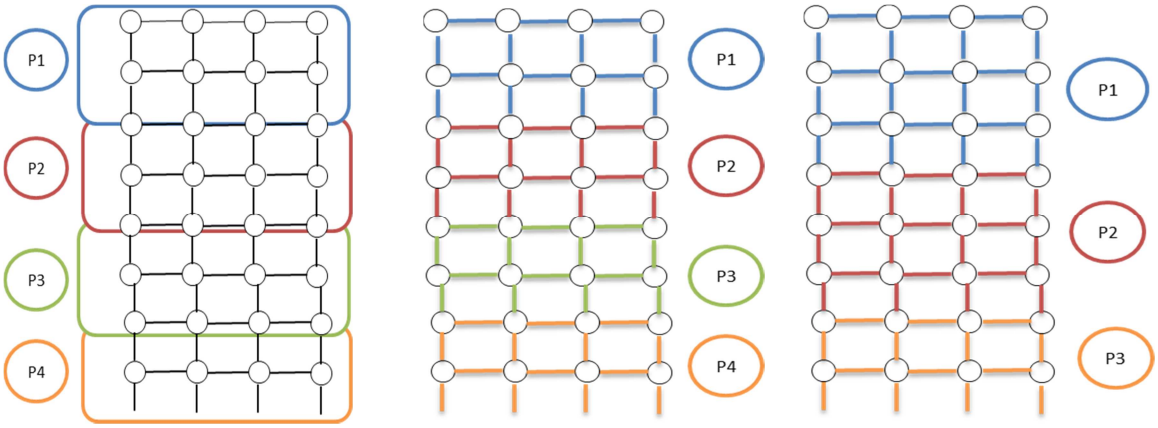
- *R* bandes de taille (2×*rs*× (*Q*+1) – 3) et
- *P*-*R* bandes de taille (2×*rs*× *Q* – 3)

Algorithme 13 : Partitionnement_statique [72]

Data : un graphe à arêtes valuées (V, E, F) de taille $rs*cs$ et P le nombre de processeurs;
Result : $(partition_1, \ldots, partition_P)$: P partitions à traiter en parallèle par P processeurs.

```
1  //initialisation
2  Q= cs/P;
3  R= cs%P;
4  foreach (Processeur p ∈ {1, ...,P}) do in parallel
5      if (p ≤ R) then
6          cs_start[p]:=(p-1)*(Q+1);
7          cs_end[p]:=cs_start[p] + Q;
8      else
9          cs_start[p]:=(p-1)*Q+R;
10         cs_end[p]:=cs_start [p]+ Q - 1;
11     //indice de début des arêtes horizontales
12     HE_StartIndex[p] := cs_start[p] * rs;
13     //indice de fin des arêtes horizontales
14     HE_EndIndex[p] := (cs_end[p] + 1) * rs − 2;
15     //indice de début des arêtes verticales
16     VE_StartIndex[p] := N + cs_start[p] * rs;
17     //indice de fin des arêtes verticales
18     VE_EndIndex[p] := N + (cs_end[p] + 1) * rs − 1;
19     partition_p:=E[[HE_StartIndex[p], ..., HE_EndIndex[p]] +
       [VE_StartIndex[p], ..., VE_EndIndex[p]] ];
```

La figure 3.4 illustre le partitionnement d'un graphe *G* de taille 4×8 (représenté dans la figure 3.4 (a)) en quatre partitions dans la figure 3.4 (b) et en trois partitions dans la figure 3.4 (c) selon l'algorithme 13. Le graphe initial est composé de 24 arêtes horizontales et 32 arêtes verticales. Dans le premier cas présenté dans la figure 3.4 (b) (cas où *P*=4), les partitions sont équilibrées et composées de 6 arêtes horizontales et 8 arêtes verticales. D'autre part, dans le cas présenté dans la figure 3.4 (c) (cas où *P*=3 donc *R*=2), les deux premières partitions contiennent 9 arêtes horizontales et 12 arêtes verticales soit 21 arêtes en total alors que la troisième partition contient 6 arêtes horizontales et 8 arêtes verticales soit 14 arêtes en total.



(a) (b) (c)

Figure 3. 4 Partitionnement du graphe de départ (a) en (b) P=4 et (c) P=3 partitions non chevauchées et équilibrées selon l'algorithme 13

4.2 Application parallèle de l'algorithme séquentiel sur les bandes

A la sortie de l'étape de partitionnement, chaque processeur dispose d'une information lui indiquant la zone de l'image sur laquelle il doit agir. Les *P* processeurs vont exécuter alors en parallèle la version séquentielle de l'algorithme de calcul du noyau par M-bord. Il à noter dans ce cadre que tous les processeurs travaillent sur la même image et qu'aucune copie de données n'est nécessaire du fait de l'absence de chevauchement entre les partitions considérées. Ceci est implémenté à l'aide de pointeurs locaux à chaque processeur qui permettent d'adresser une zone mémoire partagée par tous les processeurs et correspondant à l'image.

4.3 Stabilisation de l'exécution parallèle

Dans le cadre de l'exécution de l'algorithme de calcul du noyau par M-bord sur une bande associée à un processeur, le processus d'abaissement des arêtes sortantes peut s'arrêter soit pour des raisons topographiques (lorsqu'on atteint la limite du bassin versant associé à un minimum régional) soit lorsqu'on atteint la limite de la bande en cours de traitement. Dans ce dernier cas de figure, une situation particulière peut se produire: Soient P_i et P_{i+1} deux processeurs travaillant sur deux bandes horizontales adjacentes notées respectivement $bande_{Pi}$ et $bande_{Pi+1}$, *B* l'ensemble des arêtes qui composent la frontière entre ces deux bandes et *u* une arête de *B* telle que $u=\{x,y\}$ avec $x \in bande_{Pi}$ et $y \in bande_{Pi+1}$. Dans le cas d'un bassin versant ayant son origine dans un minimum régional situé dans $bande_{Pi}$ et s'étalant vers la bande $bande_{Pi+1}$, une arête *u* située au niveau de la frontière de la $bande_{Pi+1}$ qui n'est pas sortante à une itération donnée peut le devenir lorsque le processeur P_i termine son exécution et abaisse le sommet *x* de *u* qui lui appartient. Dans un tel cas, le processeur P_{i+1} doit être

réactivé. Ce processus est engendré par cette (ces) nouvelle(s) arête(s) sortante(s). De ce fait, il est possible de dire que la stabilité de l'exécution parallèle des *P* processeurs est atteinte lorsqu'aucune arête sortante ne se manifeste au niveau des *P*-1 frontières entre les *P* bandes considérées.

En tenant compte de ces étapes, l'algorithme parallèle de calcul de noyau par M-bord proposé se présente comme suit:

Algorithme 12 : Parallélisation par décomposition du domaine

Data : graphe à arêtes valuées (V, E, F), $Min = (V_{min}, E_{min})$le sous-graphe des minima régionaux associé et $VF(x)$ pour tout $x \in V$;
et P le nombre de processeurs
Result : F Noyau par M-bord de la fonction d'entrée et M ses minima

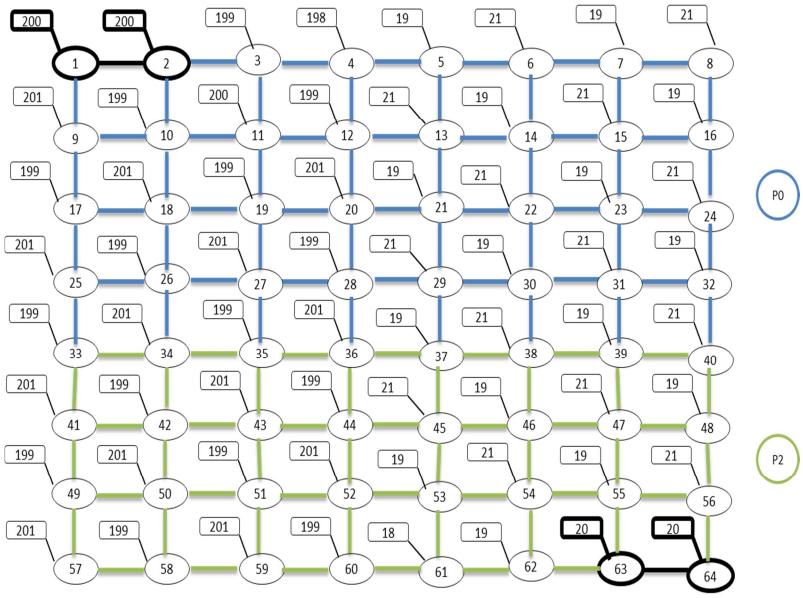
```
1  // Initialisation
2  (bande_1 , ... , bande_P):=Partitionnement_Statique(V,E,P);
3  while (il existe un processeur actif) do
4      foreach ( processeur p ∈ (1, ... , P) ) do in parallel
5          L_p := φ;
6          foreach (u ∈ bande_p sortante de Min ) do
7              L_p:=L_p ∪ {u};
8          while (il existe u ∈ L_p) do
9              L := L \ {u};
10             if (u est une arête de bord pour F) then
11                 x :=sommet de u tel que F(x) < F(u);
12                 y :=sommet de u tel que F(y) = F(u);
13                 F(u) := VF(x); VF(y) := F(u);
14                 V_min := V_min ∪ {y}; E_min := E_min ∪ {u};
15                 foreach (v={y',y} ∈ E telle que y' ∉ V_min &
                       v ∈ bande_p) do
16                     L_p := L_p ∪ {v};
```

Cet algorithme utilise un partitionnement statique pour attribuer à chaque processeur sa bande associée (ligne 2 de l'algorithme 12). Chaque bande est traitée par un processeur *p* avec *p* = {1,…, *P*} (ligne 4 de l'algorithme 12). Chaque processeur applique l'algorithme 11 sur la bande associée (lignes 4-16). Notons que la mise à jour des arêtes sortantes du sous-graphe des minima tient compte des frontières de la bande traitée par chaque processeur comme il est noté dans la boucle à la ligne 15 de l'algorithme 12. En effet, seules les arêtes appartenant à la bande en cours de traitement sont ajoutées à la liste locale L_p.

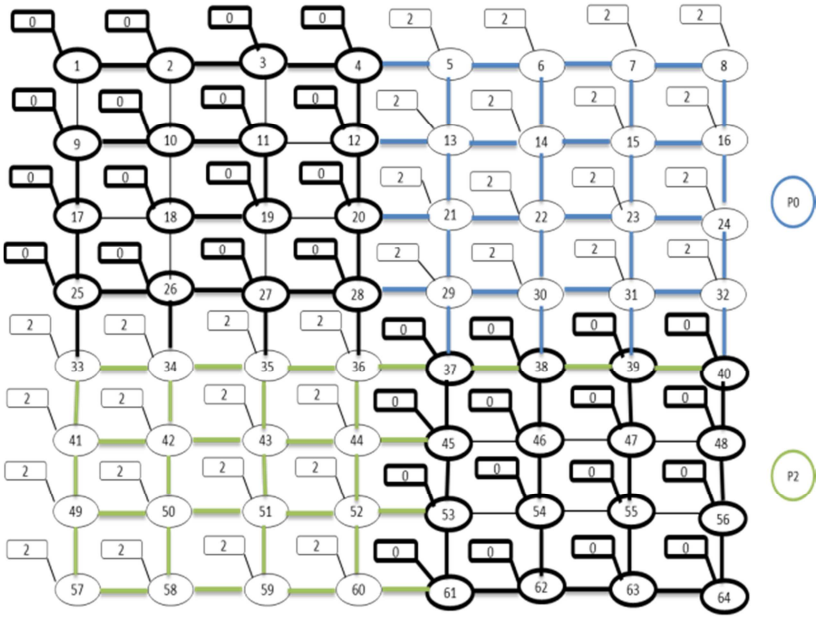
A la fin d'une itération *k*, la liste locale des arêtes sortantes du sous-graphe des minima associée à un processeur P_i est vide. A l'itération *k+1*, cette liste sera réinitialisée en lui ajoutant les nouvelles arêtes sortantes du sous-graphe des minima localisées dans la bande en cours de traitement ($bande_p$). Notons que ces arêtes ne se produisent qu'au niveau des frontières entre les processeurs. En effet, l'abaissement d'une arête sur la frontière de deux bandes associées à deux processeurs P_i et P_{i+1} par le processeur P_i peut entraîner la réactivation du processeur P_{i+1}. Enfin, ceci est répété jusqu'à ce qu'aucun processeur ne détecte de nouvelles arêtes sortantes dans la bande qu'il traite, comme il est noté dans la boucle principale à la ligne 4 de l'algorithme 12.

Ce cas est illustré dans la figure 3.5. En effet, la figure 3.5 (a) représente un graphe à arêtes valuées et en gras le sous-graphe des minima associé. Les sommets sont numérotés de 1 à 64. Les valeurs des sommets sont présentées dans les rectangles au-dessus de chaque sommet.

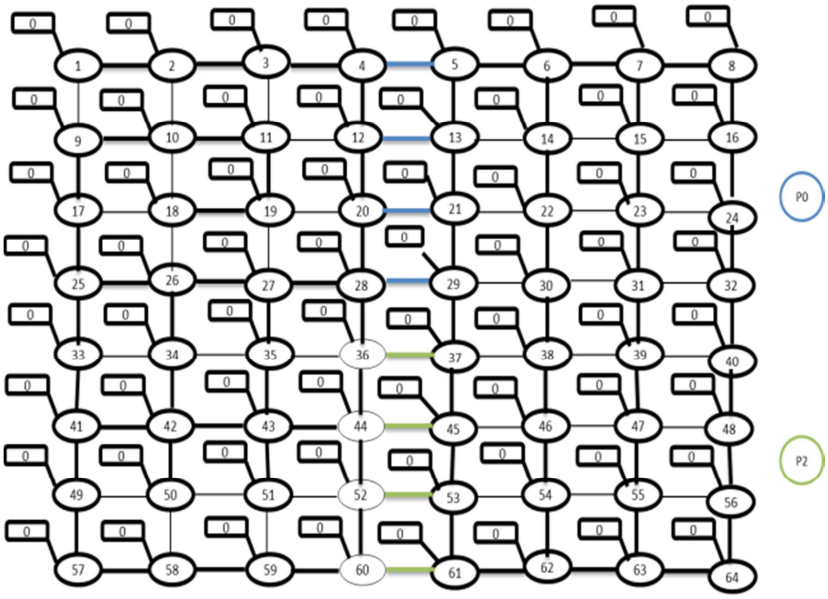
Nous considérons deux processeurs (P=2). Le processeur P_0 (respectivement P_1) traite les arêtes en bleu (respectivement en vert). A l'itération 0 (illustré dans la figure 3.5 (b)), les deux processeurs abaissent les arêtes M-bord dans leurs partitions associées. Le processeur P_1 termine son traitement mais il n'a pas abaissé toutes les arêtes de sa partition associée. L'arête liant les sommets numéro 25 et 33 est abaissée par le processeur P_0. Ce processus a entraîné le changement du type de l'arête liant les sommets 33 et 41 qui est devenue une arête M-bord. Donc, le processeur P_1 est réactivé (figure 3.5(c)).



(a)



(b)



(c)

Figure 3. 5 Réactivation d'un processeur suite au changement d'état d'une arête située au niveau de la frontière entre deux processeurs

En conclusion, la stratégie proposée dans cette section pour paralléliser l'algorithme de calcul de noyau par M-bord est basée sur l'approche de décomposition du domaine. Cette stratégie consiste en un partitionnement statique de la charge sur les différents processeurs ce qui peut entraîner un déséquilibre de charge et nécessiter un taux de communication et de synchronisation important affectant négativement la performance du parallélisme.

Dans la section suivante une deuxième stratégie de parallélisation de cet algorithme basée sur un algorithme de partitionnement dynamique est proposée.

5. Parallélisation basée sur le traitement des arêtes du graphe en alternance

L'exploitation de la parallélisation de l'opération d'abaissement des arêtes M-bord semble être une stratégie plus efficace comparée à la parallélisation basée sur l'approche de décomposition du domaine, vue la localité de cette opération comme nous l'avons mentionné auparavant. Cependant une telle approche fait face au problème de dépendance de données soulevé par les arêtes M-bord qui ne sont pas M-disjointes. Pour tirer profit de la localité tout en évitant ce problème de dépendance de données, nous proposons dans le cadre de ces travaux une deuxième stratégie qui consiste à répartir les arêtes sur des ensembles ne pouvant contenir chacun que des arêtes M-disjointes. Les arêtes de chaque ensemble peuvent alors être traitées en parallèle. Toutefois, vu la nature progressive et spatiale du processus d'abaissement des arêtes d'un ensemble E_1 qui n'étaient pas des arêtes M-bord, peuvent le devenir suite à l'abaissement d'autres arêtes d'un ensemble E_2. Par conséquent l'idempotence ne peut pas être atteinte en une itération mais elle en nécessite plusieurs. Cette stratégie peut alors être résumée par les étapes suivantes :

(i) Distribuer les arêtes du graphe de départ en 4 sous-ensembles d'arêtes indépendantes (mutuellement disjointes),

(ii) Traiter les arêtes de chaque sous-ensemble. Pour chaque sous-ensemble, répartir dynamiquement (à chaque itération) ses arêtes sur P processeurs afin de les traiter en parallèle. Le traitement consiste à détecter si l'arête examinée est une arête M-bord et à l'abaisser si tel est le cas.

(iii) Répéter l'étape (ii) jusqu'à l'idempotence.

5.1 Distribution des arêtes sur des sous-ensembles d'arêtes mutuellement disjointes

Afin d'obtenir des ensembles d'arêtes mutuellement disjointes nous proposons de distribuer ces dernières en quatre sous-ensembles : deux sous-ensembles d'arêtes horizontales obtenus en alternant les colonnes des arêtes horizontales du graphe et deux sous-ensembles d'arêtes verticales obtenus en alternant les lignes d'arêtes verticales du graphe. Notons que cette répartition est valable dans le cas de la 4-adjacence que nous considérons dans ce travail et qu'une autre adjacence nécessitera une autre répartition qui lui est adaptée. La figure 3.6 donne une illustration de la répartition que nous venons de décrire. Dans cette figure :

- TH1 désigne le premier sous-ensemble d'arêtes horizontales appartenant aux colonnes à indices paires (en rouge);
- TH2 désigne le deuxième sous-ensemble d'arêtes horizontales appartenant aux colonnes d'indices impaires (en bleu) ;
- TV1 désigne le premier sous-ensemble d'arêtes verticales liant un sommet d'une ligne d'indice *i* et un sommet de la ligne d'indice *i*+1 tel que *i* est paire (en vert) ;
- TV2 désigne le deuxième sous-ensemble d'arêtes verticales liant un sommet d'une ligne d'indice *i* et un sommet de la ligne d'indice *i*+1 tel que *i* est impaire (en oranger).

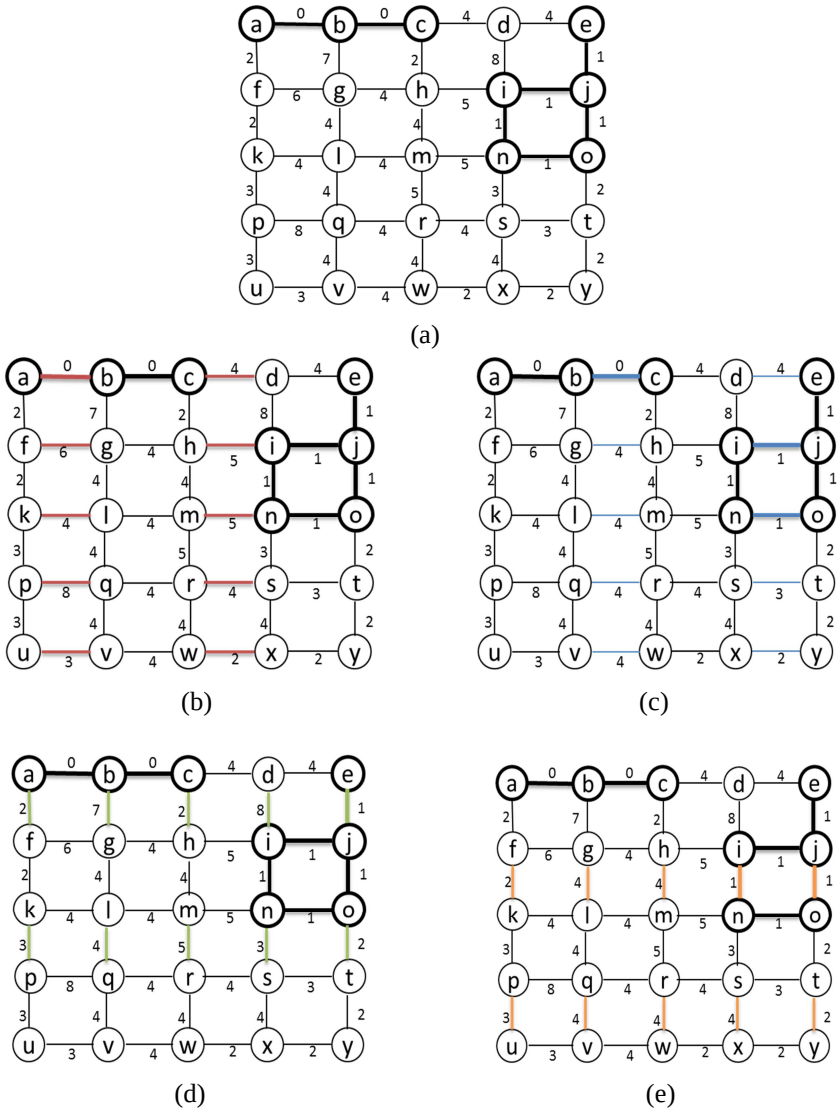


Figure 3. 6 Illustration de l'algorithme de distribution des arêtes d'un graphe en k=4 sous-ensembles indépendants (a) graphe de départ (b et c) partitionnement des arêtes horizontales en alternance (d et e) partitionnement des arêtes verticales en alternance

TH1	**{ab}**	{fg}	{kl}	{pq}	{uv}	{cd}	{hi}	{mn}	{rs}	{wx}
TH2	**{bc}**	{gh}	{lm}	{qr}	{vw}	{de}	**{ij}**	**{no}**	{st}	{xy}
TV1	{af}	{bg}	{ch}	{di}	**{ej}**	{kp}	{lq}	{mr}	{ns}	{ot}
TV2	{fk}	{gl}	{hm}	**{in}**	**{jo}**	{pu}	{qv}	{rw}	{sx}	{ty}

Tableau 3. 1 Partitionnement des arêtes du graphe en k=4 sous-ensembles (TH1, TH2, TV1 et TV2)

Afin d'obtenir la distribution d'arêtes en des sous-ensembles d'arêtes mutuellement disjointes nous proposons l'algorithme 14.

Algorithme 14 : Distribution_Arêtes

Data : un graphe de taille rs*cs avec rs désigne le nombre de colonnes et cs désigne le nombre de lignes.

Result : Sous-ensembles d'arêtes $(L_1, \ldots, L_4)$ telle que les arêtes appartenant à un sous-ensemble sont indépendantes et peuvent être traitées en parallèle.

```
1  // Initialisation
2  L1 := φ; L2 := φ; L3 := φ; L4 := φ;
3  // Traitement des arêtes horizontales:
4  for (j=0, j < cs, j++) do
5      for (i=0, i < rs-1, i+=2) do
6          L1:= L1 ∪ {u[j,i]};
7  for (j=0, j < cs, j++) do
8      for (i=1, i < rs-1, i+=2) do
9          L2:= L2 ∪ {u[j,i]};
10 // Traitement des arêtes verticales:
11 for (j=0,j < cs-1,j+=2) do
12     for (i=0,i < rs,i++) do
13         L3:= L3 ∪ {u[j,i]};
14 for (j=1,j < cs-1,j+=2) do
15     for (i=0,i < rs,i++) do
16         L4:= L4 ∪ {u[j,i]};
```

Tenant compte de cette distribution des arêtes, les arêtes de chaque sous-ensemble parmi TH1, TH2, TV1 et TV2 sont réparties dynamiquement sur *P* processeurs et traitées en parallèle. A la fin de l'exécution, la liste des arêtes M-bord de chaque sous-ensemble est mise à jour et l'abaissement parallèle est répété jusqu'à l'idempotence. L'algorithme 15 détaille le déroulement de toutes ces étapes.

Algorithme 15 : Parallélisation basée sur le traitement des arêtes du graphe de départ en alternance [71]

```
Data : (V, E, F) : graphe à arêtes valuées, (V_min,E_min) et VF(x) pour
       chaque x ∈ V;
Data : P nombre de processeurs.
Result : F un noyau par M-bord de la fonction d'entrée et ses minima
1  // Initialisation
2  flag:= 1;
3  // I désigne la relation d'adjacence utilisée
4  foreach (k = {1..I}) do
5      L_k := φ;
6  (L_1, ... , L_k):=Distribution_Arêtes(E);
7  while (flag = 1) do
8      foreach ( L_i avec i ∈ {1, ... ,k} ) do
9          (E_1, ... ,E_P):= Partitionnement_Dynamique(L_i,P);
10         flag:=0;
11         foreach (Processeur p in {1, ... ,P} ) do in parallel
12             foreach (arête u ∈ E_p) do in parallel
13                 if (u est une arête M-bord) then
14                     flag:=1;
15                     E_p:= E_p \ {u} ;
16                     F(u) := VF(x);
17                     VF(y) := F(u);
18                     V_min := V_min ∪ {y};
19                     E_min := E_min ∪ {u};
```

Dans l'algorithme proposé (Algorithme 15), les données manipulées concernent les sommets, les arêtes, les sommets minima et les arêtes minima. Ces données sont stockées dans des structures de données de type tableau dynamique. Ces structures sont partagées entre les *P* processeurs. L'accès à ces structures se fait par des pointeurs locaux à chaque processeur. Il est à noter que vue la distribution dynamique de la charge que nous avons adoptée (Algorithme 17), deux processeurs différents ne peuvent pas travailler sur la même zone mémoire de ces structures et par conséquent le problème de synchronisation ne se pose pas. Par ailleurs, toujours grâce à l'algorithme de partitionnement adopté (Algorithme 17) et vue la nature locale des traitements effectuées sur les arêtes, aucune communication n'est nécessaire entre les *P* processeurs.

En résumé, la particularité de la stratégie de parallélisation proposée dans cette section réside dans le fait que les données à traiter sont distribuées sur les différents processeurs de manière à éviter la problématique de dépendance de données décrite dans la section 3.1 de ce chapitre. Cette stratégie permet d'avoir un parallélisme ayant une granularité plus importante

que celle de la première stratégie sans pour autant augmenter le taux de synchronisation. En effet, pour la première stratégie proposée dans la section 4 si le nombre de processeurs augmente alors le taux de synchronisation entre ces processeurs augmente. En revanche, pour la deuxième stratégie proposée dans cette section, la synchronisation ne dépend pas du nombre de processeurs mais du nombre des sous-ensembles d'arêtes. Ainsi, une augmentation de nombre de processeurs n'affecte pas le taux de synchronisation. En outre, cette stratégie est basée sur un partitionnement dynamique qui attribue aux différents processeurs, à chaque itération, la même charge de données (arêtes) à traiter. Par ailleurs, cette stratégie permet d'éviter le parcours en largeur d'abord requis dans la première stratégie proposée.

