Extensibilité des moyens de traitements pour les données issues des vastes systèmes d'informations géographiques

Hiep-Thuan Do

UNIVERSITÉ D'ORLÉANS

ÉCOLE DOCTORALE SCIENCES ET TECHNOLOGIES

LABORATOIRE : LIFO

THÈSE présentée par :

Hiep-Thuan DO

soutenue le :

pour obtenir le grade de : Docteur de l'université d'Orléans

Discipline/ Spécialité : Informatique

Extensibilité des moyens de traitements pour les données issues des vastes systèmes d'informations géographiques

THÈSE dirigée par :

Sébastien LIMET	Professeur, Université d'Orléans
Emmanuel MELIN	Maître de Conférence, Université d'Orléans

RAPPORTEURS :

Hubert CARDOT	Professeur, Université François Rabelais de Tours
Yves DENNEULIN	Professeur, IMAG - Grenoble

JURY :

François ANTON	Professeur, Université technologique du Danemark
Hubert CARDOT	Professeur, Université François Rabelais de Tours
Yves DENNEULIN	Professeur, IMAG - Grenoble
Sébastien LIMET	Professeur, Université d'Orléans
Frédéric LOULERGUE	Professeur, Université d'Orléans
Emmanuel MELIN	Maître de Conférence, Université d'Orléans
Daniel PIERRE	Directeur, Société Géo-Hyd

Remerciements

Je tiens en premier lieu à remercier M. Yves DENNEULIN et M. Hubert CARDOT qui m'ont fait l'honneur d'être rapporteur de mon manuscrit de thèse.

Je tiens en particulier à exprimer ma plus sincère gratitude à Sébastien LIMET, qui m'a offert la possibilité de réaliser cette thèse et a accepté de diriger ces recherches. J'adresse mes remerciements les plus chaleureux à Sébastien LIMET et Emmanuel MELIN qui m'ont encadré toute la thèse dans une atmosphère amicale propice à une recherche enjouée. Tout au long de mes travaux de recherche, ils m'ont patiemment écouté, conseillé et guidé.

J'exprime ma reconnaissance aux membres du jury qui ont accepté d'évaluer les travaux présentés dans ce manuscrit et de se déplacer afin de participer à ma soutenance de la thèse.

Je tiens aussi à remercier Le Conseil Général du Loiret qui a donné son accord et a financé pour ces travaux. J'aimerais également remercier la société Géo-Hyd à Orléans, un autre partenaire de notre projet eXtenGIS, qui a aussi fourni des modèles numériques d'élévation pour les tests dans le cadre de la publication.

Je remercie aussi Frédéric LOULERGUE qui m'a fourni les informations, concernant sur le projet eXtenGIS, à partir desquelles j'ai pu contacter à Sébastien LIMET pour ma candidature de thèse.

Je tiens aussi à remercier Sophie ROBERT pour des cours de français et pour ses réponses dans la programmation parallèle. En particulier, grâce à sa gentillesse, je ne sens pas étranger à Orléans, aussi en France.

J'exprime aussi mes remerciements à Sylvie Haouy Maure pour sa disponibilité et son aide lors de mes tests sur la grappe des ordinateurs au LIFO.

Je remercie également les membres du LIFO, en particulier, les thésards du LIFO, non seulement pour des repas au restoU, mais aussi pour leurs discussions dans la salle de détente pour l'amélioration de ma langue française.

Enfin, je remercie affectueusement mes parents, ma famille et tous mes amis pour leur soutien permanent et inconditionnel.

Table des matières

I INTRODUCTION **5**

II PROBLÉMATIQUES **11**

1 Gros volumes de données et Calcul parallèle de haute performance **13**
1.1 Gros volume de données 14
1.2 Calcul parallèle de haute performance 14
1.2.1 Introduction 14
1.2.2 Classification des Architectures parallèles selon Flynn 15
1.2.3 Classification des Architectures parallèles selon la mémoire . . 18
1.2.4 Modèles de Programmation parallèle 20
1.2.5 Exemples de librairies facilitant la parallélisation 22
1.2.6 Conclusion 23

2 Analyse Hydrologique des Terrains **25**
2.1 Modélisation de l'écoulement de l'eau 28
2.1.1 Notion de pente 28
2.1.2 Modification d'homotopie 29
2.2 Méthodes Séquentielles 30
2.2.1 Mono direction d'écoulements 31
2.2.2 Multiples directions d'écoulements 33
2.2.3 Plate-forme TerraFlow et TerraStream 35
2.3 Parallélisation de modèles de directions d'écoulements 38
2.4 Conclusion 39

3 Ligne de Partage des Eaux dans la Morphologie Mathématique **41**
3.1 Ligne de Partage des Eaux (LPE) 42
3.1.1 LPE par simulation d'inondation 42
3.1.2 LPE par simulation de pluie 45
3.2 Parallélisation de la Ligne de Partage des Eaux 46
3.2.1 Parallélisation de la LPE dans le cadre de la mémoire distribuée 46
3.2.2 Parallélisation de la LPE dans le cadre de la mémoire partagée 49
3.3 Ligne de Partage des Eaux et Fusion hiérarchique de Région 49
3.3.1 Conclusion 51

4 Conclusion **53**

III PARAFLOW : ANALYSE PARALLÈLE DE LPE EN MÉMOIRE PRINCIPALE 55

5 Principes de base de ParaFlow 57
5.1 Arbres Couvrants de Poids Minimum . . . 60
5.1.1 Définition . . . 60
5.1.2 Algorithme séquentiel de Borůvka . . . 60
5.2 Calcul de MST adapté au traitement des MNT . . . 61
5.2.1 Notations et définitions . . . 61
5.2.2 Délimitation des cuvettes . . . 62
5.2.3 Hiérarchie des bassins versants . . . 64
5.2.4 Règle Ω . . . 66
5.3 Calcul des flots d'accumulation . . . 69
5.3.1 Objectifs . . . 69
5.3.2 Calcul des flots d'