En revanche, dans l'algorithme parallèle proposé dans cette section les arêtes appartenant à chaque sous-ensemble sont traitées en parallèle mais les quatre sous-ensembles sont traités séquentiellement ce qui entraîne une certaine sérialisation de l'algorithme.

6. Parallélisation basée sur l'examen des sommets du graphe

Par rapport à la première stratégie que nous avons proposée, la deuxième stratégie permet d'avoir une granularité plus importante même si les ensembles d'arêtes sont traités séquentiellement. Mais cet aspect séquentiel reste minime devant le gain que permet la parallélisation du traitement des arêtes de chaque sous-ensemble. Dans un souci d'améliorer davantage la granularité sans introduire pour autant des contraintes de synchronisation, nous avons essayé de proposer une autre stratégie qui permet d'explorer en parallèle toutes les arêtes traitables tout en évitant le problème de dépendance des données. Pour ce faire nous proposons une modification au niveau de l'algorithme de calcul du noyau par M-bord. Cette modification qui concerne la manière avec laquelle se font le parcours et l'examen des arêtes à traiter permet de préserver les propriétés fondamentales de l'algorithme initial. Nous montrons que cette stratégie garantit un taux de parallélisme plus important.

L'idée principale de l'algorithme parallèle que nous proposons consiste à traiter en parallèle les sommets non-minima adjacents aux sommets minima tout en préservant le paradigme d'amincissement sur lequel est basé l'algorithme séquentiel de calcul de noyau par M-bord. Rappelons que ce paradigme consiste à abaisser, itérativement, l'ensemble des arêtes qui vérifient une propriété locale : les arêtes de bord adjacentes au sous-graphe des minima.

La stratégie de parallélisation de calcul de noyau par M-bord basée sur l'examen des sommets trouve sa justification dans la propriété suivante:

Soit $G(V, E, F)$ un graphe valué se présentant sous forme d'une grille carrée (cas de la 4-adjacence). Dans un tel graphe chaque sommet fait partie de quatre arêtes (deux horizontales et deux verticales). Soit *Min* (V_{min}, E_{min}) le sous-graphe des minima associé. Etant donné deux sommets $x \in V \backslash V_{min}$ et $y \in V \backslash V_{min}$ alors les arêtes auxquelles appartiennent x et y ne peuvent être que mutuellement disjointes au sens de la définition donnée dans la section 3 de ce chapitre.

Cette propriété garantit que lors d'un traitement parallèle des arêtes appartenant à deux sommets non-minima, la problématique de dépendance des données (détaillée dans la section 3 de ce chapitre) ne se produit pas. L'algorithme 16 détaille la stratégie de parallélisation que nous avons proposé dans ce cadre.

Algorithme 16 : Parallélisation basée sur l'examen des sommets au lieu des arêtes [72]

Data : (V, E, F) un graphe à arête valuées; (V_{min}, E_{min}) le sous-graphe des minima; $VF(x)$ pour chaque $x \in V$ et P nombre de processeurs.
Result : F noyau par M-bord de la fonction d'entrée et M ses minima.

```
1  // Initialisation
2  V_Nmin = V \ {V_min} ; // l'ensemble V_Nmin contient les sommets non minima
3  while ( V_Nmin ≠ φ ) do
4  |  (S_1, ..., S_P):=Partitionnement_Dynamique(V_Nmin, P);
5  |  foreach ( p ∈ {1, ..., P} ) do in parallel
6  |  |  foreach ( x ∈ S_p ) do
7  |  |  |  foreach ( y voisin de x ) do
8  |  |  |  |  if ( y ∈ V_min ) then
9  |  |  |  |  |  if ( {x, y} est une arête M-bord) then
10 |  |  |  |  |  |  // abaissement
11 |  |  |  |  |  |  F[{x, y}]:=VF[y];
12 |  |  |  |  |  |  VF[x]:=F[{x, y}];
13 |  |  |  |  |  |  // mise à jour du sous-graphe des minima
14 |  |  |  |  |  |  E_min:= E_min ∪ {x, y} ;
15 |  |  |  |  |  |  V_min:= V_min ∪ {x} ;
16 |  |  |  |  |  |  // supprimer le sommet traité de l'ensemble des
   |  |  |  |  |  |  sommets non-minima V_Nmin
17 |  |  |  |  |  |  V_Nmin:= V_Nmin \ {x} ;
```

L'Algorithme 16 proposé consiste à traiter en parallèle les différents sommets non minima du graphe de départ. En effet, comme il est noté dans la boucle de la ligne 4 de l'algorithme 16, pour chaque sommet non minimum ayant un sommet voisin appartenant à l'ensemble des sommets minima, si l'arête liant ces deux sommets est une arête M-bord, le sommet non

minimum ainsi que l'arête M-bord détectée sont abaissés (lignes 9-10) et le reste des voisins ne sont pas examinés. Enfin, une mise à jour du sous-graphe des minima est effectuée (lignes 11-12).

A chaque itération, les sommets non minima sont répartis sur *P* processeurs. Cette répartition se fait d'une manière dynamique selon l'algorithme 17 de distribution de la charge. En effet, du fait de la variation du nombre de sommets non-minima d'une itération à l'autre, cet algorithme permet d'affecter dynamiquement des charges équilibrées aux *P* processeurs considérés.

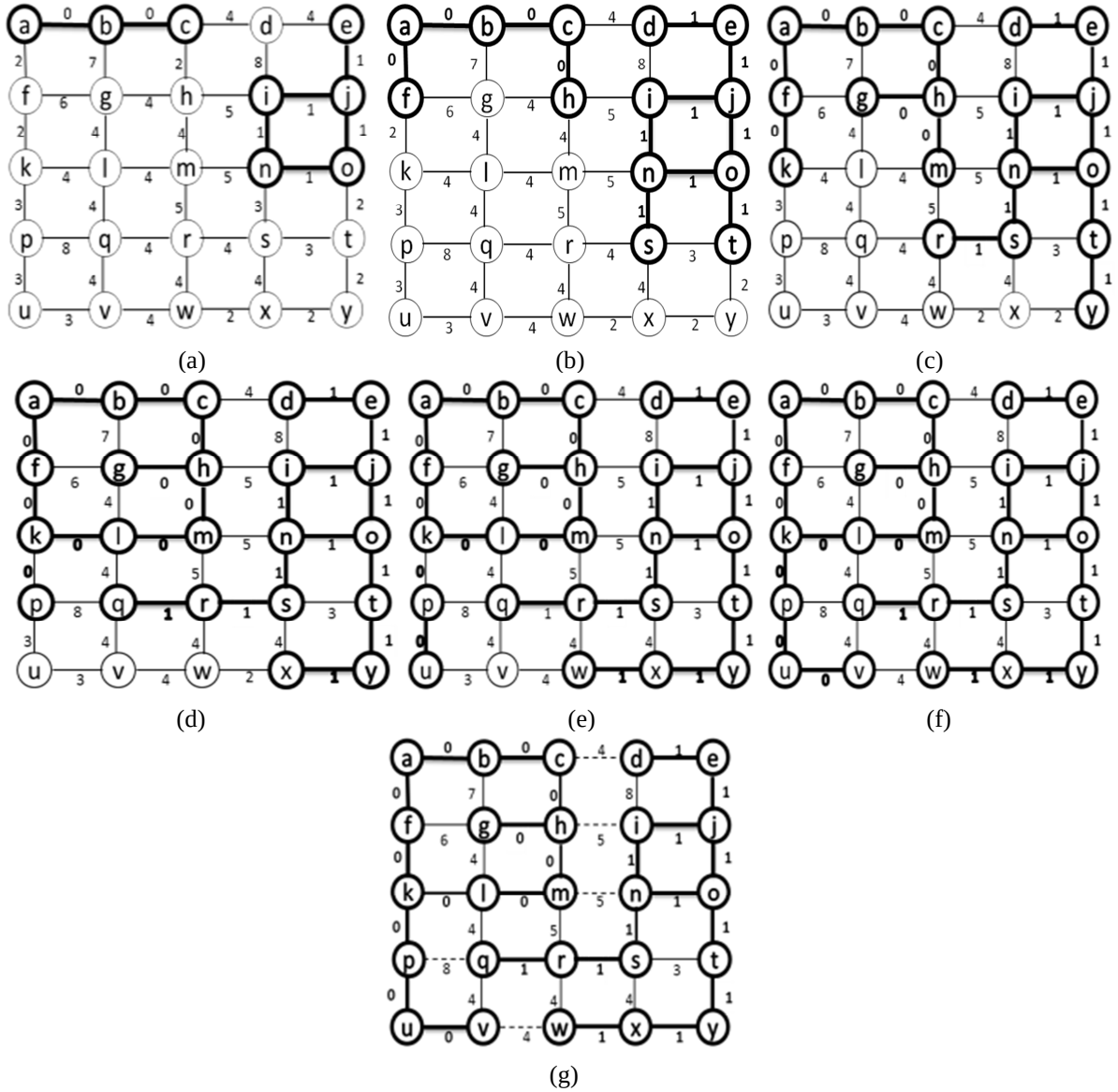


Figure 3. 7 (a) Un graphe G (V,E,F) et Min le sous-grzphe des minima représentés en gras (b), (c), (d), (e) et (f) les étapes de l'application de l'algorithm 16 sur le graphe G en utilisant P=3 processeurs et (g) le resultat final obtenu avec la LPE d'arêtes représentées en traits interrompus

Pour illustrer l'algorithme 16, nous présentons, tout d'abord, un exemple d'application de cet algorithme. Ensuite, nous présentons les différents cas de figures possibles que nous pouvons rencontrer en appliquant ce dernier sur un Graphe à Arêtes Valuées.

Prenons comme exemple le graphe à arêtes valuées (et le sous-graphe des minima correspondant) de la figure 3.7 (a). L'application de l'algorithme 16, en utilisant $P=3$, sur ce graphe conduit à l'obtention du résultat final en cinq itérations.

Deux cas peuvent être rencontrés en appliquant l'algorithme 16 sur un graphe à arêtes valuées:

- Dans le premier cas, un sommet non minimum n'a qu'un seul sommet voisin étiqueté comme sommet minimum. Dans ce cas, si l'arête liant ces deux sommets est une arête M-bord, alors, l'arête et le sommet non-minima sont abaissés, sinon l'abaissement n'est pas appliqué et l'arête est une arête séparante.
- Dans le second cas, un sommet non minimum possède plus qu'un sommet voisin étiqueté comme un sommet minimum et l'arête qui les lie est une arête M-bord. Dans ce cas, un seul sommet adjacent et une arête M-bord sont abaissés.

La figure 3.8 détaille ces cas. Elle représente en (a) un graphe à arêtes valuées avec le sous-graphe des minima associé représenté en gras. Les figures 3.8 (b) et 3.8 (c) illustrent le premier cas. Le sommet *m* est adjacent à un unique sommet minimum qui est le sommet *n*, mais l'arête {*mn*} n'est pas une arête M-bord. Ainsi, ni l'arête {*mn*} ni le sommet *m* ne sont abaissés comme le montre la figure 3.8 (b). Cependant, dans la figure 3.8 (c) le sommet non minima *f* est adjacent à un unique sommet minimum qui est le sommet *a*. Puisque l'arête liant ces deux sommets est une arête M-bord, l'arête {*fa*} et le sommet *f* sont tous les deux abaissés comme le montre la figure 3.8 (c).

Le second cas est illustré par la figure 3. 8 (d). Le sommet non minimum *d* est adjacent à trois sommets minima qui sont *e*, *i* et *c*. Par conséquent, seul l'arête {*de*} qui est une arête M-bord est abaissée ainsi que le sommet *d*.

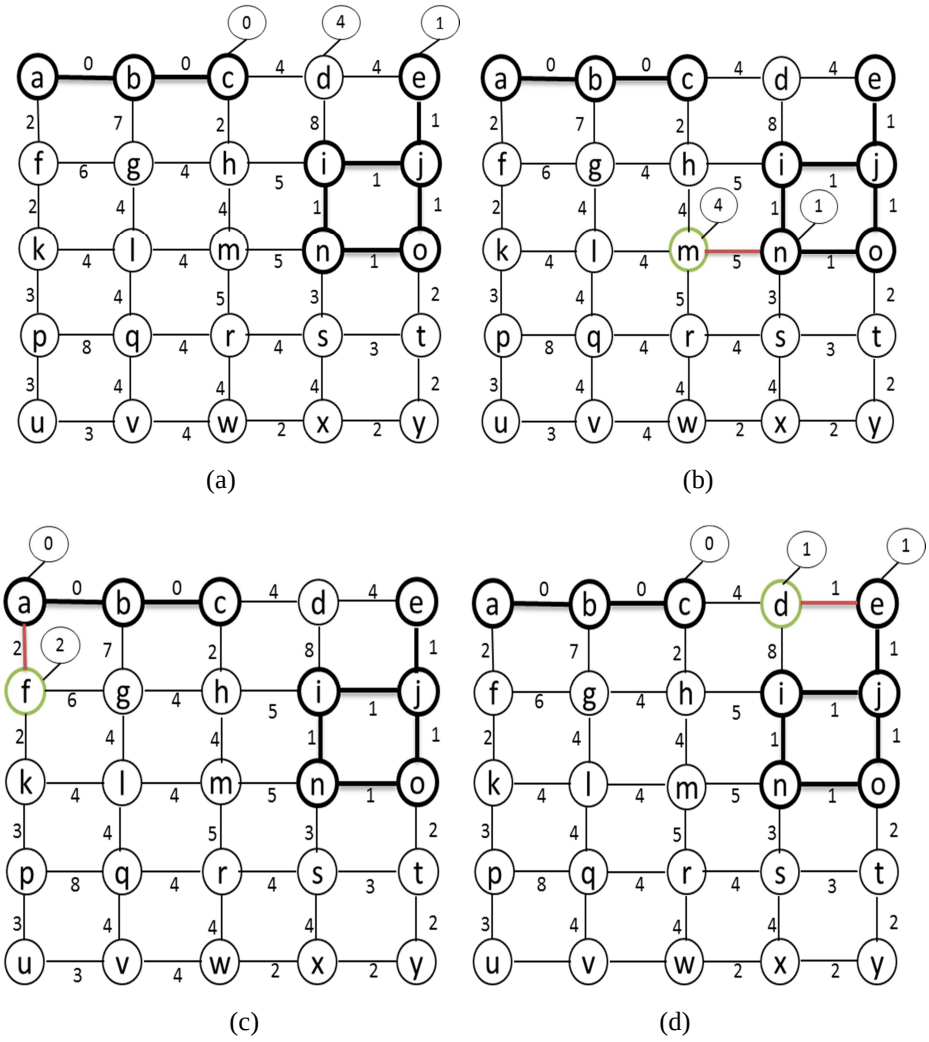


Figure 3. 8 (a) un graphe à arêtes valuées et ses minima (en gras), (b) cas d'un sommet non minimum adjacent à un seul sommet minimum mais l'arête qui les lie n'est pas M-bord (c)cas d'un sommet non minimum adjacent à un seul minimum et abaissement de l'arête M-bord qui les lie, (d) cas d'un sommet non minimum adjacent à trois sommets minima

La stratégie de parallélisation de l'algorithme de calcul de noyau par M-bord proposée dans cette section utilise un algorithme de partitionnement dynamique qui garantit l'équilibrage de la charge entre les différents processeurs à chaque itération. De plus, cette stratégie permet d'éviter le traitement alternée des données. L'algorithme proposé est adapté, grâce à la granularité importante de parallélisme qu'il permet, à un parallélisme massif.

7. Algorithme de partitionnement dynamique

L'algorithme de partitionnement dynamique (Algorithme 17) que nous avons utilisé consiste à calculer, à chaque itération, et en utilisant P processeurs une partition équilibrée $\{S_1, \ldots, S_P\}$ d'un ensemble S de données (d'arêtes ou de sommets) à traiter en parallèle. Pour l'algorithme parallèle de calcul du noyau par M-bord basé sur le traitement des arêtes d'une

façon alternée (Algorithme 15), l'algorithme 17 est utilisé pour partitionner, à chaque itération, les arêtes appartenant à chaque sous-ensemble sur *P* processeurs. Dans ce cas, l'ensemble *S* contient donc l'ensemble des arêtes à traiter en parallèle à chaque itération. Pour l'algorithme parallèle basé sur l'examen des sommets du graphe de départ (Algorithme 16), l'algorithme 17 est utilisé, à chaque itération, pour partitionner les sommets non minima à traiter en parallèle sur *P* processeurs. Donc, dans ce cas, l'ensemble *S* contient les sommets à traiter à chaque itération.

Soit $Q \in \mathbb{N}$ le quotient de la division de la taille de l'ensemble *S* (|*S*|) contenant les données à traiter en parallèle par le nombre de processeurs *P* et soit $R \in \mathbb{N}$ le reste de cette division.

L'algorithme de partitionnement dynamique produit des partitions équilibrées $\{S_1, \ldots, S_p\}$ dans le sens où les R-premiers ensembles de la partition contiennent $(|S| / P) + 1$ éléments tandis que les suivants contiennent $(|S|/P)$ éléments. Les éléments de *S* stockés dans un tableau sont déplacés vers des tableaux précédemment affectés aux sous-ensembles $\{S_1, \ldots, S_p\}$ dans l'ordre de leurs indices : le premier ensemble reçoit les premiers éléments du tableau *S* et ainsi de suite (voir figure 3. 9). Ainsi, chaque processeur calcule l'indice du premier et du dernier élément qui délimite la zone sur laquelle il doit agir.

Algorithme 17 : Partitionnement_Dynamique[72]

Data : *S* ensemble de données à traiter en parallèle et *P* nombre de processeurs.
Result : $(S_1, \ldots, S_P)$: ensemble de données à traiter par chaque processeur.

```
1  //Initialisation
2  Q= |S|/P;
3  R= |S|%P;
4  foreach (Processeur p ∈ {1, ...,P}) do in parallel
5      if (p ≤ R) then
6          start[p]:=(p-1)*(Q+1);
7          end[p]:=start[p] + Q;
8      else
9          start[p]:=(p-1)*Q+R;
10         end[p]:=start[p]+ Q - 1;
11     S_p:=S[start[p], ...,end[p]];
```

La figure 3.9 illustre l'exécution de cet algorithme. La figure 3. 9 (a) représente le cas où l'indice des processeurs est toujours supérieur à *R* (|*S*|=*20* et *P=4* donc *Q=5* et *R=0*). Dans ce

cas, les différentes partitions sont équilibrées et de même taille ($|S|/P$= 5). La figure 3. 9 (b) représente le cas où l'indice des trois premiers processeurs est inférieur à R. Dans ce cas, $|S|$=27 et P=4, donc, Q=6 et R=3. Ainsi, les trois premières partitions obtenues contiennent ($|S|/P$)+1=7 éléments et la quatrième partition contient $|S|/P$=6 éléments.

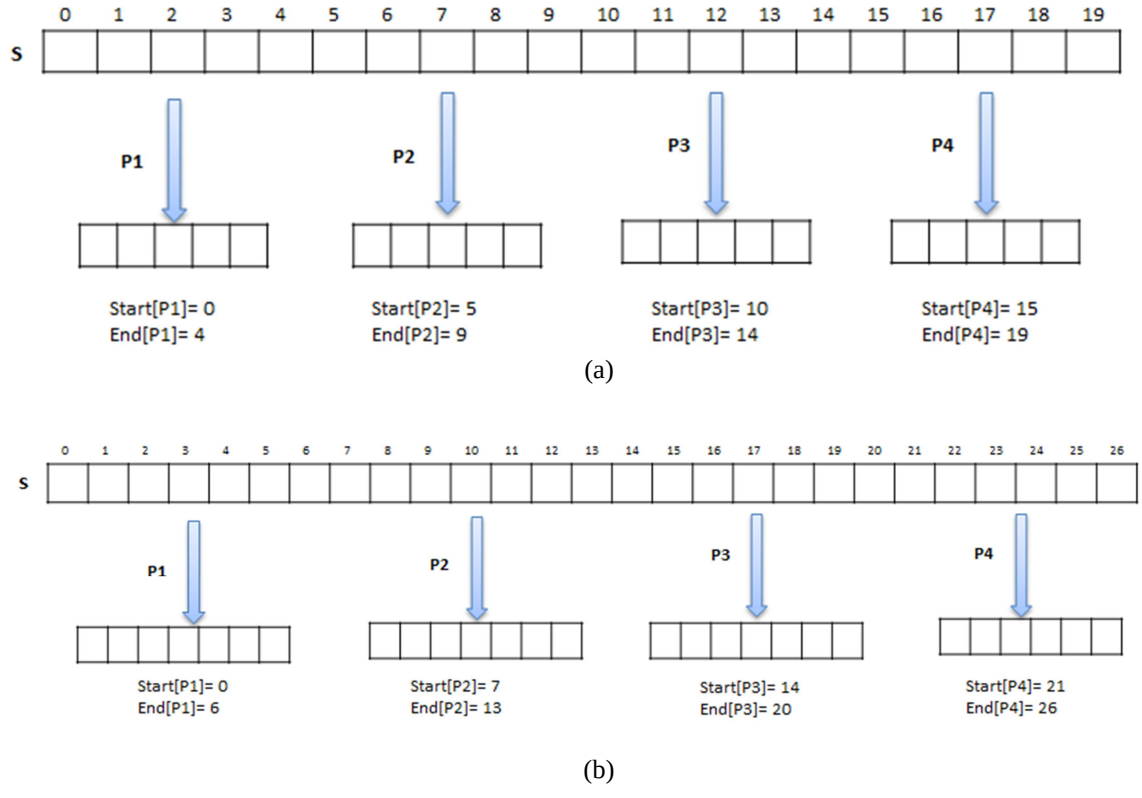


Figure 3. 9 Illustration de l'algorithme de distribution dynamique de la charge sur P processeurs (Algorithme 17) (a) cas où R=0 et (b) cas où R≠0

Après avoir présenté les différents algorithmes parallèles proposés, nous passons dans la section suivante à discuter les résultats obtenus suite à leur implémentation sur une architecture parallèle.

8. Implémentation et résultats

Dans cette section, nous présentons la plateforme sur laquelle les différents algorithmes parallèles, que nous avons proposés et discutés, ont été implémentés. Nous décrivons aussi, l'environnement logiciel de programmation et les mesures utilisées pour l'évaluation des performances. Ensuite, nous donnons les détails concernant la base d'images en niveau de gris que nous avons utilisée pour évaluer les approches de parallélisation proposées. Enfin, les évaluations des différentes stratégies proposées pour la parallélisation de l'algorithme de calcul de noyau par M-bord sont présentées et discutées.

8.1 Plateforme d'évaluation

Le choix de la plateforme d'implémentation ainsi que l'interface de programmation utilisée pour l'implémentation des différents algorithmes parallèles proposés dans ce manuscrit est motivé d'une part par l'étude de l'état de l'art présentée dans le chapitre 2 de ce manuscrit et d'autre part par la nature des algorithmes à implémenter. En effet, le modèle de programmation à mémoire partagée possède beaucoup d'avantages tels que la facilité d'implémentation, l'usage efficace de la mémoire et le faible accroissement de la taille du code. Dans ce cadre, OpenMP (Open Multi-Processing) [64,65] représente une des solutions les plus utilisées pour la programmation en mémoire partagée. C'est une API (Application Programming Interface) constituée d'un ensemble de directives de compilation. Il s'agit d'une extension aux langages C/C++ et Fortran et est supportée par la plupart des compilateurs. La communication par thread est implicite et utilise des variables qui pointent sur des emplacements de mémoire partagée [64, 65]. Nous avons mené les expérimentations sur un processeur Intel Xeon E5-2640 v3 ayant 8 cœurs, une fréquence de base de 2,6 GHz et une cache intelligente de 20 Mo, en utilisant OpenMP comme une extension au langage C/C ++ pour le support du modèle de mémoire partagée.

La performance des différents algorithmes parallèles est évaluée par les mesures de temps d'exécution et donc, du facteur d'accélération relative défini dans la section 7.2 du chapitre 2 de ce manuscrit. Rappelons que l'accélération relative représente le gain en temps d'exécution d'une tâche exécutée en utilisant *P* processeurs par rapport à son exécution en utilisant un seul processeur sur la même architecture. Elle est ainsi donnée par:

$$\textit{Accélération Relative} = \frac{T_{1CPU}}{T_{PCPU}} \qquad (3.1)$$

où T_{1CPU} désigne le temps d'exécution en utilisant un seul processeur et T_{PCPU} désigne le temps d'exécution de l'algorithme en utilisant *P* processeurs.

Pour des raisons de simplification, l'accélération relative sera notée accélération tout court dans le reste de ce manuscrit.

8.2 Base d'images de test

La base d'images que nous avons utilisée pour évaluer les performances des algorithmes parallèles est présentée dans le tableau 3.2. Elle contient des images de différentes tailles représentant des scènes variées. Elle est composée de 60 images 2D en niveau de gris de

différentes tailles (64×64, 128×128, 256×256, 512×512, 1024×1024 et 2048×2048). Les images ont été sélectionnées parmi les images standards fréquemment utilisées dans la littérature (Lena, cameraman, poivrons ...).

Nombre d'images	10	10	10	10	10	10
Résolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Nombre de sommets dans le graph correspondent	4096	16384	65536	262144	1048576	4194304
Nombre d'arêtes dans le graph correspondent	8192	32768	131072	524288	2097152	8388608
La plage du nombre des minima pour les images de chaque taille	487–12	4065-13	16220–17	67146-36880	253886-165548	408764 – 879834
La moyenne du nombre des minima	277	1727	11994	51365	216702	619010

Tableau 3. 2 Description de la base d'images sur laquelle les algorithmes séquentiels et parallèles sont appliqués

8.3 Evaluation des algorithmes parallèles de calcul de noyau par M-bord

L'évaluation des différents algorithmes parallèles de calcul de noyau par M-bord est faite dans un premier temps, en termes de nombre de cœurs et puis, dans un second temps, en termes de résolution des images. Pour l'évaluation en termes de nombre de cœurs, nous augmentons le nombre de cœurs de 2 à 8 et nous mesurons les temps d'exécution de chaque algorithme pour 10 images de taille 2048×2048. Pour l'évaluation en termes de résolutions des images, nous procédons à mesurer le temps d'exécution (et donc l'accélération) de chaque algorithme parallèle pour des images de différentes tailles en utilisant P processeurs avec P={2…8}. La base d'images utilisée pour l'évaluation en termes de résolution d'images et celle décrite dans le tableau 3. 2. Notons que, dans ce qui suit, le temps d'exécution (respectivement l'accélération) désigne la moyenne des temps d'accélération (respectivement accélérations) de 10 images de même taille.

8.3.1 Evaluation des performances en termes de nombre de cœurs

Dans cette section, nous évaluons les différents algorithmes parallèles de calcul de noyau par M-bord par rapport à la variation du nombre de cœurs. La figure 3.10 (respectivement 3.11 et 3.12) représente le temps d'exécution et l'accélération de l'algorithme 12

(respectivement Algorithme 15 et Algorithme 16). Ces figures prouvent que les différents algorithmes ont permis un gain en termes de temps d'exécution qui augmente lorsque le nombre de cœurs augmente. Par conséquent, le facteur d'accélération augmente également lorsque le nombre de cœurs augmente. Les meilleurs facteurs d'accélération sont ainsi obtenus en utilisant 8 cœurs. Ces facteurs atteignent 5.48 (respectivement 3.94 et 5.55) pour l'algorithme 12 (respectivement 15 et 16) pour des images de taille 2048×2048.

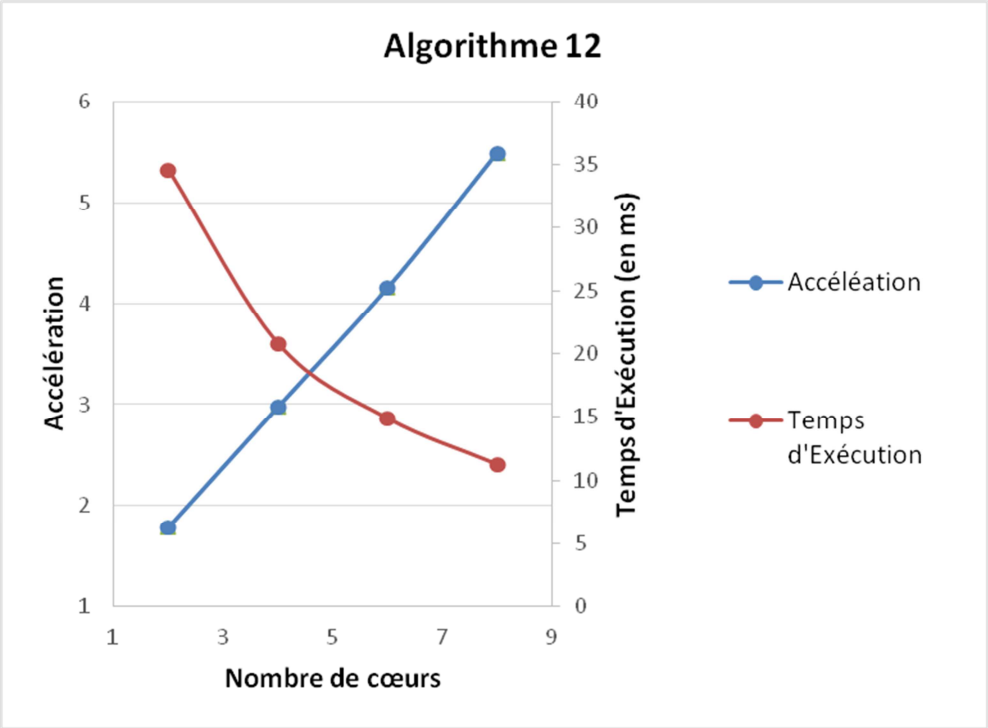


Figure 3. 10 Temps d'exécution et accélération de l'algorithme parallèle de calcul de noyau par M-bord par décomposition du domaine (Algorithme 12) en fonction du nombre de cœurs pour 10 images de taille 2048×2048

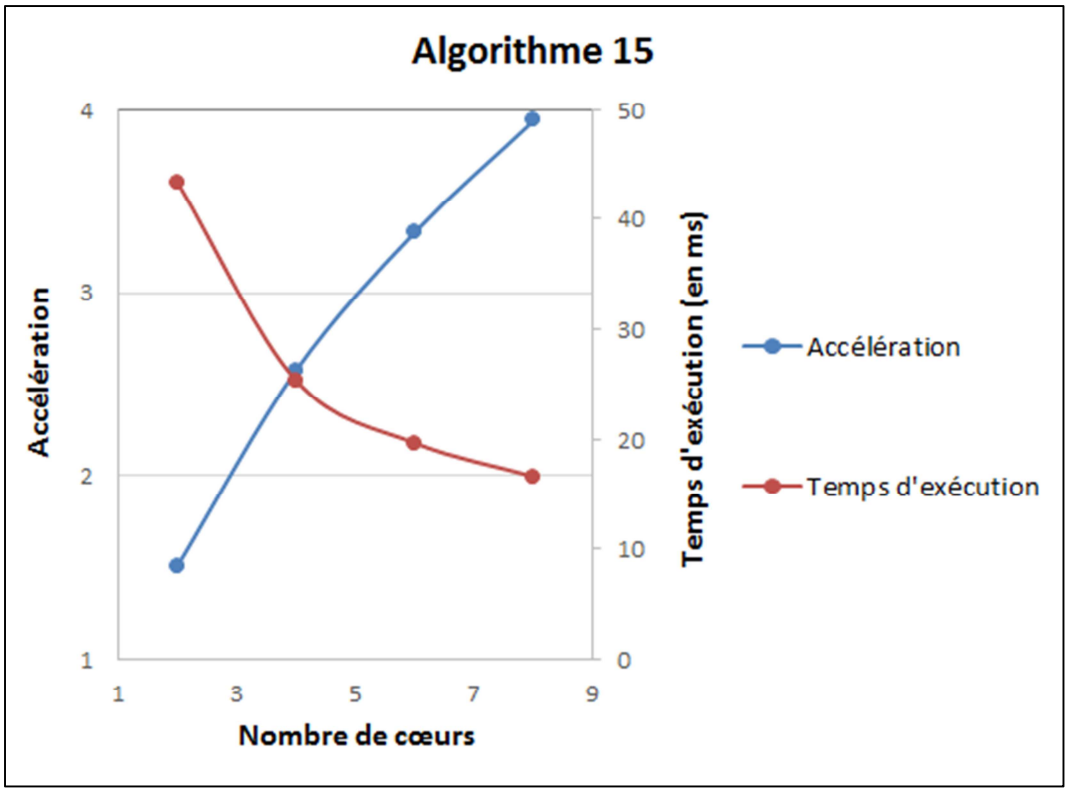


Figure 3. 11 Temps d'exécution et accélération de l'algorithme parallèle de calcul de noyau par M-bord basé sur le traitement des arêtes du graphe en alternance (Algorithme 15) en fonction du nombre de cœurs pour des images de taille 2048×2048

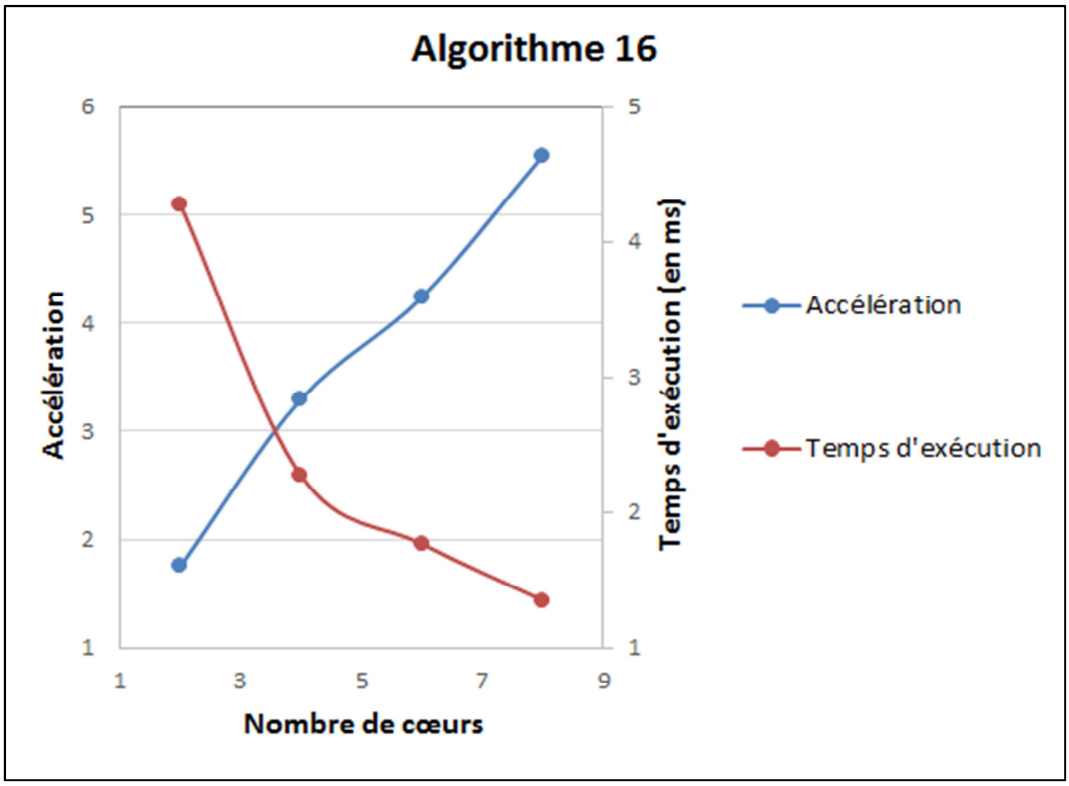


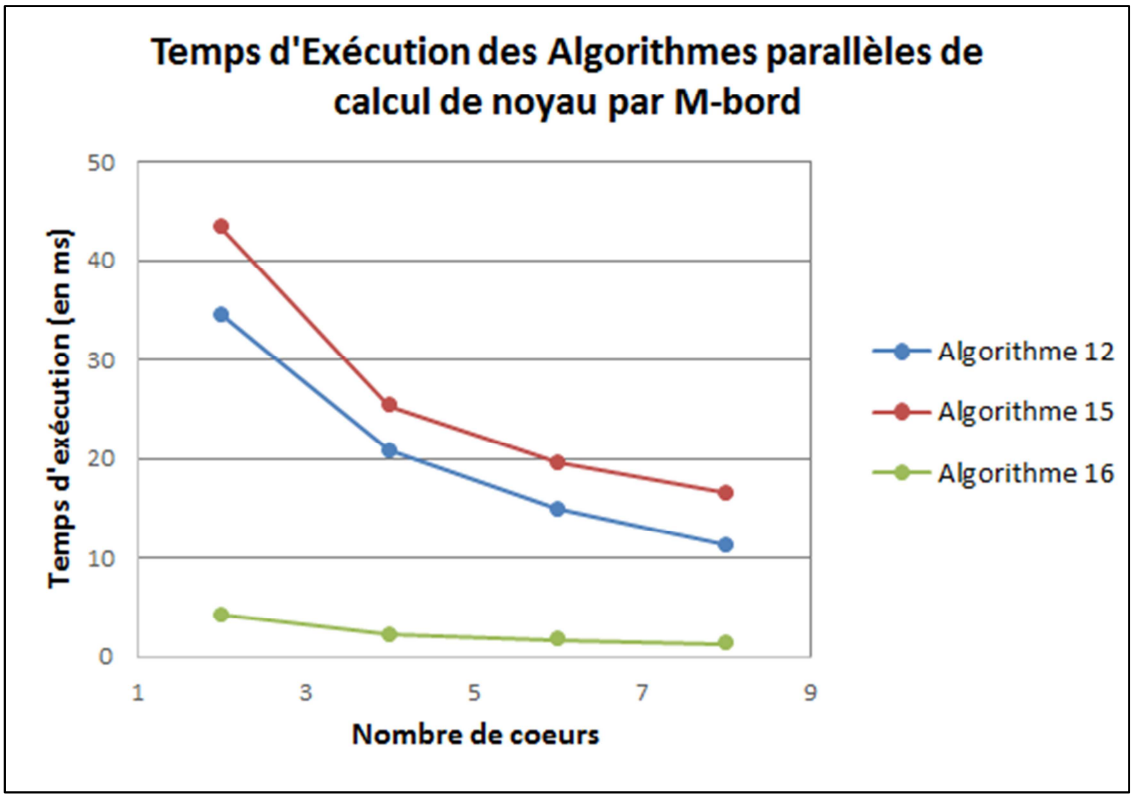
Figure 3. 12 Temps d'exécution et accélération de l'algorithme parallèle de calcul de noyau par M-bord basé sur l'examen des sommets du graphe (Algorithme 16) en fonction du nombre de cœurs pour des images de taille 2048×2048

Afin de comparer les performances des trois algorithmes parallèles de calcul de noyau par M-bord, proposés dans ce chapitre, la figure 3.13 représente l'accélération et le temps d'exécution de ces algorithmes (Algorithme 12 en bleu, Algorithme 15 en rouge et Algorithme 16 en vert) en utilisant P processeurs avec $P = \{2, \ldots, 8\}$ pour des images de taille 2048×2048.

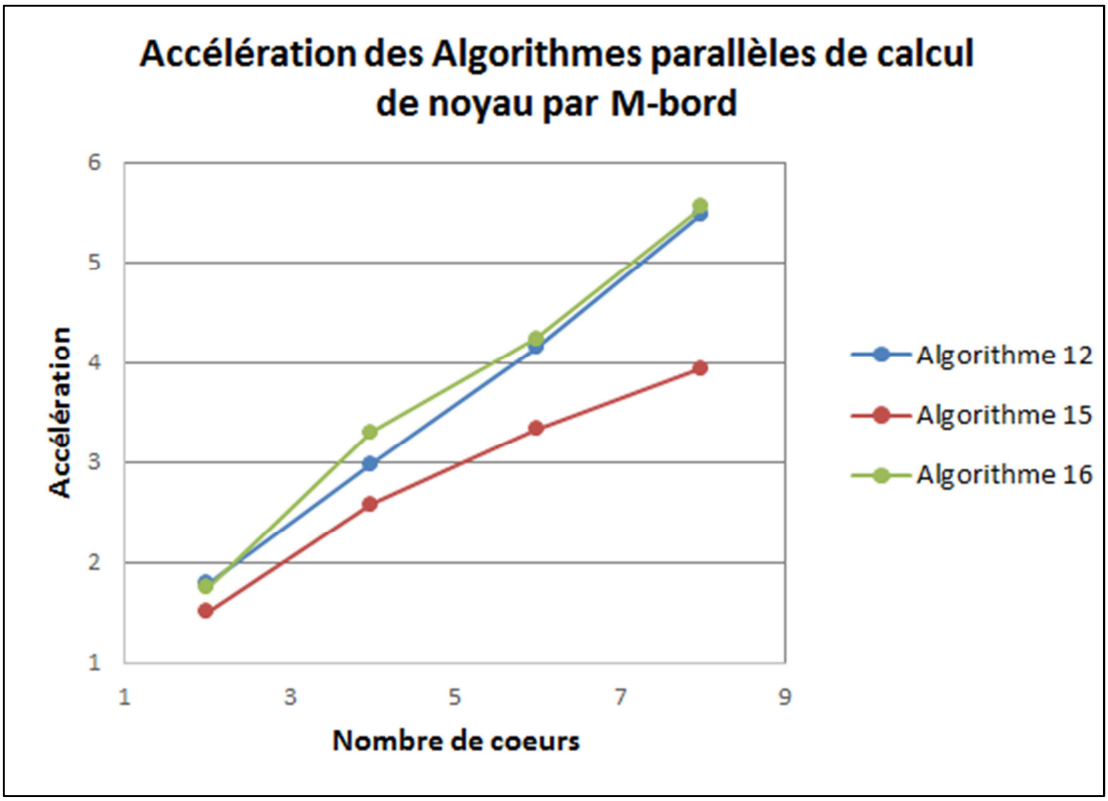
Ces figures montrent que l'algorithme parallèle basé sur l'examen des sommets du graphe de départ (Algorithme 16) a donné le meilleur temps d'exécution (le plus faible), et le meilleur facteur d'accélération parmi les trois algorithmes parallèles proposés.

8.3.2 Evaluation des performances en termes de résolution d'images

Dans cette section, nous visons à évaluer les performances des algorithmes parallèles proposés pour le calcul du noyau par M-bord en fonction de la taille des images. Pour ce faire, l'algorithme 12 (respectivement Algorithme 15 et Algorithme 16) est appliqué sur la base d'images en niveau de gris décrite dans le tableau 3. 2. Rappelons que pour chaque taille, le temps d'exécution (respectivement facteur d'accélération) mesuré représente la moyenne des temps d'exécution (respectivement facteurs d'accélération) de 10 images de cette taille.



(a)



(b)

Figure 3. 13 Temps d'exécution et accélérations des différents algorithmes parallèles de calcul de noyau par M-bord pour des images de taille 2048×2048

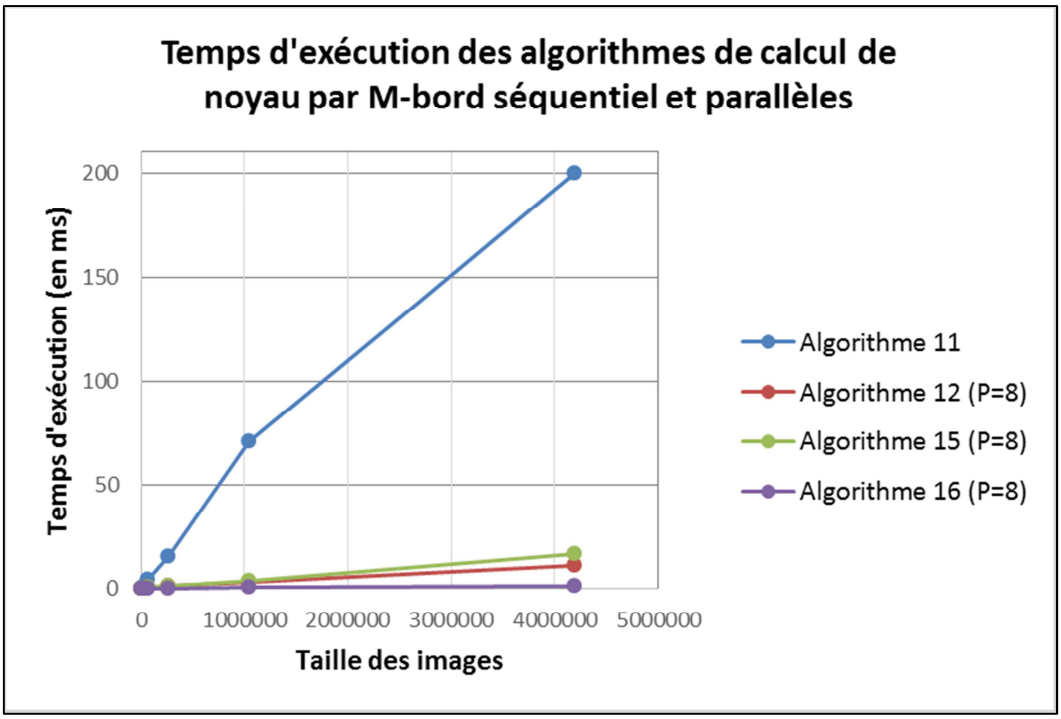


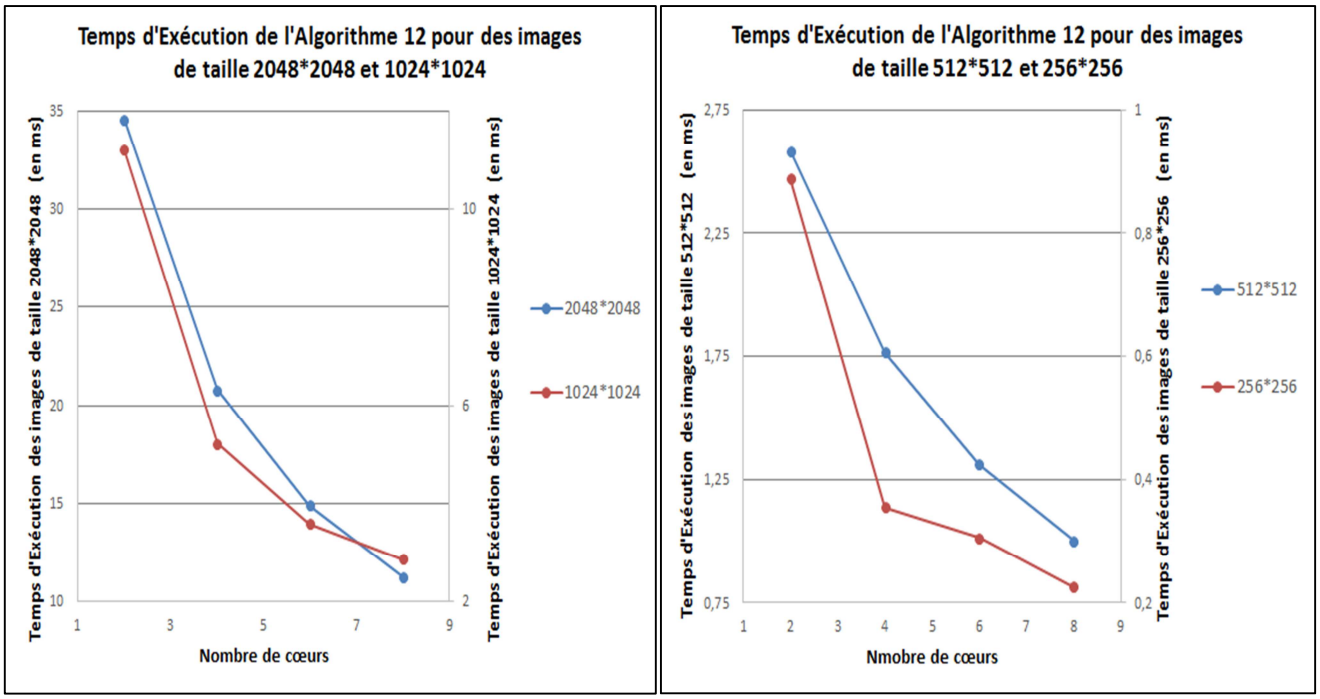
Figure 3. 14 Temps d'exécution des algorithmes de calcul de noyau par M-bord (séquentiel et parallèles) pour des images de différentes tailles

La figure 3.14 représente le temps d'exécution moyen de l'algorithme séquentiel (Algorithme 11) et des algorithmes parallèles (Algorithmes 12, Algorithme 15 et Algorithme 16 en utilisant 8 cœurs) de calcul du noyau par M-bord. D'après les courbes présentées dans cette figure, nous concluons que les différents algorithmes parallèles ont permis un gain en termes de temps d'exécution par rapport à l'algorithme séquentiel quel que soit la taille des images à segmenter. De plus, l'algorithme parallèle qui a donné le meilleur temps d'exécution est l'algorithme basé sur l'examen des sommets au lieu des arêtes du graphe (Algorithme 16).

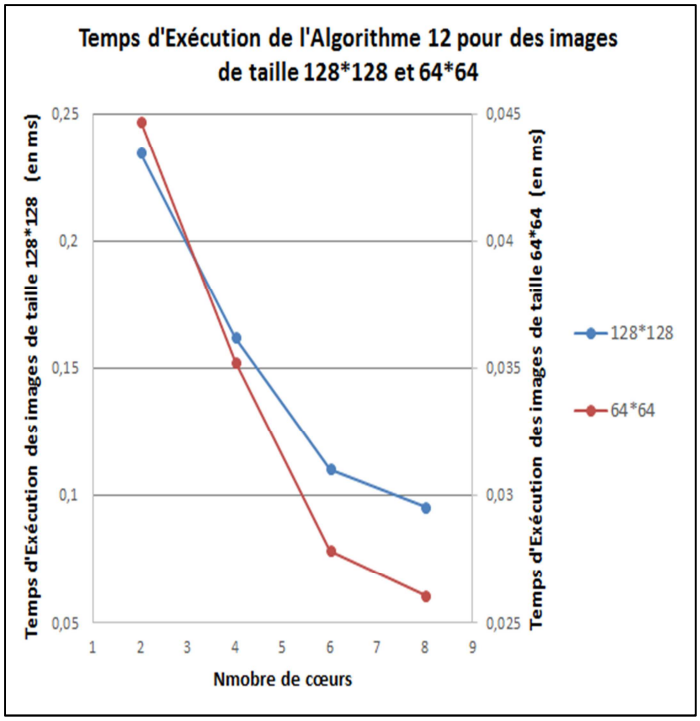
La figure 3.15 (respectivement figure 3.17 et figure 3.19) représente les courbes des temps d'exécution de l'algorithme 12 (respectivement Algorithme 15 et Algorithme 16) en fonction du nombre de cœurs pour des images de différentes tailles. Dans la figure 3.16 (respectivement figure 3.18 et figure 3.20), chaque courbe représente l'accélération de l'algorithme 12 (respectivement Algorithme 15 et Algorithme 16) en fonction du nombre de cœurs pour 10 images de même taille.

D'après ces figures, le temps d'exécution des différents algorithmes parallèles diminue en augmentant le nombre de cœurs de 2 jusqu'à 8 quel que soit la taille des images en entrée. Ceci est traduit par une augmentation des facteurs d'accélération pour les images de différentes tailles. Pour l'algorithme 12 l'accélération atteint 5.48 pour les images de taille

2048×2048, 5.49 pour les images de taille 1024×1024, 3.82 pour les images de taille 512×512, 4.14 pour les images de taille 256×256, 2.53 pour les images de taille 128×128 et 2.97 pour les images de taille 64×64 en utilisant 8 cœurs comme c'est indiqué dans le tableau 3.3. Pour l'algorithme 15, l'accélération atteint 3.94 pour les images de taille 2048×2048, 4.39 pour les images de taille 1024×1024, 3.73 pour les images de taille 512×512, 3.63 pour les images de taille 256×256, 2.49 pour les images de taille 128×128 et 2.29 pour les images de taille 64×64 en utilisant 8 cœurs comme il est indiqué dans le tableau 3.4. Pour l'algorithme 16, l'accélération atteint 5.55 pour les images de taille 2048×2048, 5.11 pour les images de taille 1024×1024, 4.45 pour les images de taille 512×512, 3.83 pour les images de taille 256×256, 3.05 pour les images de taille 128×128 et 3.03 pour les images de taille 64×64 en utilisant 8 cœurs comme il est indiqué dans le tableau 3.5.



(a) (b)



(c)

Figure 3. 15 Temps d'exécution moyen de l'algorithme 12 en fonction du nombre de cœurs pour des images de taille (a) 2048×2048 et 1024×1024 (b) 512×512 et 256×256 et (c) 128×128 et 64×64

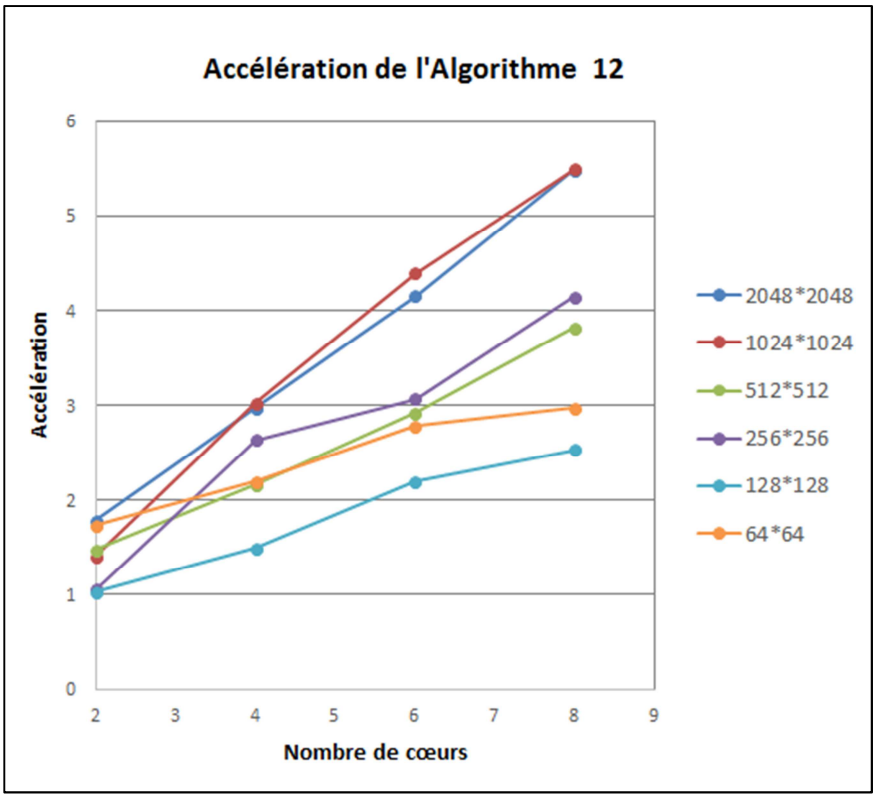
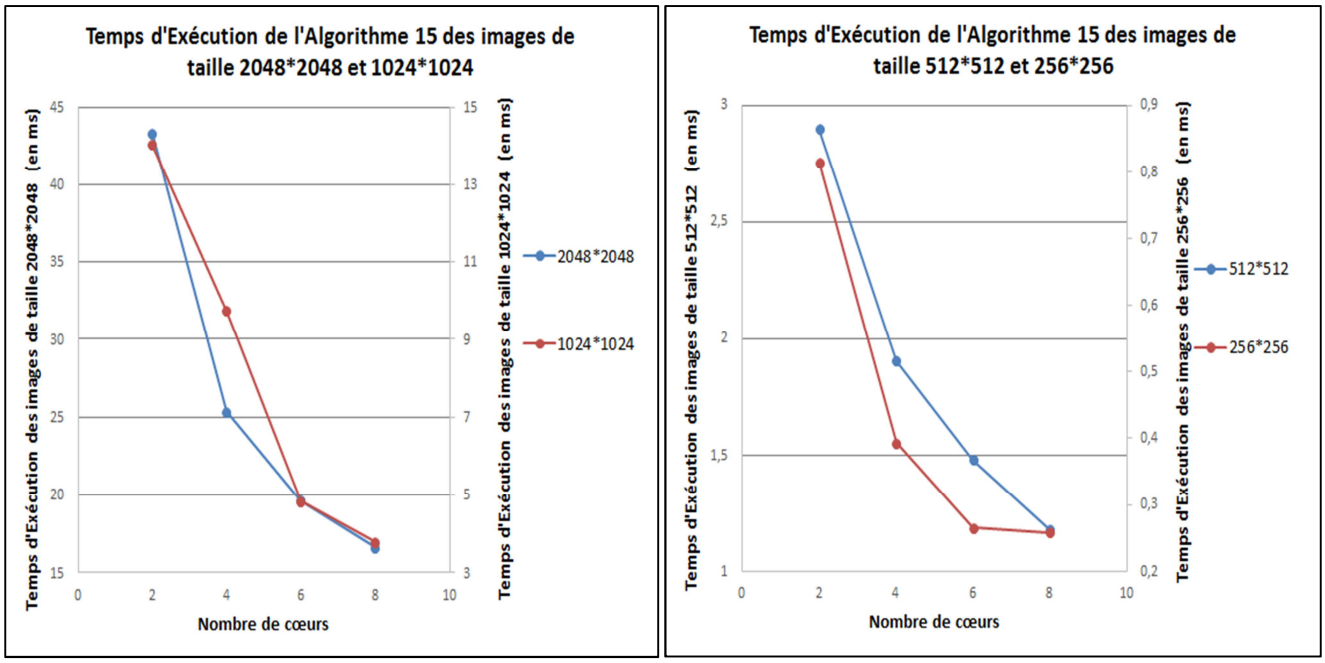


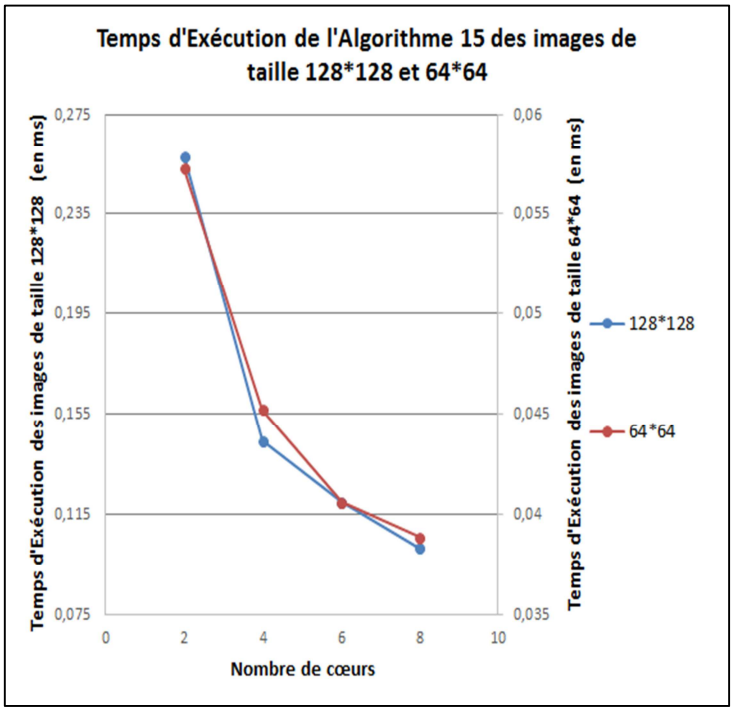
Figure 3. 16 Accélération de l'algorithme 12 en fonction du nombre de cœurs pour des images de différentes tailles

Résolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Accélération pour *P=8*	2.97	2.53	4.14	3.82	5.49	5.48

Tableau 3. 3 Accélérations de l'algorithme 12 pour des images de différentes tailles en utilisant 8 cœurs



(a) (b)



(c)

Figure 3. 17 Temps d'exécution moyen de l'algorithme 15 en fonction du nombre de cœurs pour des images de taille (a) 2048×2048 et 1024×1024 (b) 512×512 et 256×256 et (c) 128×128 et 64×64

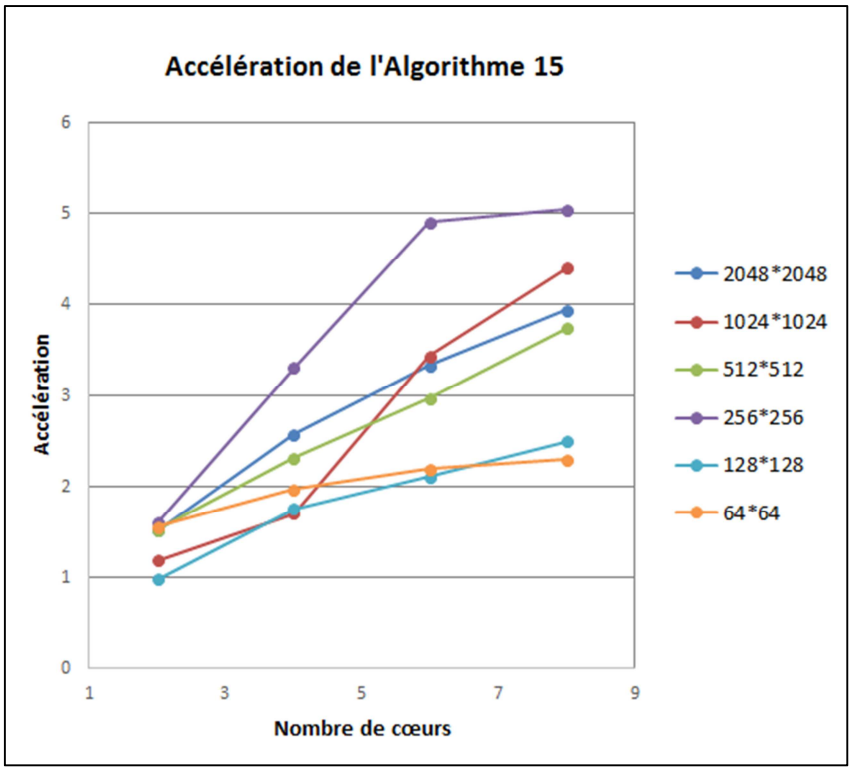
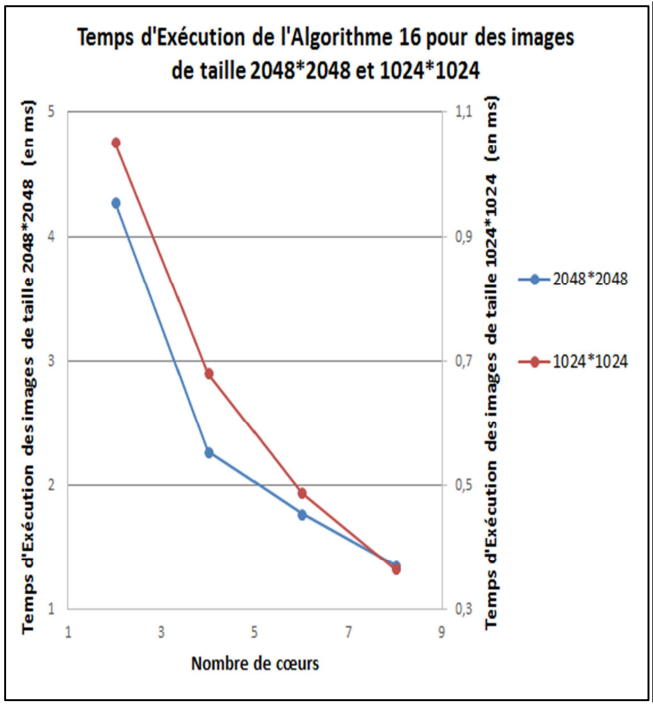


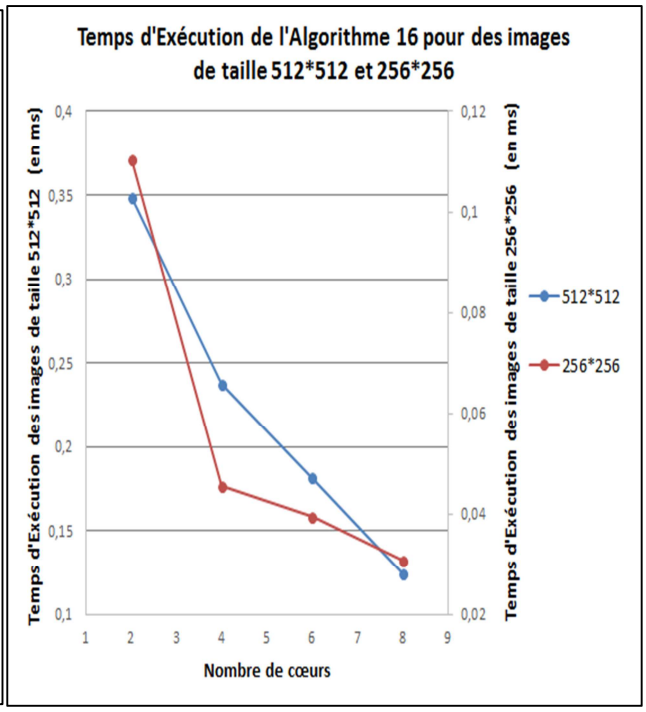
Figure 3. 18 Accélération de l'algorithme 15 en fonction du nombre de cœurs pour des images de différentes tailles

Résolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Accélération	2.29	2.49	3.63	3.73	4.39	3.94

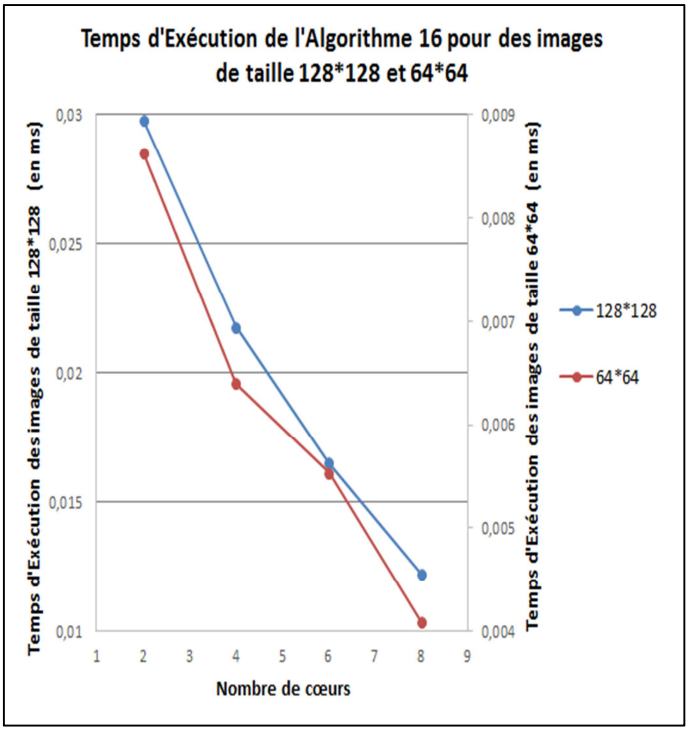
Tableau 3. 4 Accélérations de l'algorithme 15 pour des images de différentes tailles en utilisant 8 cœurs



(a)



(b)



(c)

Figure 3. 19 Temps d'exécution moyen de l'algorithme 16 en fonction du nombre de cœurs pour des images de taille (a) 2048×2048 et 1024×1024 (b) 512×512 et 256×256 et (c) 128×128 et 64×64

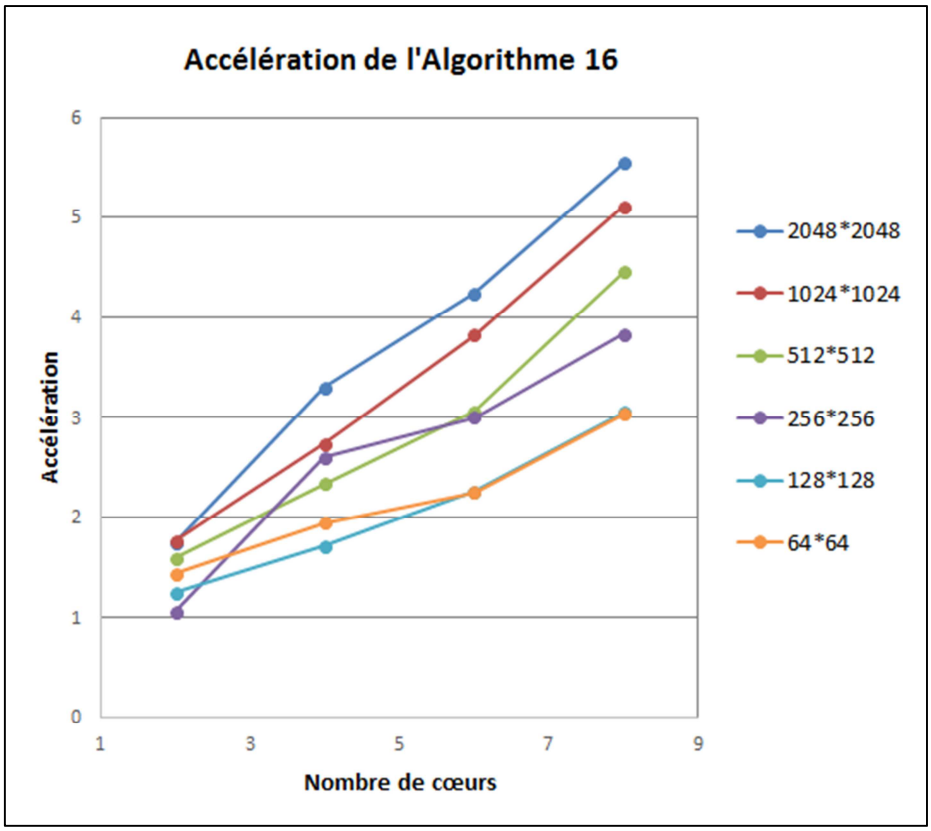


Figure 3. 20 Accélération de l'algorithme 16 en fonction du nombre de cœurs pour des images de différentes tailles

Résolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Accélération	3.03	3.05	3.83	4.45	5.11	5.55

Tableau 3. 5 Accélérations de l'algorithme 16 pour des images de différentes tailles en utilisant 8 cœurs

9. Conclusion

Dans ce chapitre, nous avons abordé la parallélisation de l'algorithme de calcul de noyau par M-bord. Dans ce cadre, nous avons étudié la problématique de dépendances des données que soulève cet algorithme et proposé trois stratégies pour sa parallélisation: la première consiste à traiter le graphe de départ sous forme de sous-domaines appelés partitions traités en parallèle. La deuxième consiste à traiter les arêtes du graphe de départ en alternance de façon à éviter la problématique de dépendance de données (abaissement de deux ou plusieurs arêtes M-bord ayant un sommet non-minimum commun). Enfin, la troisième consiste à examiner les sommets du graphe au lieu des arêtes ainsi les sommets non minima adjacents aux sommets minima sont traités en parallèle.

Ces différents algorithmes ont été implémentés sur une architecture multi-cœurs en utilisant l'interface de programmation applicative OpenMp adaptée aux architectures à mémoire partagée. Les résultats obtenus ont montré que l'algorithme parallèle basé sur l'examen des sommets du graphe de départ (Algorithme 16) a donné le meilleur temps d'exécution parmi les trois algorithmes parallèles proposés et le facteur d'accélération le plus important qui a atteint 5.99 pour les images de taille 2048×2048 en utilisant 8 cœurs.

Il est à noter que l'algorithme étudié dans ce chapitre (algorithme de calcul de noyau par M-bord) requiert deux étapes préliminaires qui sont la détection des minima régionaux et la pondération des sommets du graphe. De ce fait, l'exploitation opérationnelle de la parallélisation que nous avons proposée pour cet algorithme doit passer également par la parallélisation de ces deux étapes. Ceci fera l'objet du chapitre suivant.

Chapitre 4 : Implémentation parallèle d'une technique de segmentation basée sur le calcul du noyau par M-bord sur architecture multi-cœurs

1. Introduction

La segmentation d'images par la méthode de la Ligne de Partage des Eaux basée sur l'algorithme de calcul du noyau par M-bord, nécessite dans un premier temps la détection des minima régionaux. En effet, ces derniers constituent les points de départ à partir desquels le ravinement du relief commence. Une fois ces minima calculés, le processus d'abaissement sur lequel se base l'algorithme étudié nécessite une valuation particulière du graphe en particulier la valuation des sommets. De ce fait la parallélisation de la méthode de segmentation par LPE basée sur l'algorithme de calcul du noyau par M-bord doit inclure, outre la parallélisation de cet algorithme, la parallélisation de ces deux étapes préliminaires.

Dans ce chapitre nous nous intéressons dans un premier temps à la parallélisation de ces deux étapes. Nous présentons dans un deuxième temps une évaluation des performances de leur parallélisation, prise chacune individuellement ainsi qu'une évaluation de la performance de la parallélisation de la méthode de segmentation dans sa totalité.

2. Technique de segmentation basée sur l'algorithme de calcul du noyau par M-bord

Dans cette section, nous présentons la technique de segmentation des images en niveaux de gris basée sur le calcul de noyau par M-bord. Notons que cette technique est considérée dans ce travail dans le cadre des images 2D mais qu'elle peut être facilement étendue et appliquée aux images 3D ou 4D.

L'organigramme de la figure 4.1 résume les différentes étapes de cette technique. Cette dernière prend en entrée des images en niveau de gris représentées sous forme de GAV. La première étape consiste à calculer l'ensemble des arêtes minima du graphe d'entrée selon l'algorithme 18 (détaillé dans la section 2.1 de ce chapitre). Ensuite, les sommets du graphe sont pondérés en leurs attribuant la valeur minimale des arêtes auxquels ils appartiennent et l'ensemble des sommets minima est calculé à partir de l'ensemble des arêtes minima déjà calculé, et ce selon l'algorithme 19 (détaillé dans la section 2.2 de ce chapitre). On obtient donc le sous-graphe des minima ($Min = (E_{min}, V_{min})$). Enfin, l'algorithme de calcul du noyau par M-bord (Algorithme 11 détaillé dans la section 2 du chapitre 3 de ce manuscrit) est appliqué sur le graphe à arêtes valuées correspondant à l'image d'entrée et ses minima. Cet algorithme permet d'obtenir un noyau par M-bord à partir duquel est déduite la LPE d'arêtes.

En conclusion, pour un graphe à arêtes valuées G (V, E, F), l'approche peut être résumée en trois principales étapes :

(i) la recherche des minima régionaux de F,

(ii) la pondération des sommets et déduction de l'ensemble des sommets minima à partir de l'ensemble des arêtes minima,

(iii) le calcul du noyau par M-bord.

Ces différentes étapes seront détaillées dans les paragraphes suivants.

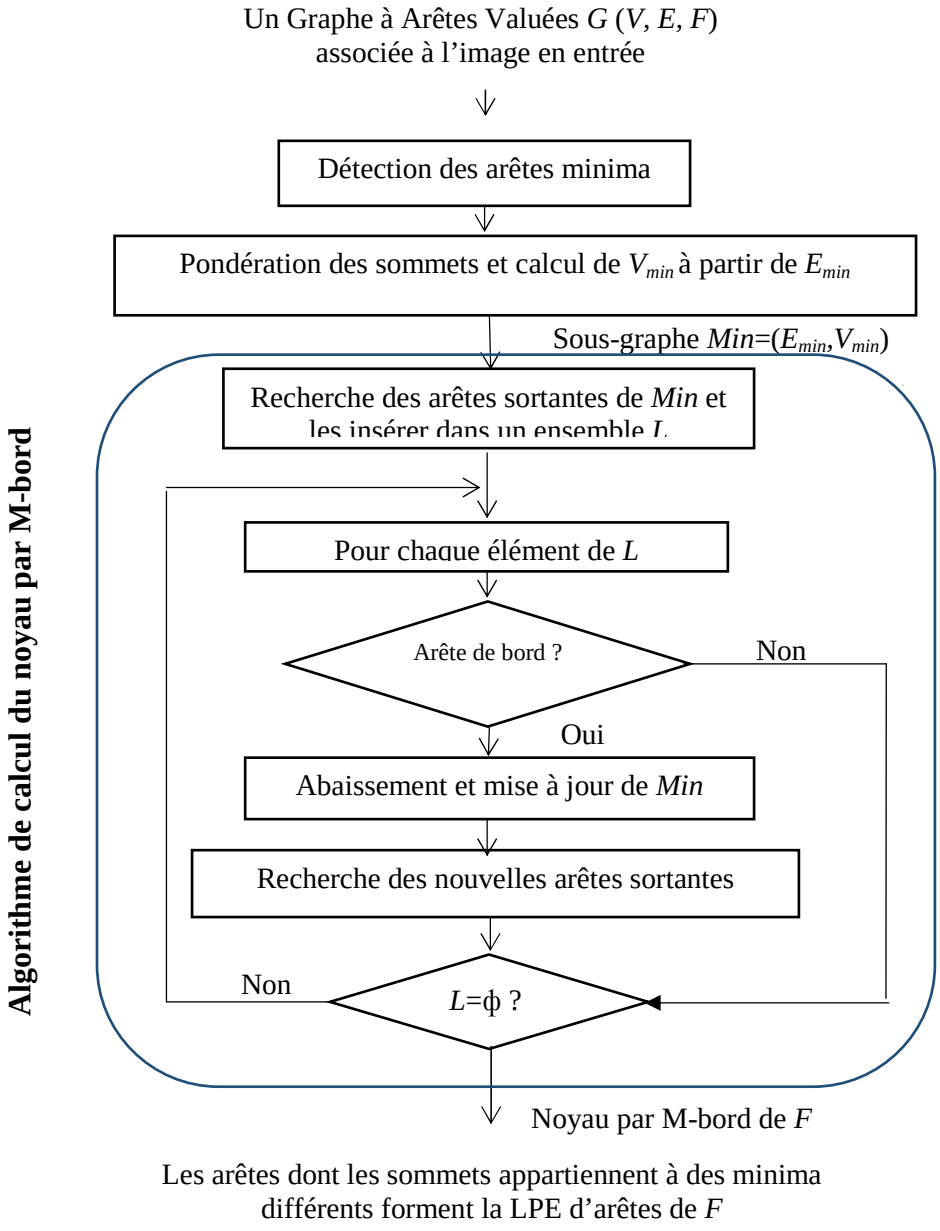


Figure 4. 1 La technique de segmentation basée sur le calcul du noyau par M-bord

2.1 Etape 1 : Détection des minima régionaux

L'étape de détection des minima est une étape préliminaire commune à plusieurs algorithmes de calcul de la LPE. Nous rappelons qu'un minimum régional se présente sous forme d'un plateau d'éléments dont la valuation est inférieure à celles de tous ses voisins (les éléments qui lui sont externes et connexes). La détection des minima régionaux en particulier et des extrema régionaux d'une manière plus générale est un traitement qui n'est pas local. En effet, on ne peut pas décider si un élément appartient à un minimum régional en examinant seulement ses voisins immédiats. Le parcours du plateau auquel appartient cet élément est nécessaire dans ce cas. Ceci requiert des explorations qui combinent des recherches en largeur et en profondeur dans le graphe représentant l'image en entrée. En effet, le graphe est parcouru élément par élément. A chaque fois qu'un élément est détecté comme un minimum local alors une recherche en profondeur est lancée afin de trouver le plateau auquel il appartient et décider ensuite si ce plateau est un minimum régional ou non.

L'algorithme 18 représente le pseudo-code de cette étape (détection des minima régionaux) dans le cadre d'un Graphe à Arêtes Valuées. Cet algorithme consiste à balayer l'ensemble des arêtes du graphe de départ comme indiqué dans la boucle itérative de la ligne 4 de l'algorithme 18. Le parcourt se fait ligne par ligne en partant du haut vers le bas (Raster Scan Order). Pour chaque arête *u*, le plateau auquel appartient *u* est parcouru en utilisant l'approche de parcours en profondeur d'abord (Depth First Scan) comme il est indiqué dans la boucle de la ligne 9 de l'algorithme 18. Si une arête voisine à *u* dont l'altitude est inférieure à celle de *u* est détectée, alors l'arête *u* ainsi que le plateau à laquelle elle appartient sont étiquetés comme étant des arêtes traitées, mais non minima (comme indiqué dans les lignes de 12 à 16 de l'algorithme 18). Sinon, le plateau auquel appartient l'arête *u* est étendu et est étiqueté comme minimum régional (lignes 17-20 de l'algorithme 18).

Algorithme 18 : Calcul des minima régionaux [111]

```
Data : (V, E, F) un graphe à arête valuées
Result : E_min: L'ensemble des minima régionaux de F
// Initialisation
L := φ;
E_min := φ;
foreach ( u ∈ E ) do
    if (u n'est pas encore traitée) then
        E_min := E_min ∪ {u};
        L := L ∪ {u};
        //Parcourir le plateau auquel appartient u
        while ( L ≠ φ ) do
            foreach ( w ∈ L ) do
                L := L \ {w};
                foreach ( v ∈ E voisine de w) do
                    if ( w ∈ E_min & F[v] < F[w] ) then
                        //w est traitée mais n'est pas
                        //étiquetée comme arête minimum
                        E_min := E_min \ {w}; L := L ∪ {w};
                    else if ( F[v] == F[w] ) then
                        if ( ( w ∈ E_min & v non encore traitée ) || ((w
                          traitée et ∉ E_min) & (v non traité ou ∉ E_min) ) )
                          then
                            E_min := E_min ∪ {v};
                            L := L ∪ {v};
```

L'application de cet algorithme est illustrée par la figure 4.2. La figure 4.2 (a) présente un GAV et la figure 4.2 (b) présente le résultat de l'application de l'algorithme 18 sur ce graphe.

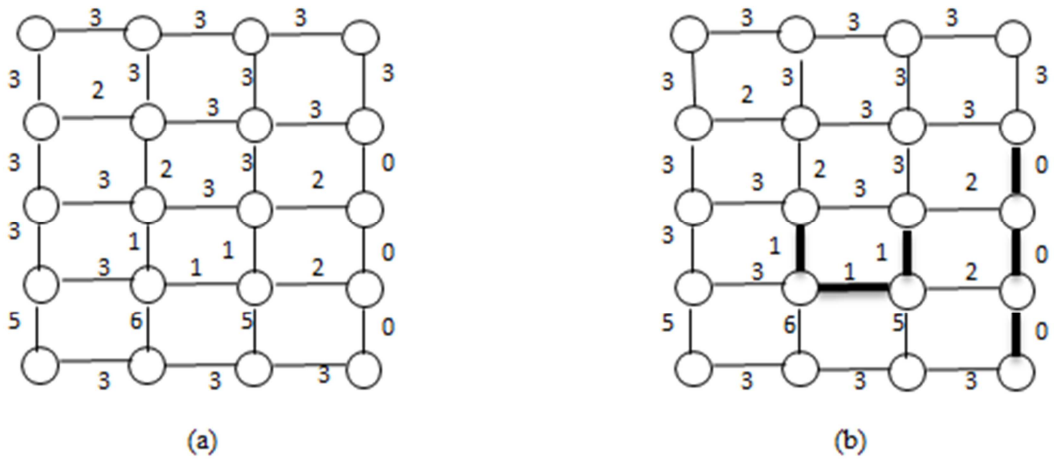


Figure 4. 2 (a) Un Graphe à Arêtes Valuées G et (b) les arêtes minima obtenus en appliquant l'algorithme 18 sur le graphe G

2.2 Etape 2 : Pondération des sommets et calcul de l'ensemble des sommets minima

La fonction de pondération des sommets selon [17] consiste à attribuer à chaque sommet $x \in V$ la valeur minimale des arêtes auxquelles il appartient (*cf.* Algorithme 19). Pour ce faire, les sommets sont d'abord initialisés en leur attribuant la valeur maximale qu'ils peuvent avoir (niveau de gris maximal : 255). Ensuite, ces sommets sont parcourus ligne par ligne (Raster Scan Order). Pour chaque sommet *x*, les arêtes auxquelles il appartient sont explorées itérativement et à chaque fois l'altitude minimale de ces arêtes est affectée à *x*. Le parcours des sommets est également exploité pour déterminer l'ensemble des sommets minima. Dans ce cadre, s'il existe une arête voisine *v* étiquetée comme faisant partie d'un minimum régional ($v \in E_{min}$), alors *x* est étiqueté comme sommet appartenant à un minimum régional et il est ainsi ajouté à l'ensemble V_{min}.

Algorithme 19 : Pondération des sommets et calcul de l'ensemble de sommets minima à partir de l'ensemble des arêtes minima

Data : (V, E, F) un graphe à arête valuées et E_{min} ensemble d'arêtes minima
Result : VF pondération des sommets et V_{min} ensemble de sommets minima.

```
1 // Initialisation
2 V_min:=φ;
3 foreach ( x ∈ V ) do
4 |  VF[x]:=255 ;
5 |  foreach ( y ∈ V voisin de x ) do
6 |  |  if ( F[{x,y}] < VF[x] ) then
7 |  |  |  VF[x]:=F[{x,y}];
8 |  |  if ( {x,y} ∈ E_min ) then
9 |  |  |  V_min := V_min ∪ {x};
```

3. Parallélisation de la technique de segmentation basée sur le calcul du noyau par M-bord

Dans cette section, nous proposons une stratégie de parallélisation de la technique de segmentation que nous avons présentée dans la section précédente. Il est à noter que lorsqu'il s'agit d'une séquence de traitements à effectuer, deux types de stratégies de parallélisation peuvent être envisagés:

- La première stratégie consiste à considérer la séquence comme un seul traitement. Sa parallélisation est alors faite d'une manière globale.
- La deuxième stratégie consiste à paralléliser individuellement chacune des étapes de la séquence.

Dans le cas de la technique de segmentation que nous considérons, chacune des deux dernières étapes qui la composent requiert le résultat global de l'étape qui la précède. De ce fait, c'est la deuxième stratégie qui s'avère la plus appropriée pour sa parallélisation. L'application de cette stratégie est illustrée par la figure 4.3.

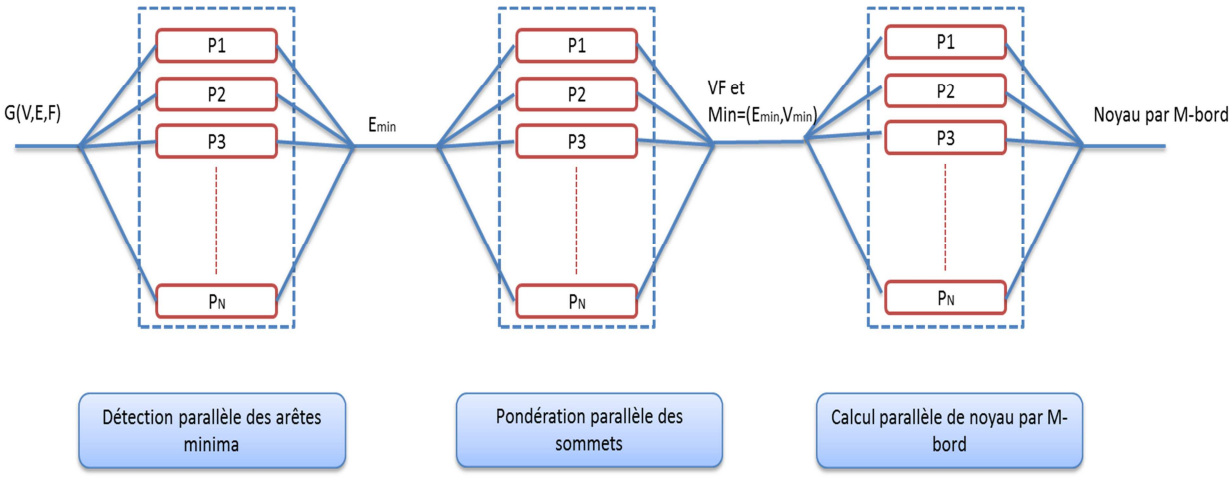


Figure 4. 3 Stratégie de parallélisation proposée de la technique de segmentation basée sur le calcul de noyau par M-bord

3.1 Parallélisation de l'algorithme de détection des minima régionaux

Pour la parallélisation de l'étape de détection des minima régionaux (Algorithme 18), nous proposons une stratégie basée sur le modèle *Diviser* et *Fusionner* (*Split and Merge* en Anglais). Ce choix est motivé par les parcours en profondeur d'abord du graphe qu'utilise l'algorithme de détection des minima. Ce type de parcours rend d'autres stratégies de parallélisation difficilement envisageables.

La parallélisation proposée consiste donc à :

(i) Décomposer le domaine de l'image en entrée (représentée sous-forme de GAV) en sous-domaines.

(ii) Appliquer parallèlement la version séquentielle de l'algorithme 18 à chaque sous-domaine.

(iii) Fusionner les résultats partiels pour aboutir au résultat global.

Parmi ces trois étapes, la fusion constitue l'étape la plus problématique vue la nature non locale du processus de détection des minima. Ainsi des configurations particulières peuvent se manifester après la détection des résultats partiels et doivent par conséquent être traitées adéquatement afin d'obtenir le résultat correct.

3.1.1 Phase de partitionnement

La division du graphe de départ G en des partitions ($partition_1$, …, $partition_P$) se fait selon le même algorithme utilisé dans la proposition de parallélisation de l'algorithme de calcul de noyau par M-bord basée sur la décomposition du domaine (Algorithme 12 détaillé dans la section 4 du chapitre 3). Rappelons que le partitionnement se fait d'une manière statique et que chaque partition représente un sous-graphe de G et est traitée par un processeur p avec $p=\{1,\ldots,P\}$. Chaque processeur calcule les indices des frontières de sa partition correspondante et dispose d'un pointeur lui permettant d'accéder à sa zone de travail. Il est à noter dans ce cas que tous les processeurs travaillent sur le même graphe et que des copies locales de ce dernier ne sont pas nécessaires vu qu'avec le partitionnement considéré les sous-graphes sont disjoints.

3.1.2 Application parallèle de l'algorithme de détection des minima régionaux aux partitions

Une fois le partitionnement effectué, l'algorithme séquentiel de calcul des arêtes minima est appliqué à chaque partition. Les différentes partitions sont traitées en parallèle. Ainsi, dans chaque partition, des minima régionaux locaux sont détectés et étiquetés. Cet étiquetage est effectué localement par chaque processeur en utilisant une plage d'étiquettes différentes de celles utilisées par les autres processeurs. Ceci permet d'avoir des minima régionaux ayant une étiquette unique au niveau global.

3.1.3 Phase de fusion

En traitant seulement une partie du graphe, un processeur ne peut disposer que d'un ensemble partiel des données du problème qu'il est en train de résoudre à savoir la détection des minima régionaux dans notre cas. De ce fait, les minima obtenus lors de l'étape précédente sont des minima locaux à chaque sous-graphe et ne peuvent pas être considérés comme des minima régionaux au niveau global du graphe. Ces derniers nécessitent une étape

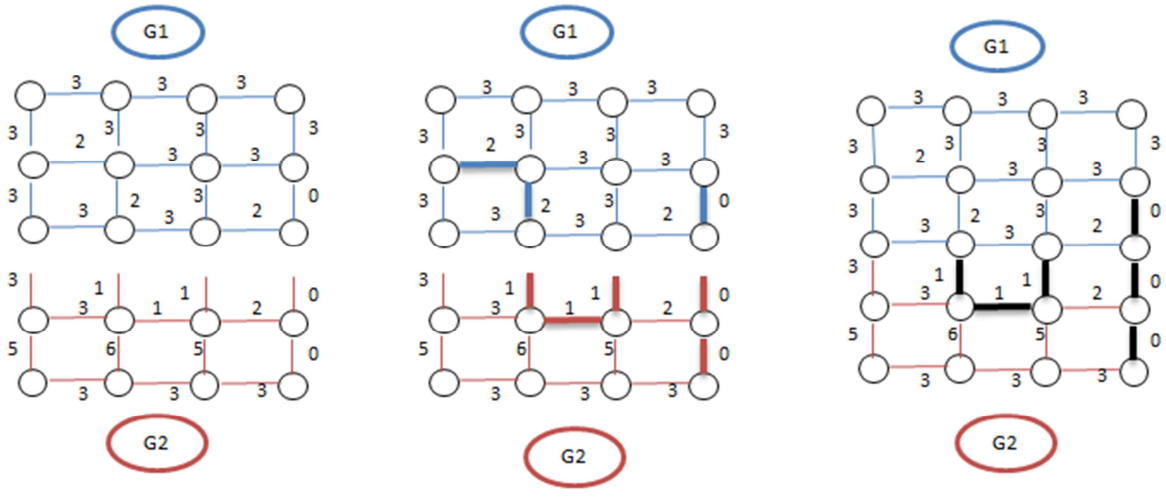
supplémentaire pour prendre la décision quant à leur nature finale lors de l'élaboration de la solution globale. Cette étape consiste à fusionner les résultats partiels en prenant en considération certaines configurations particulières succeptibles de se manifester. L'exemple suivant illustre ces configurations lors de la détection des minima régionaux sur un graphe traité par deux processeurs.

Minima régionaux locaux et problématique de fusion

Considérons le graphe *G* présenté dans la figure 4.2 (a). Soit *G1* et *G2* deux graphes tel que $G1 \cup G2 = G$ comme illustré dans la figure 4.4 (a), *G1* est représenté en bleu et *G2* en rouge. Le résultat de l'application de l'algorithme séquentiel de détection des arêtes minima (Algorithme 18) indépendamment sur chacun de ces deux graphes est représenté dans la figure 4.4 (b). Cette figure montre que:

Si nous considérons le cas de MR_{11}. Ce dernier vu localement au niveau de *G1* est un minimum régional. Cependant vu au niveau du graphe *G* dans sa globalité MR_{11} n'est plus un minimum régional car il possède des arêtes qui lui sont adjacentes, appartenant à *G2* et qui ont une altitude inférieure à la sienne. Pour que l'exécution de l'algorithme 18 sur les deux sous-graphes *G1* et *G2* produise le même résultat que celui obtenu directement sur le graphe *G* (figure 4.2 (b)) MR_{11} doit être supprimé.

Par ailleurs, nous constatons également que MR_{12} et MR_{22} représentent en réalité au niveau global le même minimum régional. Ainsi et pour aboutir au même résultat global donné par la figure 4.2 (b), ces deux minima doivent être fusionnés et une même étiquette doit leur être attribuée.



(a) (b) (c)

Figure 4. 4 Application de l'algorithme de détection des minima régionaux (Algorithme 18) sur un graphe (a) partitionné en deux sous-graphes (b) avant la phase de fusion et (c) après la phase

En partant de cet exemple, nous pouvons conclure que deux cas particuliers peuvent se manifester après l'obtention des minima régionaux au niveau de chaque sous-graphe:

- Le premier cas est celui d'un minimum régional *MR* qui s'étend sur plusieurs sous-domaines. Ce dernier se présente comme un ensemble de minima locaux adjacents appelés dans ce qui suit *sous-minima* régionaux et notés SMR_i tel que $MR = \bigcup_{i=\{1..p\}} SMR_i$. Chaque SMR_i est détecté localement par le $i^{ème}$ processeur. Ces SMR_i doivent être fusionnés pour aboutir au minimum régional *MR* recherché au niveau du graphe global.
- Le deuxième cas correspond à des minima régionaux particuliers détectés localement par un processeur et possédant au moins une arête voisine appartenant à un sous-domaine adjacent et ayant une altitude inférieure à celle de ce minima. Au niveau global, ces minima sont des faux minima régionaux *FMR* et doivent par conséquent être supprimés par la phase de fusion.

Tenant compte de ces deux cas, l'obtention de la solution globale nécessite l'exécution d'une phase de fusion qui s'effectue au niveau de chaque frontière entre deux sous-graphes voisins G_i et G_{i+1}. Ce processus est appelé *fusion élémentaire* et est noté Fe_i. Il consiste à parcourir les arêtes de la frontière en question et à :

- Supprimer les minima régionaux locaux d'un des sous-graphes qui sont voisins à une arête d'altitude inférieure de l'autre sous-graphe.
- Fusionner les sous-minima régionaux locaux qui possèdent la même altitude et qui appartiennent à des sous-graphes adjacents.

Le processus de fusion élémentaire doit être effectué au niveau de chaque frontière entre deux sous-graphes voisins. Ainsi, pour une répartition de la charge du travail sur *P* processeurs, *P*-1 fusions élémentaires Fe_i sont nécessaires. La fusion globale *Fg* peut être alors définie comme l'union de ces fusions élémentaires (4.1):

$$Fg = \bigcup_{i=\{1..p-1\}} Fe_i \qquad (4.1)$$

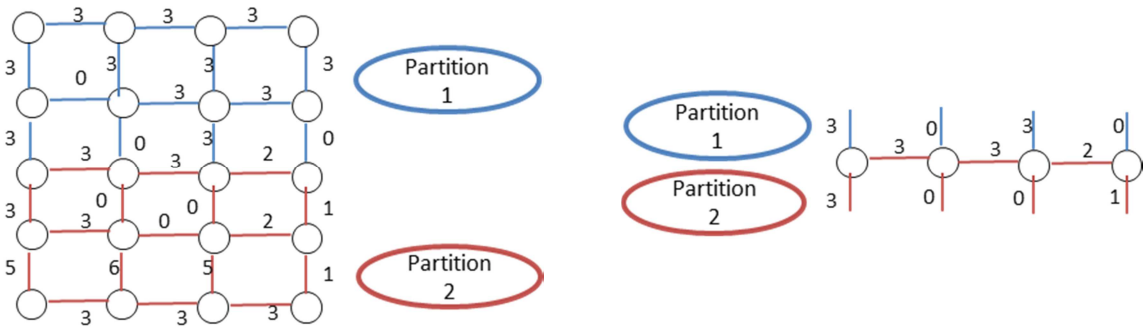
Notons que cette définition ne formalise pas l'ordre avec lequel s'exécutent les fusions élémentaires. Dans ce cadre, l'ordre le plus intuitif est séquentiel dans lequel les fusions élémentaires seront exécutés successivement en commençant par Fe_1 jusqu'à Fe_{p-1}. Pour des raisons d'optimisation, nous démontrons que le choix d'un autre ordre plus élaboré permettra

d'introduire un certain degré de parallélisation à l'exécution des fusions élémentaires. Le choix de l'ordre adopté sera motivé et détaillé dans le paragraphe suivant.

Optimisation de l'étape de fusion:

Un minimum régional local est étiqueté par le processeur qui le détecte. Nous rappelons que chaque processeur utilise une plage d'étiquettes disjointe de celles des autres processeurs. De cette manière, chaque étiquette attribuée localement à un minimum régional sera unique au niveau global. Notons par ailleurs qu'un plateau correspondant à un minimum régional global peut s'étendre spatialement sur plus qu'une partition (deux ou plus). Dans ce cas, il sera étiqueté par l'algorithme parallèle par plusieurs étiquettes qui correspondent aux étiquettes des minima régionaux locaux qui le composent. Dans ce cas de figure, les plateaux des minima régionaux locaux doivent être fusionnés et étiquetés par la même étiquette. Dans notre proposition nous attribuons la plus petite étiquette au minimum global fusionné. D'autre part, un plateau situé dans une zone frontière (illustrée dans la figure 4.5) peut être étiqueté par un processeur comme un minimum local. Cependant, si ce plateau est adjacent à une arête ou plateau de valeur inférieure dans une partition voisine alors il doit être supprimé. La suppression se fait en lui attribuant l'étiquette des arêtes non minima.

En conclusion, les deux opérations sur lesquelles est basée la fusion élémentaire à savoir la suppression et la fusion des minima régionaux locaux sont mises en œuvre en pratique par un changement d'étiquettes. Pour des raisons d'optimisation, et puisque un minimum régional peut subir plusieurs changements d'état et par conséquent avoir plusieurs étiquettes durant le déroulement de toutes les fusions élémentaires, alors nous avons opté pour une application différée de ces changements sur le graphe. Ces changements sont plutôt effectués dans une table de correspondance qui donne pour chaque minimum régional la correspondance entre son étiquette initiale et l'étiquette qui lui est attribuée suite à une opération élémentaire de fusion. Une fois les fusions élémentaires au niveau de toutes les frontières sont terminées, l'application des étiquettes finales se fait d'une manière unique et globale sur tout le graphe.



(a) (b)

Figure 4.5 Deux partitions adjacentes (a) et la frontière entre elles (b)

Comme nous l'avons décrit dans le paragraph précédent, l'étape de fusion des résultats partiels fournis par les différents processeurs s'effectue à travers le déroulement de toutes les fusions élémentaires qui ont lieu au niveau des frontières entre les partitions. Trois stratégies sont possibles pour l'exécution de ces fusions élémentaires:

- La première stratégie est séquentielle. Dans ce cas, les fusions élémentaires s'exécutent successivement l'une après l'autre en commençant par la première frontière jusqu'à la dernière frontière. Dans un contexte d'optimisation, cette stratégie s'avère inefficace [61, 62] car elle est coûteuse en terme de temps.
- La deuxième stratégie est parallèle mais fausse de point de vue concurrence [61]. En effet, un minimum local d'une $partition_i$ dont le plateau associé est adjacent en même temps à la $partition_{i-1}$ et la $partition_{i+1}$ risque que son étiquette soit modifiée différemment par deux fusions élémentaires en même temps, ce qui peut entraîner un faux résultat. Une solution pour surmonter cette problématique consiste à utiliser une zone critique pour sérialiser les mises à jour, ce qui entraîne de nombreuses synchronisations conduisant à détériorer les performances du parallélisme.
- La troisième stratégie est pyramidale: dans ce cas, des fusions élémentaires au niveau de certaines frontières sont exécutées en parallèle. La détermination des frontières concernées par le parallélisme est guidée par un ordre pyramidale comme indiqué dans la figure 4.6. Cet ordre garantit qu'une partition ne peut participer à un instant donné qu'à une seule fusion élémentaire.

En adoptant la stratégie pyramidale, et si le nombre de partitions est un multiple de 2 (i. e., $P = 2^q$), alors le nombre d'itérations nécessaire pour exécuter toutes les fusions élémentaires est donné par (4.2) [61]:

$$q = log_2(P) \qquad (4.2)$$

Le degré moyen de parallélisme (ADP : Average Degree of Parallelism) qui est la division du nombre de frontières à fusionner par le nombre d'itérations est donné par (4.3) [61]:

$$ADP = (2^q - 1)/q \qquad (4.3)$$

La figure 4.6 illustre l'approche de fusion pyramidale. Dans le cas où P = 16 partitions, il y a $\log_2(16) = 4$ itérations. Le degrè moyen de parallélisme est donc égal à 15/4 = 3,75.

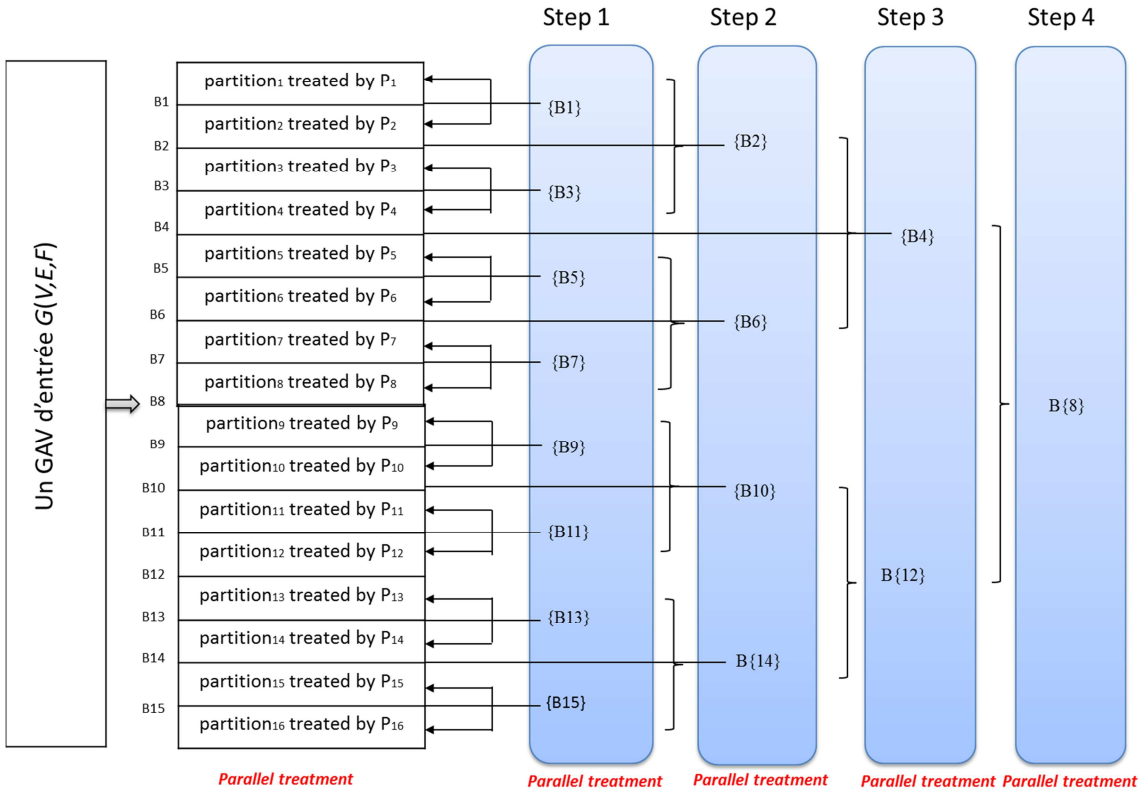


Figure 4. 6 Illustration de l'approche de fusion pyramidale au niveau des frontières des partitions

En tenant compte des stratégies que nous avons adoptées concernant la mise à jour des étiquettes à travers une table de correspondance et la parallélisation pyramidale des fusions élémentaires au niveau des frontières, le processus de fusion globale que nous proposons est donné par l'agorithme 21. Dans cet algorithme, les frontières concernées par les fusions élémentaires parallèles entre P_i et P_{i+1} sont déterminées à chaque itération par leur indice i qui doit vérifier la condition suivante:

$$(P_i\text{-}Start) \,\%\, Step = 0 \qquad (4.4)$$

Où Start désigne l'indice de début des frontières concernées par l'exécution parallèle et *Step* le pas qui sépare les indices des frontières concernées.

Ces deux variables sont initialisées à *Start*=0 et *Step*=2. Puis, à chaque itération, elles sont mises à jour comme suit:

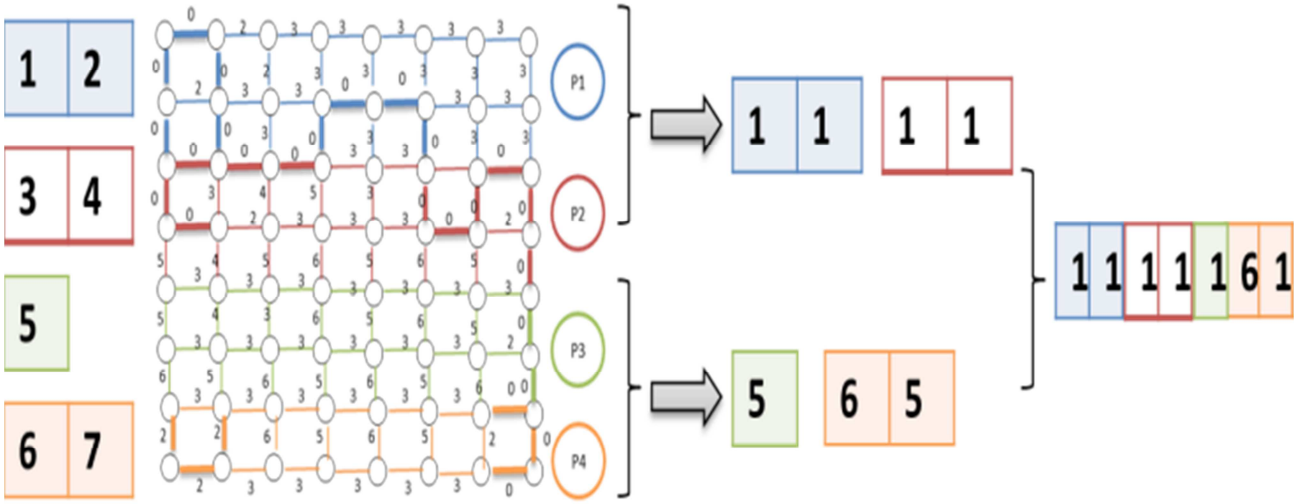
$$Start = Step\text{-}1 \qquad (4.5)$$

$$Step = 2 \times Step \qquad (4.6)$$

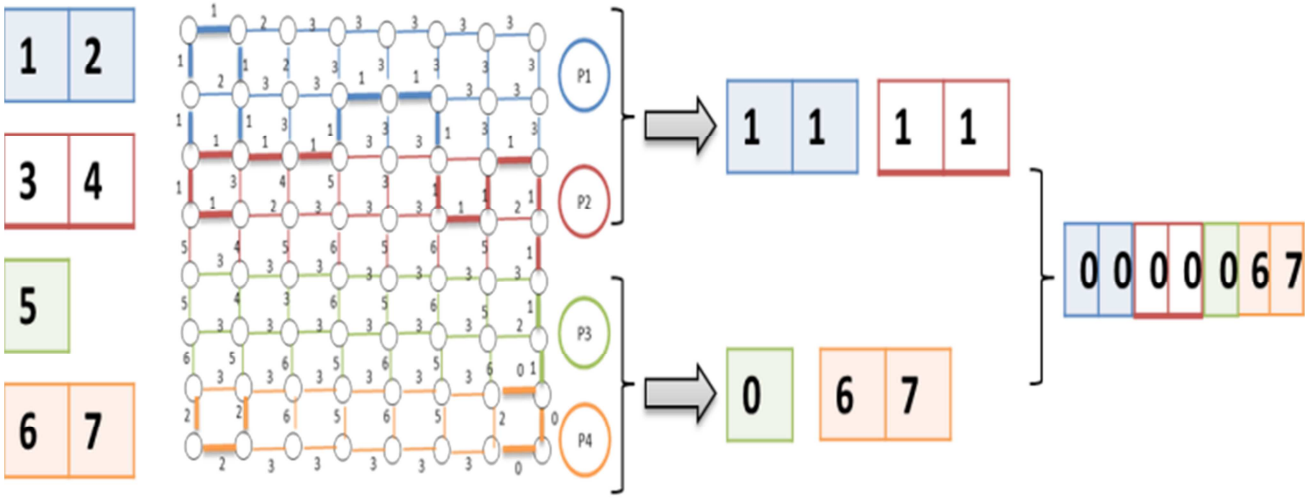
Pour une frontière donnée entre une partition associée à un processeur P_i et une partition associé au processeur P_{i+1}, la fusion élémentaire Fe_i est effectuée comme suit : les arêtes *u* associées à P_i sont comparées à leurs arêtes voisines *v* associées à P_{i+1}. Généralement ces arêtes sont en nombre de trois : deux arêtes horizontales et une arête verticale sauf pour la première et la dernière arête de la frontière qui n'ont que deux arêtes voisines (*cf.* figure 4.5). Pour minimiser le coût du balayage, l'altitude de *u* n'est comparée qu'à l'altitude minimale parmi celles des voisins considérés. Prenons comme exemple la frontière présentée dans la figure 4.4 (b), l'arête d'altitude 0 appartenant à la partition 1 possède trois arêtes voisines appartenant à la partition 2 (deux arêtes horizontales d'altitude 3 et une arête verticale d'altitude 0). Dans la phase de fusion l'altitude de cette arête n'est comparée qu'à l'altitude minimale des arêtes voisines qui est l'arête d'altitude 0. A chaque fois qu'une opération de fusion ou de suppression est nécessaire alors cette dernière est effectuée par modification des étiquettes dans la table de correspondance (Tableau *TabC* dans l'algorithme 21). Il est à noter à ce niveau que le choix que nous avons adopté concernant l'utilisation des plages d'étiquettes différentes par les différents processeurs lors de l'étape d'exécution parallèle de l'algorithme 18 rend l'utilisation d'une table de correspondance globale possible même dans un contexte d'exécution parallèle. En effet, chaque processeur va écrire dans une zone de cette table qui correspond à la plage d'étiquettes qui lui est associée. Les accès concurrentiels à une même cellule sont ainsi évités. La figure 4.7 illustre l'utilisation de la table de correspondance pour mettre en œuvre les opérations de fusion et de suppression par mise à jour des étiquettes. Dans ce cadre, deux configurations peuvent se produire:

- La première configuration est illustrée par la figure 4.7 (a). Dans ce cas, à la première itération, le minimum étiqueté 1 par P_1 est équivalent au minimum étiqueté 3 par P_2. De plus, le minimum étiqueté 3 par P_2 est équivalent au minimum étiqueté 2 par P_1 qui est à son tour équivalent au minimum étiqueté 4 par P_2. Ainsi, les minima étiquetés 1, 2, 3 et 4 sont équivalents. En parallèle, le minimum étiqueté 5 par P_3 et le minimum étiqueté 7 par P_4 sont équivalents. À la deuxième itération, l'équivalence entre les minima étiquetés 1 et 5 donne la table de correspondance globale finale.

- La deuxième configuration est illustrée par la figure 4.7 (b). Dans ce cas, à la première itération, l'équivalence entre les minima étiquetés 1 et 2 par P_1 et les minima étiquetés 3 et 4 par P_2 est notée. En parallèle, le minimum étiqueté 7 par P_4 a conduit à la suppression du minimum étiqueté 5 par P_3. A la deuxième itération, ce dernier minimum est équivalent au minima étiqueté 1. Ceci entraîne la suppression de ces minima. Finalement, la table de correspondance globale est obtenue.



(a)



(b)

Figure 4. 7 L'utilisation de la table de correspondance pour sauvegarder la mise à jour des labels lors de la phase de fusion (a) cas d'équivalence entre les minima locaux et (b) cas de suppression de faux minima locaux

Algorithme 21 : Fusion_Pyramidale

```
Data : une fonction F et ses minima régionaux locaux Min
Data : (P_1, ..., P_N): ensemble de partitions et N le nombre de processeurs.
Result : M minima régionaux globaux de F fusionnés.
// Initialisation
Nb_bords := N − 1;
Start:=0;Step:=2;
while (Start < Nb_bords) do
    //Faire en parallèle
    foreach (Partition P_i tel que i ∈ (1, ... , N)) do in parallel
        if ( ( (P_i - Start) %Step ) = 0 ) then
            True:=1;
        IsActive:=( True & (p < (N-1)) ) ;
        if (IsActive) then
            //Parcourir la frontière sud de la partition nommée BP_i
            foreach ( u ∈ BP_i ) do
                //Calculer l'arête voisine à u ayant l'altitude minimale
                v:= MinVoisin (u);
                if ( F(u) == F(v) ) then
                    if (u ∈ Min & v ∈ Min) then
                        //Fusion des plateau minima de u et v en leur attribuant la même étiquette
                    else if (u ∈ Min & v ∉ Min) then
                        //Suppression du plateau de u de l'ensemble des minima
                    else if (u ∉ Min & v ∈ Min) then
                        //Suppression du plateau de v de l'ensemble des minima
                if ( F(u) < (v) ) then
                    if (v ∈ Min) then
                        //Suppression du plateau de v de l'ensemble des minima
                if ( F(u) > F(v) ) then
                    if (u ∈ Min) then
                        //Suppression du plateau de u de l'ensemble des minima
    //Mise à jour final de la table de correspondance pour obtenir les étiquettes globales
    foreach ( i ∈ (1, ... , Nb_Labels ) do
        h := i;
        while ( TabC[h]! = h ) do
            h := TabC[h];
        TabC[i] := h;
    Start:=Step-1;
    Step:=2*Step;
```

3.1.4 Algorithme parallèle de détection des minima régionaux

En tenant compte des trois étapes décrites dans les paragraphes précédents, la détection parallèle des minima régionaux peut être décrite par l'algorithme 20. Cet algorithme détaille principalement l'étape de l'exécution parallèle de la détection des minima régionaux dans chaque partition du graphe. Pour l'étape de la répartition de la charge, l'algorithme 20 fait appel à l'algorithme de partitionnement statique qui assure la division du graphe initiale en des

partitions non chevauchées (Algorithme 13 détaillé dans la section 4.1 du chapitre 3). Pour l'étape de fusion, il fait appel à l'algorithme de fusion pyramidale (Algorithme 21) détaillé dans la section 3.1.3.

Algorithme 20 : Algorithme parallèle de détection des arêtes minima

```
   Data : (V, E, F) un graphe à arêtes valuées et P le nombre de
          processeurs.
   Result : E_min: l'ensemble des minima de la fonction d'entrée F.
 1 // Initialisation
 2 foreach ( processeur p ∈ (1, ... , P) ) do in parallel
 3 |  L_p := φ;
 4 E_min := φ;
 5 (partition_1, ... ,partition_P):=Partitionnement_statique((V, E, F), P) ;
 6 foreach ( processeur p ∈ (1, ... , P) ) do in parallel
 7 |  foreach ( u ∈ partition_p ) do
 8 |  |  if (u non encore traitée) then
 9 |  |  |  E_min := E_min ∪ {u};
10 |  |  |  L_p := L_p ∪ {u};
11 |  |  |  //Parcourir le plateau auquel appartient u
12 |  |  |  while ( L_p ≠ φ ) do
13 |  |  |  |  foreach ( w ∈ L_p ) do
14 |  |  |  |  |  L_p := L_p \ {w};
15 |  |  |  |  |  foreach ( v ∈ E voisin de w) do
16 |  |  |  |  |  |  if (( w ∈ E_min) & (F[v] < F[w]) ) then
17 |  |  |  |  |  |  |  //w est traitée mais non étiquetée comme arête
   |  |  |  |  |  |  |   minima
18 |  |  |  |  |  |  |  E_min := E_min \ {w};
19 |  |  |  |  |  |  |  L_p := L_p ∪ {w};
20 |  |  |  |  |  |  else if ( F[v] = F[w] ) then
21 |  |  |  |  |  |  |  if ( ( w ∈ E_min & v non encore traitée ) ∥ ((w
   |  |  |  |  |  |  |   est traitée et ∉ E_min) & (v non traitée ou
   |  |  |  |  |  |  |   ∉ E_min) ) ) then
22 |  |  |  |  |  |  |  |  E_min := E_min ∪ {v};
23 |  |  |  |  |  |  |  |  L_p := L_p ∪ {v};
24 E_min=Fusion_Pyramidale(F, (E_min(1),... , E_min(P)) ,(partition_1,
   ... ,partition_P), P);
```

L'application de la mise à jour des étiquettes consiste à appliquer la table de correspondance globale sur le Graphe à Arêtes Valuées. La figure 4.8 illustre cette phase dans les deux configurations possibles: équivalence entre les minima locaux (figure 4.8 (a)) et suppression des faux minima (figure 4.8 (b)). Notons que cette phase (application de la mise à jour des étiquettes sur le graphe) ne présente aucune dépendance de données. Ainsi, le parallélisme massif est appliqué pour sa mise en œuvre.

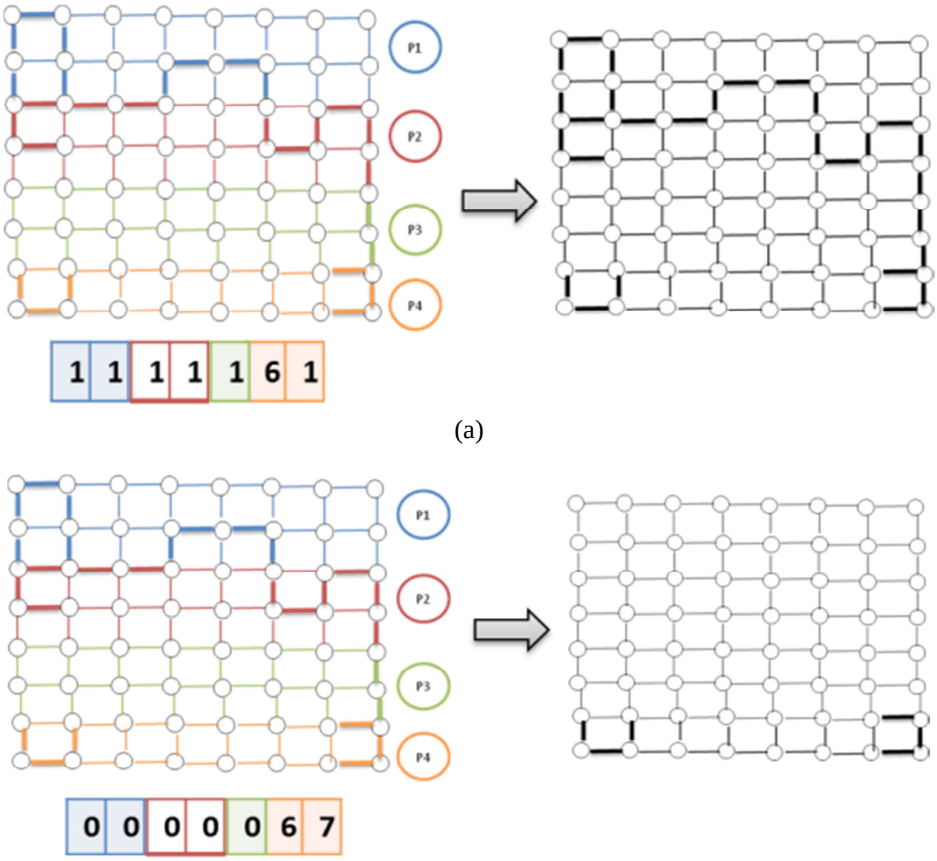


Figure 4. 8 Application de la mise à jour des étiquettes sur le graphe (a) cas d'équivalence entre minima et (b) cas de suppression de faux minima

3.2 Parallélisation de l'algorithme de pondération des sommets et calcul de l'ensemble des sommets minima

Pour l'étape de pondération des sommets, nous notons que d'après le principe détaillé par l'algorithme 19, chaque sommet peut être pondéré indépendamment des autres sommets du graphe. De ce fait, aucun problème de dépendance de données ne se présente par rapport à la parallélisation et par conséquent le parallélisme massif peut être appliqué pour son optimisation.

L'algorithme parallèle de pondération des sommets (Algorithme 22) consiste donc à traiter les différents sommets du graphe en parallèle en utilisant *P* processeurs.

Algorithme 22 : Pondération parallèle des sommets et calcul parallèle de l'ensemble de sommets minima à partir de l'ensemble des arêtes minima

Data : $((V, E, F)$ graphe à arêtes valuée, E_{min}: l'ensemble des arêtes minima et P le nombre de processeurs.
Result : VF altitude des sommets du graphe de départ et V_{min} l'ensemble des sommets minima.

```
1  // Initialisation
2  V_min := φ;
3  foreach ( processeur p ∈ (1, ... , P) ) do in parallel
4      // parallélisme massif
5      foreach ( v ∈ V ) do
6          VF[v]:=255;
7          foreach ( w ∈ V voisin de v) do
8              u:={v, w} ;
9              if (F[u] < VF[v]) then
10                 VF[v] = F[u];
11                 if (u ∈ E_min) then
12                     V_min= V_min ∪ {v};
```

4. Résultats et discussion

Dans cette section, nous présentons et évaluons les résultats obtenus en implémentant la technique parallèle de segmentation basée sur le calcul de noyau par M-bord sur l'architecture multi-cœurs détaillée dans la section 8.1 du chapitre 3 et en utilisant la même base d'images décrite dans la section 8.2 du même chapitre.

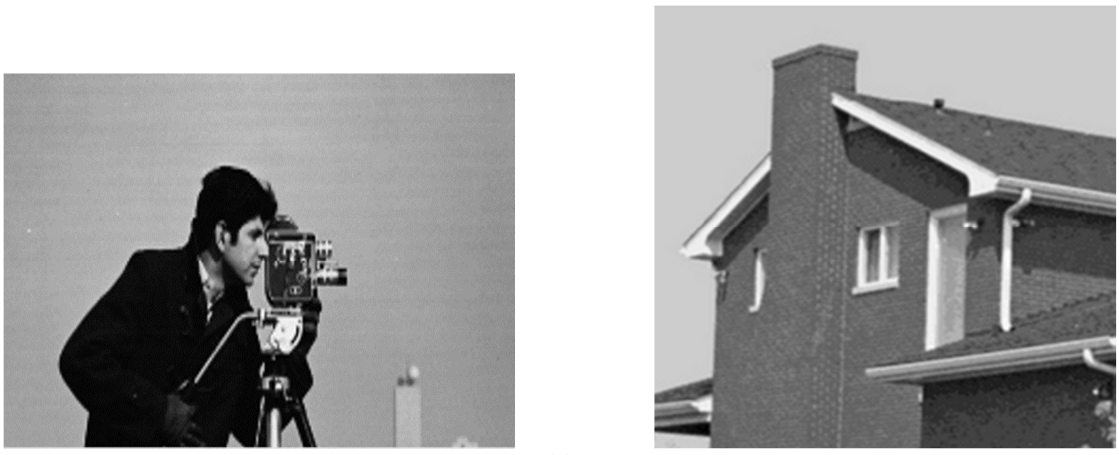
Rappelons que cette technique est constituée de trois principales étapes: la détection des minima, la pondération des sommets (et calcul de l'ensemble de sommets minima) et enfin le calcul de noyau par M-bord. De plus, la parallélisation de cette technique consiste à paralléliser chacune de ses étapes. Ainsi, nous évaluons, dans un premier lieu, la parallélisation de chaque étape et, dans un second lieu, la totalité de la technique de segmentation. Notons que les algorithmes parallèles de calcul de noyau par M-bord ont été évalués dans le chapitre 3. En conséquence, nous nous limitons à évaluer les deux premières étapes de la technique (l'étape de détection des minima et l'étape de pondération de sommets). En outre, l'évaluation de la totalité de la technique de segmentation se fait en utilisant l'algorithme parallèle de calcul de noyau par M-bord le plus performant en termes de temps d'exécution et qui a donné le meilleur facteur d'accélération qui est l'algorithme 16.

Initialement, nous présentons des illustrations des images segmentées en appliquant la technique parallèle proposée sur des images en niveaux de gris afin d'évaluer la qualité de

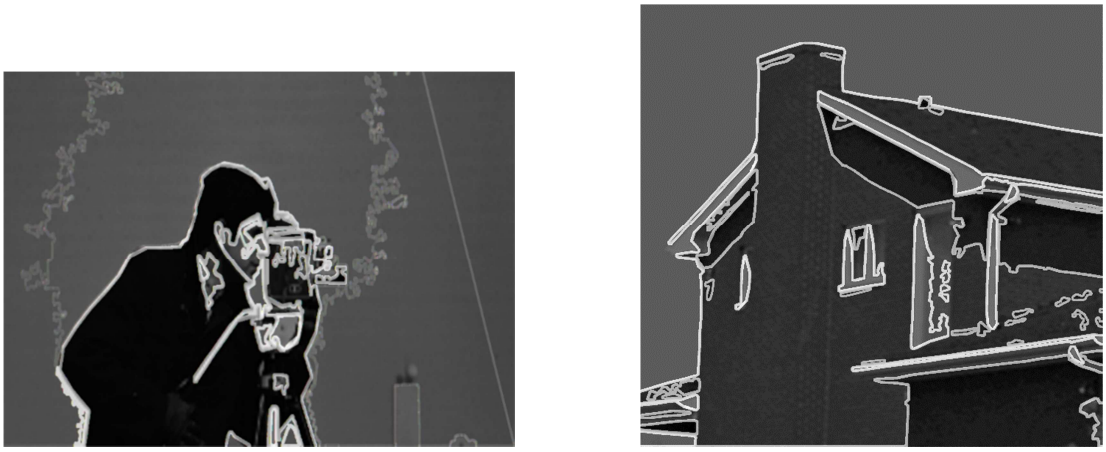
segmentation obtenue. Ensuite, nous discutons l'évaluation de cette technique en termes de temps d'exécution et d'accélération par rapport à la variation du nombre de cœurs ainsi que par rapport à la résolution des images segmentées. Pour ce faire, la même démarche suivie pour évaluer l'algorithme de calcul de noyau par M-bord sera suivie pour évaluer, dans un premier lieu, chaque étape de cette technique, et dans un second lieu, sa totalité.

4.1 Illustration des résultats et évaluation de la qualité de segmentation obtenue

La figure 4.9 présente une illustration des résultats obtenus en appliquant la technique de segmentation basée sur le calcul de noyau par M-bord sur des images de tailles différentes. Pour réduire la sur-segmentation, les images sont filtrées par un filtre anisotropique [101, 104] avant d'appliquer la technique de segmentation. Ce filtre permet de limiter le nombre de minima régionaux grâce au lissage des régions qu'il assure tout en préservant les structures et les contours. Après avoir segmenté l'image filtrée, les résultats obtenus sont superposés aux images initiales.



(a)



(b)

Figure 4. 9 (a) images initiales et (b) les images segmentées obtenues en appliquant la technique de segmentation basée sur le calcul de noyau par M-bord sur les images de (a)

En plus de l'optimisation du temps de calcul, la technique de segmentation parallèle proposée doit préserver la même qualité de segmentation que celle produite par la version séquentielle. Afin de valider cet aspect, nous avons procédé à la mesure de la similarité entre les deux résultats de segmentation obtenus respectivement par la version séquentielle et parallèle. Les critères que nous avons utilisés à cet égard sont l'indice de Dice [111] et l'indice de Jaccard [112] donnés respectivement par 4.7 et 4.8.

$$Dice = 2 \times \frac{|A \cap B|}{|A| + |B|} \qquad (4.7)$$

$$Jaccard = \frac{|A \cap B|}{|A \cup B|} \qquad (4.8)$$

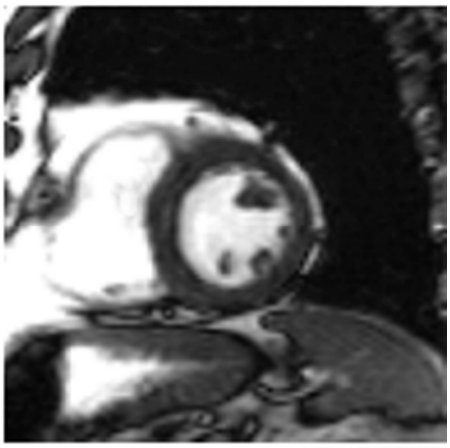
tel que: A désigne l'image résultat de segmentation séquentielle et B désigne l'image résultat de segmentation parallèle.

Le tableau 4.1 présente les mesures obtenues. Ces derniers montrent une similarité d'environ 95% à 98%.

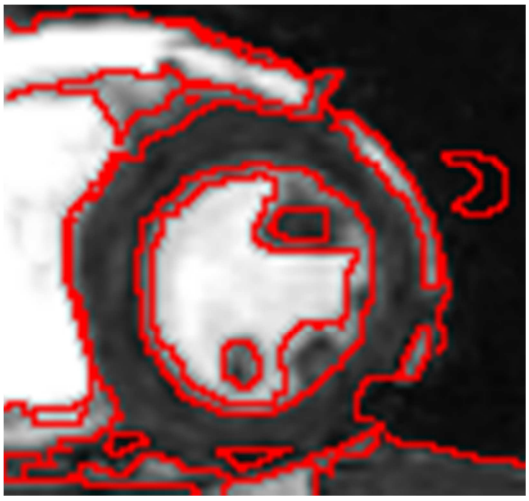
Nombre d'images	10	10	10	10	10	10
Résolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Indice de Dice	0,963	0,959	0,949	0,950	0,963	0,969
Indice de Jaccard	0,981	0,979	0,968	0,974	0,975	0,978

Tableau 4. 1 Evaluation de la qualité de segmentation parallèle basée sur l'algorithme 16

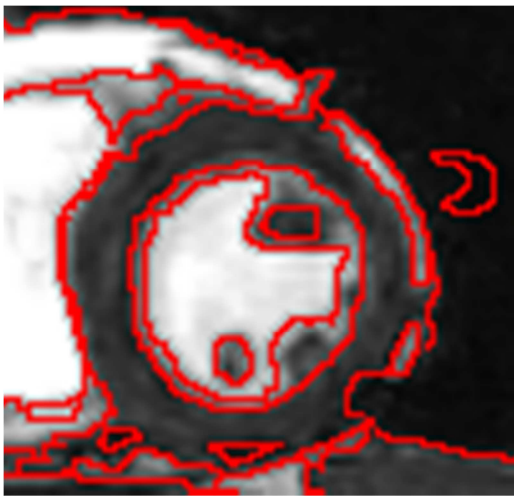
En plus de la validation de la stratégie de parallélisation sur une base d'images représentant des scènes variées, nous avons également testé la technique parallèle sur une image 2D d'une IRM cardiaque. La figure 4.10 donne une illustration des résultats obtenus. Il est à noter dans ce cadre que tout comme pour la base d'images variées, cette image médicale a été filtrée auparavant afin de réduire les minima régionaux et éviter ainsi le problème de sur-segmentation. Les figures 4.10 (b) et 4.10 (c) montrent les contours obtenus au niveau de la région du myocarde ventriculaire gauche respectivement par la technique séquentielle et parallèle et donnent une idée sur la similarité entre les deux résultats.



(a)



(b)



(c)

Figure 4. 10 Segmentation d'une image (a) 2D d'IRM cardiaque (b) application de la technique séquentielle de segmentation par LPE d'arêtes basée sur le calcul de noyau par M-bord et (c) application de la technique parallèle de segmentation basée sur l'algorithme 16

4.2 Evaluation de la technique parallèle de segmentation basée sur le calcul du noyau par M-bord

L'évaluation de la technique parallèle de segmentation est effectuée en suivant la même démarche suivie pour évaluer les algorithmes parallèles de calcul du noyau par M-bord : initialement l'évaluation se fait en termes de nombre de cœurs et ensuite en termes de tailles d'images.

4.2.1 Evaluation en termes de nombre de cœurs

Initialement chaque étape de la technique de segmentation est évaluée à part, ensuite, la totalité de cette technique est évaluée.

La figure 4.11 (respectivement figure 4.12) représente le temps d'exécution et l'accélération de l'algorithme parallèle de détection des minima (respectivement l'algorithme parallèle de pondération des sommets) en fonction du nombre de cœurs pour des images de taille 2048×2048. Ces figures montrent que l'accélération de ces deux algorithmes augmente avec le nombre de cœurs pour atteindre 1.6 pour l'algorithme 20 et 5.87 pour l'algorithme 22 en utilisant 8 cœurs.

En conséquence, la figure 4.13 montre que l'accélération de la technique parallèle de segmentation par coupure par LPE augmente avec le nombre de cœurs utilisés pour atteindre 2.1 en utilisant 8 cœurs pour des images de taille 2048×2048.

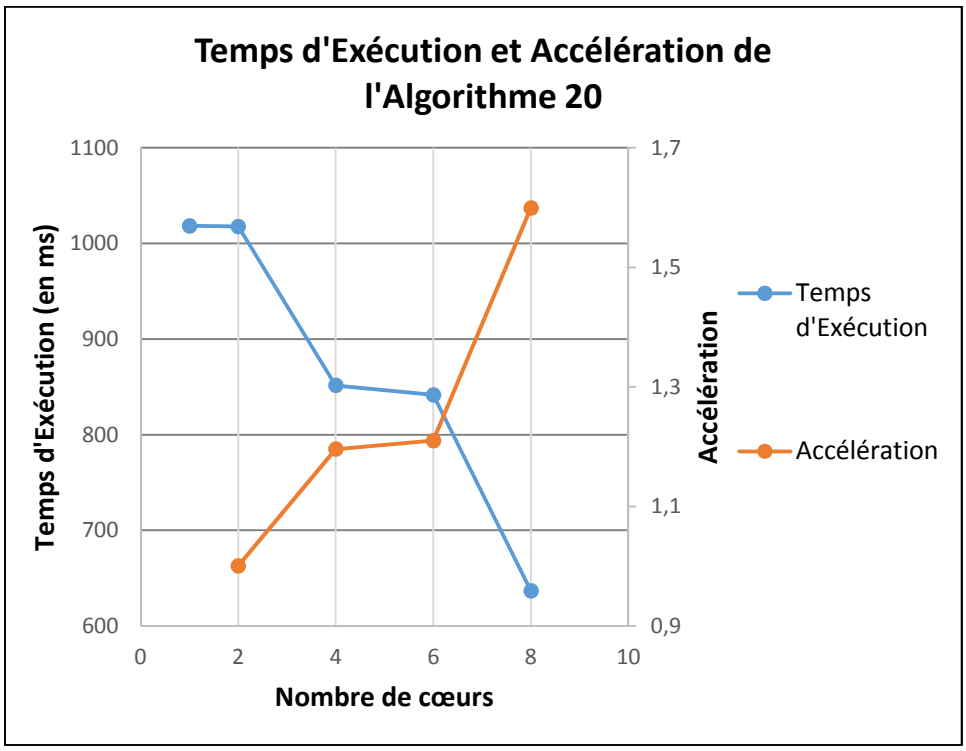


Figure 4. 11 Temps d'exécution et accélération de l'algorithme parallèle de détection des minima régionaux (Algorithme 20) en fonction du nombre de cœurs

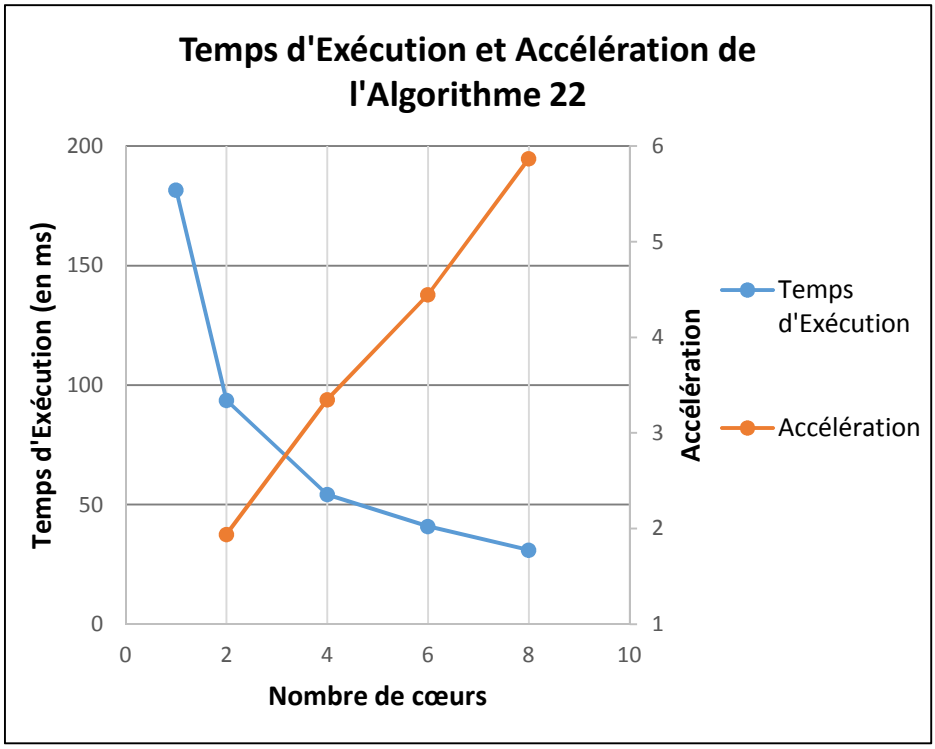


Figure 4. 12 Temps d'exécution et accélération de l'algorithme parallèle de pondération des sommets du graphe (Algorithme 22) en fonction du nombre de cœurs

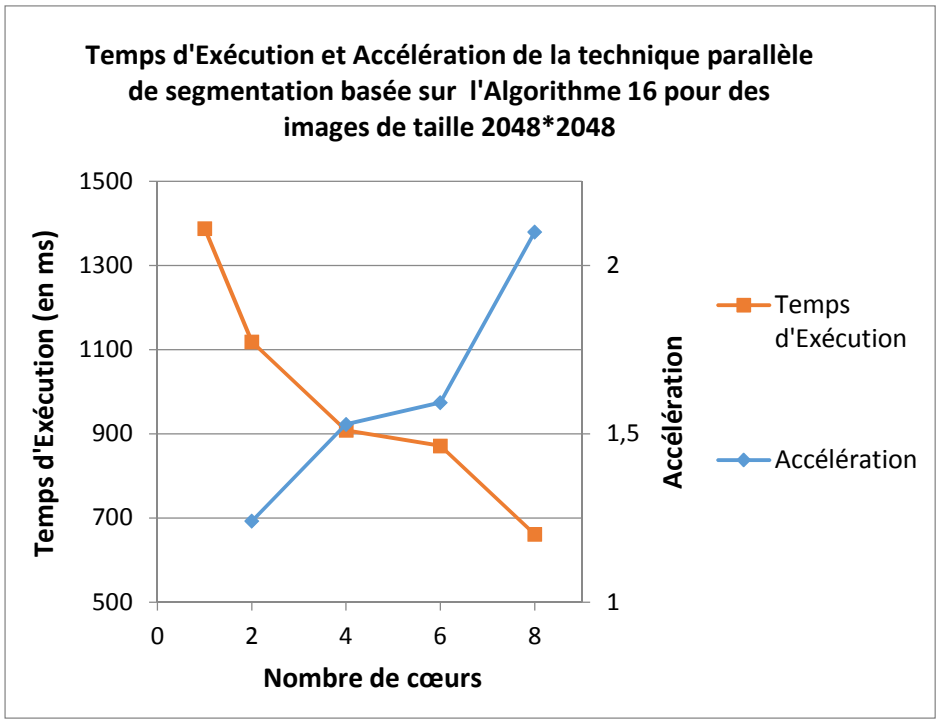


Figure 4. 13 Temps d'exécution et accélération de la totalité de la technique de segmentation basée sur l'algorithme 16 en fonction du nombre de cœurs

4.2.2 Evaluation en termes de résolution des images

Les figures 4.14 et 4.15 représentent l'évolution de l'accélération des algorithmes 20 et 22 en fonction du nombre de cœurs pour des images de différentes tailles. Ces figures montrent

que le gain en termes de temps pour ces deux algorithmes est garantie quel que soit la résolution des images en entrée.

Dans la figure 4.14, nous remarquons que l'augmentation de l'accélération de l'algorithme 20 quand le nombre de cœurs passe de 4 à 6 est inférieure à celle quand le nombre de cœurs passe de 6 à 8. Ceci est dû à la phase de fusion. En effet, pour P = 4 cœurs, et selon l'équation 4.1, seulement 2 itérations sont nécessaires pour effectuer la fusion pyramidale des minima régionaux locaux. Pour P = 6 cœurs, 3 itérations sont nécessaires pour obtenir le résultat global. Ainsi, lorsque le nombre de cœurs augmente de 4 à 6, le temps d'exécution de l'étape de détection des minima régionaux dans les partitions diminue mais le coût de fusion augmente (le nombre d'itérations passe de 2 à 3). Cependant, lorsque le nombre de cœurs passe de 6 à 8, le nombre d'itérations de fusion requises reste le même (3 itérations). Ainsi, dans ce cas, le temps d'exécution de l'étape de détection des minima régionaux dans chaque partition diminue sans augmentation du coût de la phase de fusion. Par conséquent, le gain de l'étape de détection des minima régionaux en termes de temps est plus important de 6 à 8 cœurs par rapport à celui de 4 à 6 cœurs.

En contrepartie, la figure 4.14 montre que l'accélération de l'algorithme 22 augmente avec le nombre de cœurs utilisés.

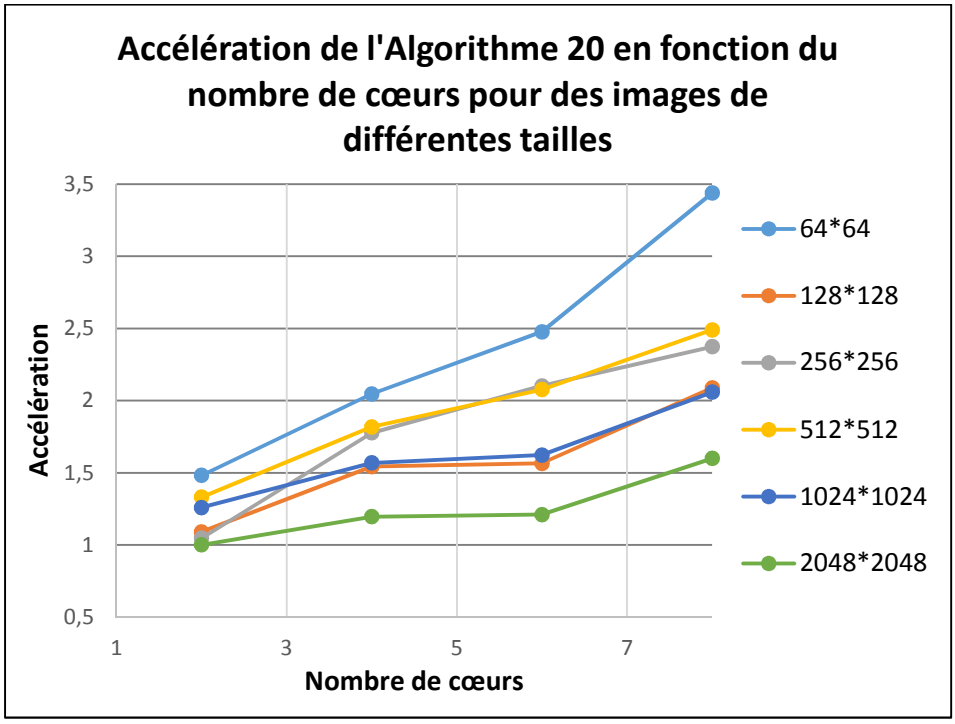


Figure 4. 14 Accélération de l'algorithme 20 pour des images de différentes tailles en fonction du nombre de cœurs

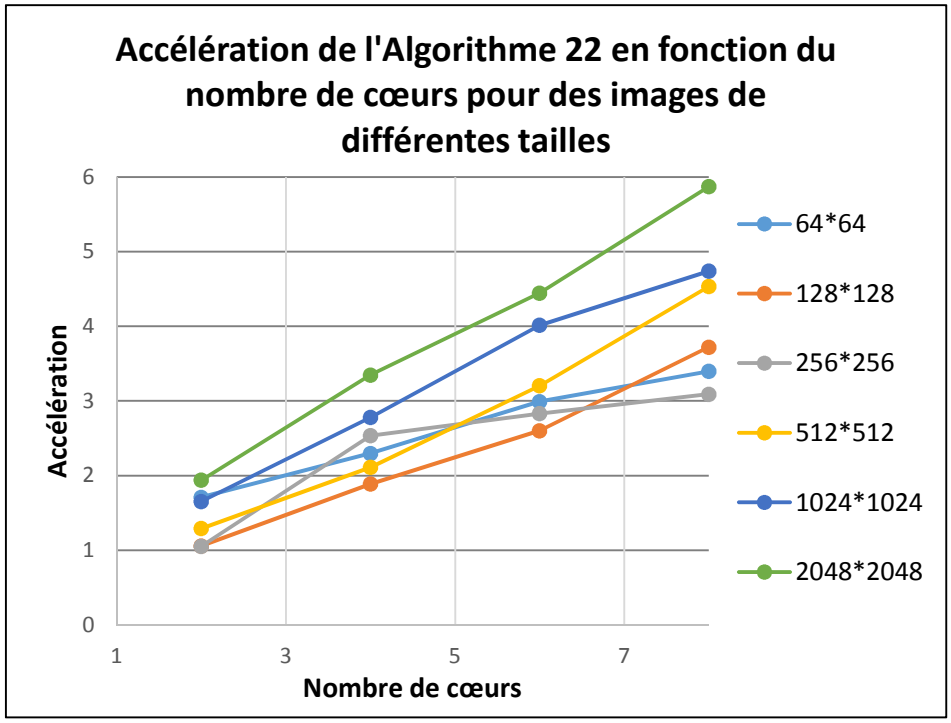
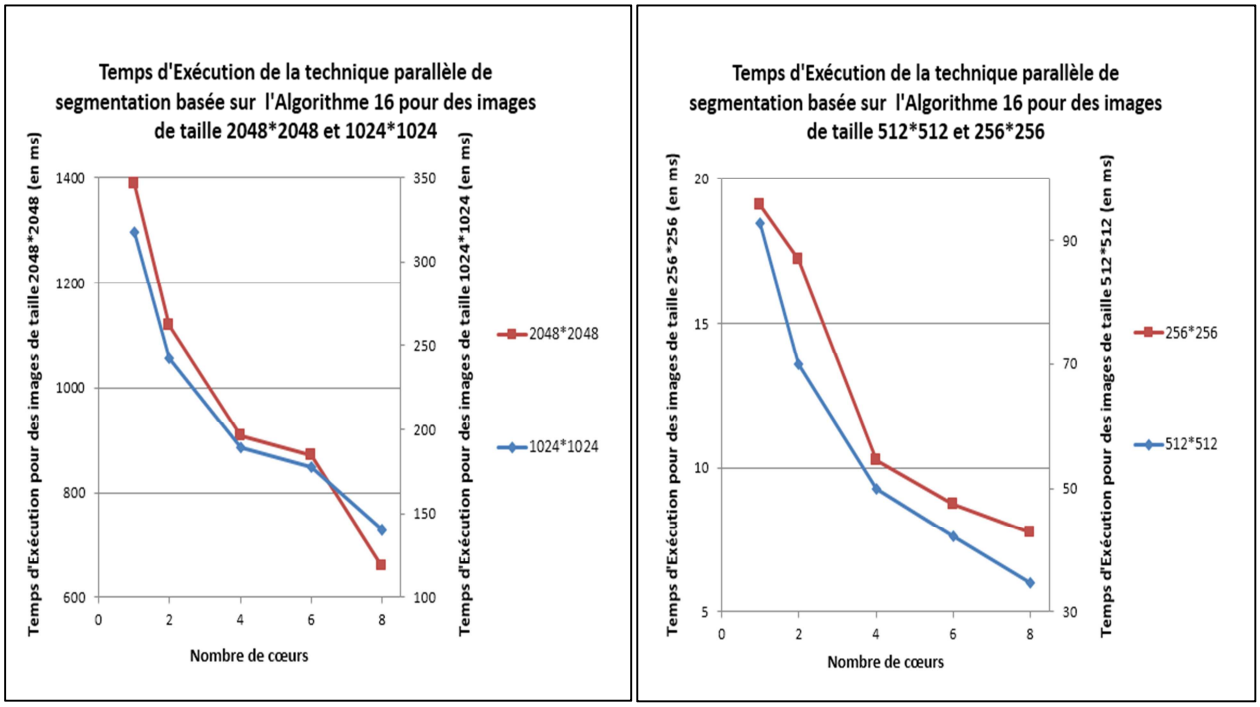


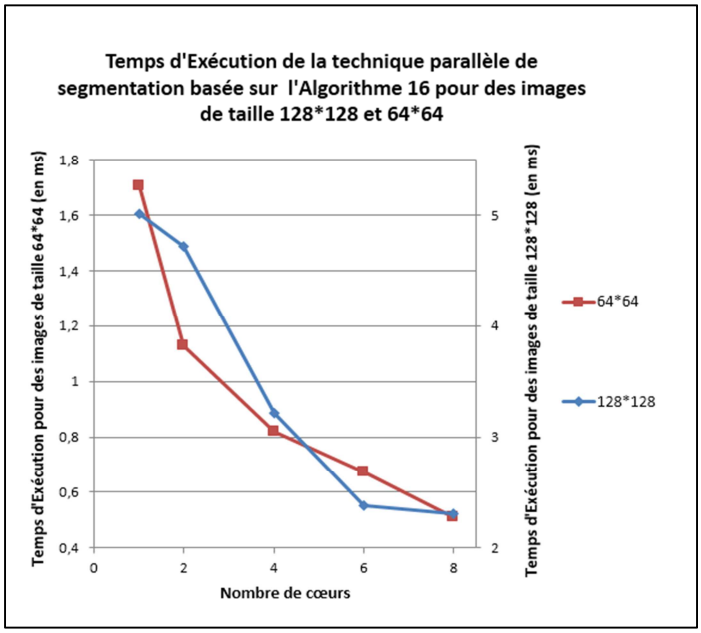
Figure 4. 15 Accélération de l'algorithme 22 pour des images de différentes tailles en fonction du nombre de cœurs

Le temps d'exécution (respectivement l'accélération) de la totalité de la technique de segmentation est présenté dans la figure 4.16 (respectivement 4.17) pour des images de différentes tailles. Ces figures montrent que l'accélération de la technique de segmentation proposée a conduit à un facteur d'accélération qui est garanti pour toute résolution d'images. Ce facteur est de 3.34 pour des images de taille 64×64, de 2.17 pour des images de taille 128×128, de 2.46 pour des images de taille 256×256, de 2.67 pour des images de taille 512×512, de 2.25 pour des images de taille 1024×1024 et 2.09 pour des images de taille 2048×2048 comme il est noté dans le tableau 4.2.

D'autre part, le palier noté dans ces figures quand le nombre de cœurs passe de 4 à 6 est dû à l'étape de calcul des minima régionaux (Algorithme 20). En effet, nous avons montré précédemment que le temps d'exécution de l'algorithme de calcul de noyau par bord et de l'algorithme de pondération des sommets diminue avec le nombre de cœurs et que la phase de fusion pyramidale des minima régionaux locaux entraîne ce palier pour l'algorithme parallèle de calcul des minima régionaux.



(a) *(b)*



(c)

Figure 4. 16Temps d'éxécution de la totalité de la technique de segmentation basée sur l'algorithme 16 pour des images de différentes tailles en fonction du nombre de cœurs

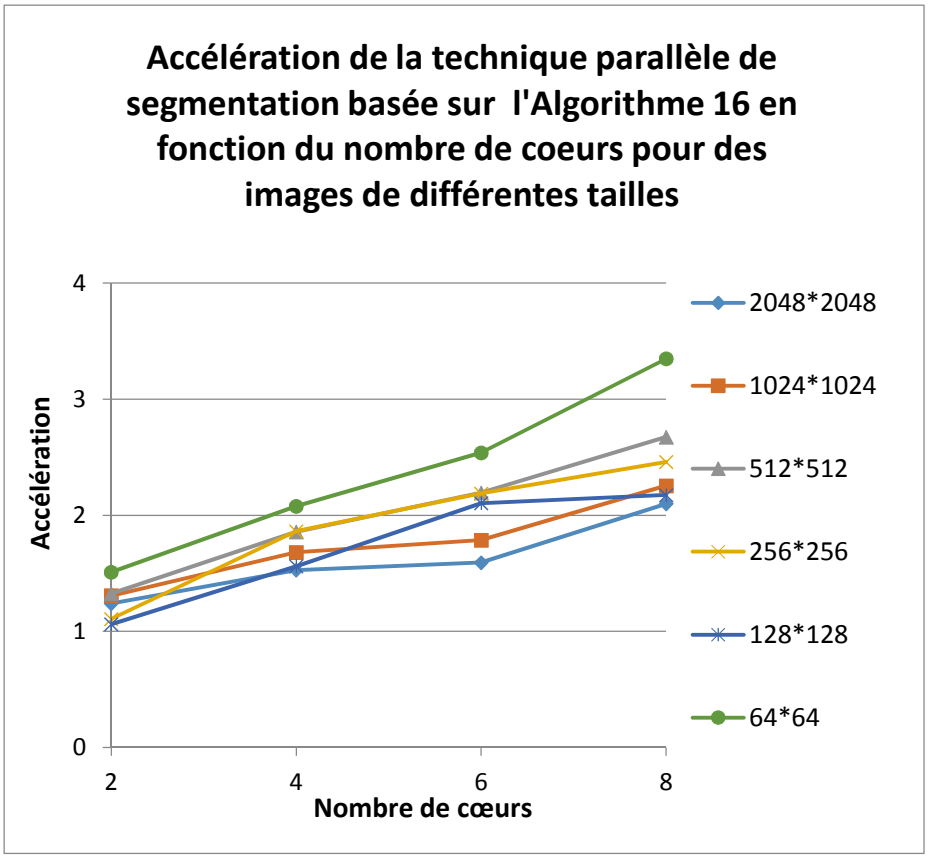


Figure 4. 17 Accélération de la totalité de la technique de segmentation basée sur l'algorithme 16 pour des images de différentes tailles en fonction du nombre de cœurs

Résolution	64×64	128×128	256×256	512×512	1024×1024	2048×2048
Accélération	3.34	2.17	2.46	2.67	2.25	2.09

Tableau 4. 2 Accélérations de la totalité de la technique de segmentation basée sur l'algorithme 16 pour des images de différentes tailles en utilisant 8 cœurs

5. Conclusion

Dans ce chapitre, nous avons présenté une technique de segmentation basée sur l'algorithme de calcul de noyau par M-bord. Cette technique est constituée de trois principales étapes: la détection des minima, la pondération des sommets (et calcul de l'ensemble de sommets minima) et enfin le calcul de noyau par M-bord. L'objectif principal dans ce chapitre est d'optimiser cette technique en termes de temps de calcul, en parallélisant chacune de ses étapes. Ainsi, nous avons proposé des algorithmes parallèles pour chaque étape. Ces algorithmes sont implémentés sur la même architecture utilisée dans le chapitre précédent. Nous avons évalué, dans un premier lieu, la parallélisation de chaque étape et, dans un second lieu, la totalité de la technique de segmentation. Puisque les algorithmes parallèles de calcul

de noyau par M-bord ont été évalués dans le chapitre 3, nous nous sommes limités à évaluer les deux premières étapes de la technique (l'étape de détection des minima et l'étape de pondération de sommets). En outre, l'évaluation de la totalité de la technique de segmentation est faite en utilisant l'algorithme parallèle de calcul de noyau par M-bord le plus performant en termes de temps d'exécution et donc qui a donné le meilleur facteur d'accélération (ce facteur atteint 5.99 pour des images de taille 2048×2048 en utilisant 8 cœurs) qui est l'algorithme basé sur l'examen des sommets au lieu des arêtes (Algorithme 16 détaillé dans la section 6 du chapitre 3).

Conclusion générale

Les travaux menés dans le cadre de cette thèse ont porté sur la parallélisation, et l'optimisation en termes de temps d'exécution de la segmentation d'images par la Ligne de Partage des Eaux (LPE), et plus précisément, la LPE d'arêtes qui constituent une notion particulière de la LPE définie dans le cadre des graphes à arêtes valuées.

Dans ce cadre, nous avons commencé dans un premier temps par présenter les différentes définitions de la LPE présentées dans la littérature ainsi que les algorithmes séquentiels qui permettent de les calculer. Par rapport à ces définitions, nous avons porté un intérêt particulier à celles de la LPE d'arêtes et les algorithmes associés. Cet intérêt est motivé par les propriétés intéressantes que possèdent les algorithmes de cette catégorie par rapport à la parallélisation notamment en termes de localité des traitements et des structures de données requises pour une implémentation optimisée en termes de temps de calcul, sur machine parallèle. Cette étude a été détaillée dans le chapitre 1.

Dans un deuxième temps, nous avons traité la problématique de l'accélération de la Ligne de Partage des Eaux, par l'exploration des stratégies de parallélisation. Nous avons d'abord rappelé les concepts et techniques de base du calcul parallèle tout en mettant l'accent sur les architectures d'implémentation qu'ils requièrent. Nous avons ensuite établi un état de l'art des principaux travaux qui se sont intéressés à la parallélisation de la Ligne de Partage des Eaux. Dans cet état, nous avons voulu couvrir à la fois différentes variantes de la LPE (LPE par inondation, LPE par ruissellement, LPE topologique …) tout en nous focalisant sur deux approches de parallélisation en l'occurrence l'approche par décomposition du domaine et l'approche par décomposition fonctionnelle. Un bilan sur les différents algorithmes parallèles de calcul des différents types de la LPE a été ainsi présenté. Dans ce bilan, nous avons fait recours à certains critères qui permettent d'évaluer les stratégies de parallélisation de la LPE.

Dans un troisième temps, nous avons abordé la parallélisation de l'algorithme de calcul de la LPE d'arêtes, appelé algorithme de calcul du noyau par M-bord. Ceci a constitué l'essentiel de nos contributions. À notre connaissance, la parallélisation de cet algorithme n'a pas encore fait l'objet de travaux dans la littérature. Compte tenu de ce fait, nous avons exploré différentes stratégies de sa parallélisation et nous avons abouti à la proposition de trois stratégies:

- La première suit l'approche de parallélisation basée sur le modèle *Split and Merge*. Elle consiste à diviser le graphe de départ en *P* partitions équilibrées et non-chevauchées, traitées en parallèle par *P* processeurs.

- La deuxième consiste à diviser les arêtes du graphe de départ en des ensembles d'arêtes indépendantes qui peuvent être traitées en parallèle. Dans ce cadre nous avons montré que dans le cas où nous considérons un graphe utilisant la 4-adjacence, l'indépendance susmentionnée des arêtes est obtenue entre l'ensemble des arêtes horizontales et celui des arêtes verticales du graphe. Les arêtes d'un même ensemble étant traitées en parallèle. Deux ensembles différents doivent cependant être considérés d'une manière alternée.
- La troisième stratégie évite le traitement en alternance qui réduit le degré de parallélisme. La particularité de cette troisième stratégie réside dans la manière selon laquelle est examiné le graphe. Ainsi, et au lieu de faire un parcours de ce dernier selon les arêtes, ce parcours est effectué selon les sommets. Nous avons rendu cela possible en proposant une modification au niveau de l'algorithme de calcul du noyau par M-bord tout en préservant son principe de base.

Afin de valider nos propositions, nous avons implémenté les différents algorithmes parallèles proposés sur une architecture multi-cœurs à mémoire partagée. Les expérimentations menées dans ce cadre ont montré que toutes les stratégies proposées ont permis un gain en termes de temps de calcul et, donc, d'accélération qui augmente avec le nombre de cœurs utilisés. De plus, ce gain est garanti quel que soit la résolution des images en entrée. D'autre part, la comparaison des performances de ces trois stratégies de parallélisation proposées a montré que la dernière basée sur l'examen des sommets du graphe de départ au lieu des arêtes a donné le meilleur facteur d'accélération.

Dans un dernier temps, nous avons proposé une parallélisation de la méthode de segmentation d'images basée sur l'algorithme de calcul de noyau par M-bord. Cette méthode comprend dans l'ordre, trois principales étapes : la détection des minima régionaux, la pondération des sommets et finalement le calcul de noyau par M-bord. L'étude de ces étapes a révélé que chacune d'elles nécessite la sortie de celle qui la précède. Nous avons, de ce fait, opté pour une parallélisation individuelle de chacune d'elles.

Pour l'étape de détection des minima régionaux, vue la nature des traitements qu'elle nécessite et qui alternent entre une recherche en largeur et en profondeur, nous avons opté pour une stratégie de parallélisation basée sur la décomposition du domaine de l'image en des sous-domaines de tailles égales et ce, afin d'équilibrer les charges de calcul entre les processeurs et de limiter les temps d'attente entre ces derniers. Chaque processeur applique la version séquentielle de l'algorithme de détection des minima sur la partition qui lui est

associée en utilisant un jeu local d'étiquettes. Le résultat final est alors obtenu par fusion des résultats locaux. Dans cette phase de fusion les processeurs adjacents communiquent afin de détecter les équivalences entre leurs étiquettes locales respectives et ce dans le cas où il y a des minima régionaux qui sont étendus sur deux ou plusieurs partitions. Cette phase de fusion permet également de détecter des faux minima qui peuvent être localement générés par un processeur. Dans notre travail, nous avons analysé et spécifié toutes les règles qui régissent cette phase de fusion. En outre, et dans un souci d'optimisation du temps de calcul, nous avons proposé une implémentation de cette phase qui opère selon une approche pyramidale.
Pour l'étape de pondération des sommets, la stratégie de parallélisation que nous avons proposé était du type massif vu que cette étape ne présente aucun problème de dépendance de données. En effet tous les traitements qu'elle demande se font d'une manière ponctuelle au niveau de chaque sommet.

Afin de valider nos propositions, nous avons implémenté la technique parallèle proposée sur la même architecture multi-cœurs utilisée pour évaluer les stratégies de parallélisation de l'algorithme de calcul de noyau par M-bord. Dans ce cadre, nous avons tout d'abord évalué la performance de la technique parallèle en termes de qualité de segmentation obtenus. Pour ce faire, nous avons comparé les résultats de cette technique par rapport aux résultats obtenus en appliquant la technique séquentielle. Ensuite, nous avons évalué d'une manière indépendante chaque étape de la technique de segmentation en termes de nombre de cœurs et de résolution des images. Les résultats obtenus ont montré que les différents algorithmes parallèles ont permis un gain en termes d'accélération en utilisant un nombre de processeurs allant de 2 à 8. Pour l'étape de calcul du noyau par M-bord, l'algorithme parallèle basé sur l'examen des sommets a donné le meilleur facteur parmi les trois algorithmes parallèles proposés. Ainsi, nous avons adopté ensuite ce dernier pour l'évaluation de la totalité de la technique. Pareil, des résultats satisfaisants en termes d'accélération ont été obtenus quel que soit la taille des images à segmenter.

Il est à noter que, outre la performance, nos propositions concernant la phase de calcul du noyau par M-bord peuvent être comparées par leur adaptation par rapport aux architectures parallèles. Ainsi la première stratégie est plutôt adaptée aux architectures multi-cœurs pour vue que la granularité du parallélisme ne soit pas très importante. La deuxième et surtout la troisième approche sont plutôt adaptées au parallélisme massif.

Au bout de ces travaux, nous pensons que les problématiques étudiées dans cette thèse ainsi que les contributions, aussi bien au niveau des stratégies proposées que des expérimentations réalisées, ouvrent les portes à de nombreuses pistes de recherche pour des travaux futurs. Parmi les pistes qui peuvent être explorées nous pouvons citer :

- L'implémentation sur architecture de type GPU de la troisième stratégie de parallélisation de l'algorithme de calcul du noyau par M-bord proposée.
- Approfondissement de l'étude de la parallélisation de l'étape de détection des minima régionaux notamment en utilisant les approches basées sur les algorithmes Union/Find et la comparaison de leurs résultats par rapport à l'approche que nous avons proposé.
- Extension des algorithmes proposés au cas des images 3D et 4D.

Bibliographie

[1] J. Cousty, L. Najman, M. Couprie, S. Clément-Guinaudeau, T. Goissen, J. Garot. "Segmentation of 4D cardiac MRI: automated method based on spatio-temporal watershed cuts," Image and Vision Computing, 28, pages 1229-1243, 2010.

[2] C. Jordan." Nouvelles observations sur les lignes de faîtes et de thalweg," Comptes Rendus des Séances de l'Académie des Sciences, 75 pages 1023-1025, 1872.

[3] H. Digabel et C. Lantuéjoul. "Iterative algorithms," Dans Proc. 2nd European Symp. Quantitative Analysis of Microstructures in Material Science, Biology and Medicine, pages 85-99, 1978.

[4] M. Couprie et G. Bertrand," Topological grayscale watershed transform," Proc. Of SPIE Vision Geometry, volume 3168, pages 136-146, 1997.

[5] S. Beucher et F. Meyer, "The morphological approach for segmentation: the watershed transformation," Centre de morphologie mathématique Ecole des Mines de Paris. Chapitre 12, pages 433-481, 1993.

[6] S. Beucher et C. Lantuéjoul, "Use of watersheds in contour detection," Proc. International Workshop on Image Processing Real-Time Edge and Motion Detection/Estimation, 1979.

[7] J. Cousty, M. Couprie, L. Najman, et G. Bertrand, "Weighted fusion graphs: merging properties and watersheds," Discrete Appl. Math., vol. 156, no. 15, pages 3011–3027, 2008.

[8] J. Cousty, G. Bertrand, L. Najman, et M. Couprie, "Watershed cuts: minimum spanning forests and the drop of water principle," IEEE Trans. Pattern Anal. Mach. Intell., 2009.

[9] L. Vincent et P. Soille, "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations," IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 13, no. 6, pages 583-598, 1991.

[10] F. Meyer, "Un Algorithme Optimal de Ligne de Partage des Eaux," Proc. Huitième Congrès AFCET, pages 847-859, 1991.

[11] F. Meyer, “Topographic Distance and Watershed Lines,” Signal Processing, vol. 38, no. 1, pages 113-125, 1993.

[12] L. Najman et M. Schmitt, “Watershed of a Continuous Function,” Signal Processing, vol. 38, no. 1, pages 68-86, 1993.

[13] A. Meijster et J. Roerdink, “A Disjoint Set Algorithm for the Watershed Transform,” Proc. European Signal Processing Conf., pages 1669-1672, 1998.

[14] A. Bieniek et A. Moga, “A Connected Component Approach to the Watershed Segmentation,” Proc. Symp. Math. Morphology and Its Applications to Image and Signal Processing, pages 215-222, 1998.

[15] G. Bertrand, “On topological watersheds”, Journal of Mathematical Imaging and Vision 22 (2-3), pages 217-230, 2005.

[16] J. Cousty, G. Bertrand, L. Najman et M. Couprie, “Watershed cuts”, 8th International Symposium on Mathematical Morphology, Rio de Janeiro, Brazil, MCT/INPE, VOL 1, pages 301–312, 2007.

[17] J. Cousty, “Lignes de partage des eaux discrètes : théorie et application à la segmentation d'images cardiaques,” Thèse de doctorat, Université Marne-La-Vallée, 2007.

[18] M. Couprie, L. Najman et G. Bertrand, “Quasi-linear algorithms for the topological watershed,” Journal of Mathematical Imaging and Vision, 22(2–3), pages 231–249, Special issue on Mathematical Morphology, 2005.

[19] L. Najman, M. Couprie et G. Bertrand, “Watersheds, mosaics and the emergence paradigm,” Discrete Applied Mathematics, 147(2-3), Special issue on DGCI, pages 301–324, 2005.

[20] L. Najman et M. Couprie, “Watershed algorithms and contrast preservation”, Discrete Geometry for Computer Imagery, France. Springer, 2886 (1), pages 62-71, 2003.

[21] Jos B. T. M. Roerdink et A. Meijster, “The watershed transform: definitions, algorithms and parallelization strategies,” Fundamenta Informaticae, 41, pages 187–228, 2000.

[22] J. Cousty, M. Couprie, L. Najman et G. Bertrand, “Grayscale watersheds on perfect fusion graphs,” In Proceedings of the 11th International Workshop on Combinatorial Image Analysis, volume 4040 de Lecture Notes in Computer Science, pages 60–73. Springer, 2006.

[23] C. Allène, “Paradigmes de segmentation de graphe : comparaisons et applications en traitement d’images,” Thèse de Doctorat en Informatique, Université Paris-Est, École doctorale ICMS, École des Ponts ParisTech – CERTIS, ESIEE - A2SI, 2009.

[24] R.E. Tarjan, “Data Structures and Network Algorithms”, SIAM, 1978.

[25] Lotufo, R.A., Falcao, A.X., Zampirolli, F.A. “Ift-watershed from gray-scale marker”, In SIBGRAPI’02, Fortaleza-CE, Brazil, pages 146–152, 2002.

[26] L. Najman et M. Couprie“, Building the component tree in quasi-linear time”. IEEE Trans. Image Processing, 15(11), pages 3531-3539, 2006.

[27] M. Thorup”, On RAM priority queues”. Dans Proc. of the 7th ACM-SIAM Symposium on Discrete Algorithms, pages 59-67, 1996.

[28] T. H. Cormen, C. Leiserson et R. Rivest, ”Introduction to algorithms”, second edition. MIT Press, 2001.

[29] J. Maxwell“, On hills and dales”. Philosophical Magazine, 4/40, pages 421-427, 1870.

[30] J. Cousty, G. Bertrand, L. Najman, et M. Couprie, “Watershed Cuts: Thinnings, Shortest Path Forests, and Topological Watersheds”, IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 32, No. 5, May 2010.

[31] R. Romero-Zaliz et J.F. Reinoso-Gordo, “An Updated Review on Watershed Algorithms”, Springer International Publishing AG 2018. C. Cruz Corona (ed.), Soft Computing for Sustainability Science, 2018.

[32] M. M. Hizem, “Recherche de chemins dans un graphe à pondération dynamique : application à l'optimisation d'itinéraires dans les réseaux routiers”, Automatique / Robotique, Ecole Centrale de Lille, 2008.

[33] C. Rossignol, “Graphes pondérés, Graphes étiquetés, Graphes probabilistes“, 2008-2009.

[34] C. Allène, J-Y. Audibert, M. Couprie, J. Cousty et R. Keriven, “Some links between min-cuts, optimal spanning forests and watersheds”. Dans Mathematical Morphology and its Applications to Signal and Image Processing, proc. 8th International Symposium on Mathematical Morphology, pages 253-264, 2007.

[35] A. Kôrbes, Giovani B. Vitor, Janito V. Ferreira, Roberto de Alencar Lotufo”, A Proposal for a Parallel Watershed Transform Algorithm for Real-Time Segmentation”. Proceedings of Workshop de Visao Computacional WVC’2009.

[36] C. Breshears, “The Art of Concurrency”, 1st edition O'Reilly, 2009.

[37] I. Foster, “Designing and building parallel programs”, Addison Wesley, 1994.

[38] M. McCool, J. Reinders, and A. Robison, “Structured Parallel Programming: Patterns for Efficient Computation”, 1st edition, Morgan Kaufmann, 2012.

[39] G. Amdahl, “Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities". In: AFIPS Conference Proceedings. Vol. 30. 1967, pages: 483-485.

[40] S. H. Roosta, “Parallel Processing and Parallel Algorithms: Theory and Computation”, Springer, 2000.

[41] M. J. Flynn, “Some Computer Organizations and their effectiveness”, IEEE Transactions on Computers, C-21(9), pages : 948–960, 1972.

[42] A. Tanembaum, “Modern Operating Systems”, Prentice Hall, 1992.

[43] J-L. Roch, “Complexité parallèle et algorithmique PRAM”, Ecole d’automne. Conception et Analyse d’Algorithmes Parallèles (CAPA 93), Port d’Albret, Landes, France, Septembre 1993.

[44] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. Beowulf, “A parallel workstation for scientific computation”, Proceedings of the 24th International Conference on Parallel Processing, pages: 11–14, Oconomowoc, WI, 1995.

[45] M. I. Gordon, W. Thies, et S. Amarasinghe, “Exploiting coarse-grained task, data, and pipeline parallelism in stream programs”, Dans ACM SIGOPS Operating Systems Review, vol. 40, no. 5. ACM, pages: 151–162, 2006.

[46] D. Skillicorn and D. Talia, “Models and languages for parallel computation”, ACM Computing Surveys, 30(2), pages :123–169, 1998.

[47] G. Nicolescu et P. J. Mosterman, “Model-based design for embedded systems”. CRC, 2009.

[48] S. Lee, S.-J. Min, and R. Eigenmann, “OpenMP to GPGPU: a compiler framework for automatic translation and optimization”, ACM Sigplan Notices, vol. 44, no. 4, pages: 101–110, 2009.

[49] C. Pheatt, “Intel R threading building blocks”, Journal of Computing Sciences in Colleges, vol. 23, no. 4, pages: 298–298, 2008.

[50] D. W. Walker and J. J. Dongarra, “MPI: A message-passing interface standard”. Supercomputer, 12(1), pages: 56–68, 1996.

[51] A. Moga, “Parallel watershed algorithms for image segmentation”. Thèse de Doctorat, Tampere University of Technology, Finland, 1997.

[52] A. Moga, T. Viero, B. P. Dobrin et M. Gabbouj, “Implementation of a distributed watershed algorithm”, Mathematical Morphology and its Applications to Image

Processing, J. Serra et P. Soille, Eds. Kluwer Acad. Publ., Dordrecht, pages: 281–288, 1994.

[53] A. Moga, B. Cramariuc, et M. Gabbouj, “Parallel watershed transformation algorithms for image segmentation”. Parallel Computing 24, pages: 1981–2001, 1998.

[54] A. Bieniek, H. Burkhardt, H. Marschner, M. N¨olle et G. Schreiber, “A parallel watershed algorithm”. 10th Scandinavian Conference on Image Analysis (SCIA’97), Lappeenranta, Finland, pages: 237–244.

[55] A. Moga, T. Viero, Gabbouj, M. Nolle, G. Schreiber et H. Burkhardt, “Parallel watershed algorithm based on sequential scanning”. Proc. IEEE Workshop on NonlinearSignal and Image processing, Neos Marmaras, Halkidiki, Greece, I. Pitas, Ed., pages: 991–994, 1995.

[56] A. Meijster, et J. B. T. M. Roerdink, “A proposal for the implementation of a parallel watershed algorithm”. In Computer Analysis of Images and Patterns, V. Hlavac and R. Sara, Eds., vol. 970 of Lecture Notes in Computer Science. Springer-Verlag, New York–Heidelberg–Berlin, pages 790–795, 1995.

[57] A. Meijster et J. B. T. M. Roerdink, “Computation of watersheds based on parallel graph algorithms”. In Mathematical Morphology and its Applications to Image and Signal Processing, P. Maragos, R. W. Shafer, and M. A. Butt, Eds. Kluwer Acad. Publ., Dordrecht, pages 305–312, 1996.

[58] C. A. Navarro, N. Hitschfeld-Kahler, et L. Mateu, “A survey on parallel computing and its applications in data-parallel problems using GPU architectures”, Communications in Computational Physics, vol. 15, no. 2, pages: 285–329, 2014.

[59] A.X. Falcao, J. Stolfi, and R. de Alencar Lotufo, “The Image Foresting Transform: Theory, Algorithm and Applications,” IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 26, no. 1, pages: 19-29, 2004.

[60] R. Tarjan, “Efficiency of a Good but Not Linear Set Union Algorithm,” J. ACM, vol. 22, pages: 215-225, 1975.

[61] L. Cabaret, L. Lacassagne et D. Etiemble, “Parallel Light Speed Labeling: an efficient connected component algorithm for labeling and analysis on multi-core processors,” Journal of Real-Time Image Processing, 2016.

[62] L. Cabaret, L. Lacassagne et D. Etiemble, “Parallel Light Speed Labeling: An Efficient Connected Component Labeling Algorithm For Multi-Core Processors,” ICIP2015.

[63] Jung-Me Park, Carl G. Looney, Hui-Chuan Chen, “Fast Connected Component Labeling Algorithm Using A Divide and Conquer Technique,” The 15th International Conference on Computers and their Applications, 2000.

[64] A. Grama, “Introduction to Parallel Computing. Pearson Education, Upper Saddle River, 2003.

[65] H. Kasim, V. March, R. Zhang, S. See, ”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, 2008.

[66] A. Körbes, R. Lotufo, ”Analysis of the watershed algorithms based on the breadth-first and depth-first exploring methods”. Pages: 133–140, 2009.

[67] E. Dijkstra, “A note on two problems in connexion with graphs,” Numerische Mathematik, vol. 1, no. 1, pages: 269–271, December 1959.

[68] E.F. Moore, "The shortest path through a maze", Proc. lnternat. Syrup. on Theory of Switching, Annals of the computation laboratory of Harvard University 30, pages: 285-292, 1957.

[69] C. Berge, “The Theory of graphs and its applications”, Wiley, 1964.

[70] T. Chen, “Gushing and immersion alternative watershed algorithm”, pages: 246–248 2001.

[71] Y. Braham, M. Akil, M. H. Bedoui, “Parallel Implementation of a Watershed Algorithm on Shared Memory Multicore Architecture”, The 9th International Conference on Machine Vision, ICMV Nice France, Novembre 2016.

[72] Y. Braham, Y. Elloumi, M. Akil, M. H. Bedoui: “Parallel Computation of Watershed Transform in Weighted Graphs on Shared Memory Machines”, Journal of Real Time Image Processing, 2018.

[73] A. Abraham, “Topological Watershed”, Technical Report no 0821, June 2008.

[74] F. Meyer et S. Beucher, “Morphological Segmentation,” J. Visual Comm. and Image Representation, vol. 1, no. 1, pages: 21-46, 1990.

[75] R. Beare, “A Locally Constrained Watershed Transform”, IEEE Transactions on Pattern Analysis and Machine Intelligence, VOL 28, NO. 7, 2006.

[76] S. Beucher, “Ligne de Partage des Eaux et Segmentation Hiérarchique”, CMM, Ecole des Mines de Paris, Ecole d'été de Morphologie Mathématique, 2000.

[77] C. Rambabu, T. Rathore, I. Chakrabarti, “A new watershed algorithm based on hillclimbing technique for image segmentation,” Conference on Convergent Technologies for Asia-Pacific Region, vol. 4, pages: 1404–1408, 2003.

[78] C. Rambabu, , I. Chakrabarti, ”An efficient hill climbing-based watershed algorithm and its prototype hardware architecture.” J. Signal Process. Syst. 52(3), pages: 281–295, 2008.

[79] L. Najman, J. Cousty, M. Couprie, H. Talbot, S. Clément-Guinaudeau, T. Goissen et J. Garot, “An open, clinically-validated database of 3D+t cine-MR images of the left ventricle with associated manual and automated segmentations”, The Insight Journal Special Issue entitled ISC/NA-MIC Open Science Workshop at MICCAI, Juin 2007.

[80] J. Lebenberg, I. Buvat, A. Lalande, P. Clarysse, C. Casta, et al. “Non-supervised Ranking of Different Segmentation Approaches: Application to the Estimation of the Left Ventricular Ejection Fraction From Cardiac Cine MRI Sequences”, IEEE Transactions on Medical Imaging, Institute of Electrical and Electronics Engineers (IEEE), 31 (8), pages: 1651-1660, 2012.

[81] A. Meijster, et J. B. T. M. Roerdink, “The Implementation of a Parallel Watershed Algorithm”. In Proc. Computing Science in the Netherlands, Utrecht, pages 134-142, 1995.

[82] D. B. K. Trieu et T. Maruyama, "Real-time image segmentation based on a parallel and pipelined watershed algorithm". Journal of Real-Time Image Processing, pages: 319–329, Décembre 2007.

[83] D. B. K. Trieu et T. Maruyama, "A pipeline implementation of a watershed algorithm on FPGA", International Conference on Field Programmable Logic and Applications, Aout 2007.

[84] S. Beucher, L. Fabrice et S. Raphael, "Réalisation de la Ligne de Partage des Eaux par File d'Attente Hiérarchique Parallèle: Etude Algorithmique", Juin 1997.

[85] H. Sun, J. Yang, and M. Ren. “A fast watershed algorithm based on chain code and its application in image segmentation.” Pattern Recognition Letters, 26(9), pages: 1266–1274, 2005.

[86] A. Körbes, G. B. Vitor, R. de Alencar Lotufo, J. V. Ferreira, “Analysis of a step-based watershed algorithm using CUDA”, Nature-Inspired Computing Design, Development, and Applications, 2012.

[87] Y. Braham, Y. Elloumi, M. Akil, M. H. Bedoui, “Parallélisation d’un Algorithme de Calcul de Coupures Par Lignes de Partage des Eaux dans les Graphes à Arêtes Valuées”, 8ème WORKSHOP AMINA, Applications Médicales de l'Informatique : Nouvelles Approches, Novembre 2016.

[88] M. Swiercz, M. Iwanowski, “Fast, parallel watershed algorithm based on path tracing”, International Conference on Computer Vision and Graphics ICCVG, Berlin pp 317-324, 2010.

[89] C. Fiorio, “Modélisation, Analyse, Représentation des Images Numériques Approche combinatoire de l’imagerie”, Habilitation à Diriger les Recherches, Informatique, Université Montpellier II, Novembre 2008.

[90] L. Moumoun, M. El Far, M. Chahhou, T. Gadi et R. Benslimane, “Solving the 3D watershed over-segmentation problem using the generic adjacency graph”, 2010.

[91] A. Procházka, O. Vysata, E. Jerhotova, “Wavelet use for reduction of watershed transform over-segmentation in biomedical images processing”, 2010.

[92] H. Zhu, B. Zhang, A. Song, W. Zhang, “An improved method to reduce over-segmentation of watershed transformation and its application in the contour extraction of brain image”, pages: 407–412, 2009.

[93] V. Gies et T. Bernard, “Statistical solution to watershed over-segmentation”. International Conference on Image Processing (3), pages: 1863–1866, 2004.

[94] D. B. K. Trieu et T. Maruyama, “Implementation of a Parallel and Pipelined Watershed Algorithm on FPGA”, IEEE, 2006.

[95] M.H.F. Wilkinson, H. Gao, W. H. Hesselink, J.E. Jonker et A. Meijster, ”Concurrent computation of attribute filters using shared memory parallel machines”. IEEE Trans. Pattern Anal. Mach. Intell. 30(10), pages: 1800-1813, 2008.

[96] R. Mahmoudi, “Real time image processing: algorithm parallelization on multicore multithread architecture”. Thèse de Doctorat en Informatique, Université Paris-Est, 2011.

[97] R. Mahmoudi, M. Akil et M. H. Bedoui, “Concurrent computation of topological watershed on shared memory parallel machines”, Parallel Computing 69, pages: 78–97, 2017.

[98] J. Van Neerbos, L. Najman, M.H.F. Wilkinson, “Towards a Parallel Topological Watershed: First Results”. In: Soille P., Pesaresi M., Ouzounis G.K. Mathematical Morphology and Its Applications to Image and Signal Processing. ISMM. Lecture Notes in Computer Science, vol 6671. Springer, Berlin, Heidelberg 2011.

[99] J. D. Touloumdjian et S. Sadough, “Segmentation par Ligne de Partage des Eaux sous contraintes”, Ecole Nationale Supérieure des Télécommunications, cours DEA ATS, Avril 2004.

[100] R. Lerallut, E. Decencière et F. Meyer, “Image filtering using morphological amoebas”, Image and Vision Computing 25, pages : 395–404, 2007.

[101] P. Perona et J. Malik, “Scale Space and Edge Detection unisng Anisotropic”, IEEE Transactions on Pattern Analysis and Machine Intelligence, VOL 12 (7), 1990.

[102] C. Tauber, “Filtrage anisotrope robuste et segmentation par B-spline snake : application aux images échographiques”, thèse en Informatique et Télécommunications Spécialité Informatique de l’Image et du Langage, Institut National Polytechnique de Toulouse, 2005.

[103] R. Manning, “On the flow of Water in Open Channels and Pipes”, Transactions Institute of Civil Engineers of Ireland, vol. 20, pages: 161-209, 1891, Supplement, vol 24, pages 179-207, 1895.

[104] G. Grieg, O. Kubler, R. Kikinis, and F. A. Jolesz, “Nonlinear Anisotropic Filtering of MRI Data”, IEEE Transactions on Medical Imaging, 11(2), pages: 221-232, 1992.

[105] A. Geist et al., “PVM 3.0 User's Guide and Reference Manual”, Technical Report RNL/TM-12187, Oak Ridge National Laboratory, 1993.

[106] D. Nuguet, “A massively parallel implementation of the watershed based on automata”. In: IEEE International Conference on Application-Specific Systems, Architectures and Processors, pages: 42–52, 1997.

[107] H. Zhou, X. Yang, Y. Tang, N. Xiao, “Further Optimized Parallel Algorithm of Watershed Segmentation Based on Boundary Components Graph”, H. Jin et al. (Eds.): NPC 2004, IFIP International Federation for Information Processing, pages: 498-501, 2004.

[108] R. Ayari, A. Ben Abdallah, F. Ghorbel et M. H. Bedoui, “Analysis of regional deformation of the heart left ventricule”, IRBM, vol. 38, no 1, pages: 90-97, 2017.

[109] A. Ben Abdallah, F. Ghorbel, Kaouther Chatti, H. Essabbah, M. H. Bedoui. “A new uniform parameterization and invariant 3-D spherical harmonic shape descriptors for shape analysis of the heart's left ventricle-A Pilot Study”, Pattern Recognition Letters, 31, pages: 1981-1990, 2010.

[110] R. Mahmoudi et M. Akil, “Analyses of the Watershed Transform”, International Journal of Image Processing, 5 (5), pages: 521-541, 2011.

[111] R. Enficiaud. “Algorithmes multidimensionnels et multispectraux en Morphologie Mathématique: Approche par méta-programmation”, Thèse de Doctorat, Centre de Morphologie Mathématique, École des Mines de Paris, 2007.

[112] LR. Dice, “Measures of the amount of ecologic association between species”, Ecology, 26(3), pages : 297–302, 1945.

[113] Paul Jaccard, “Bulletin de la Société vaudoise des sciences naturelles”, 37, page: 241-272, 1901.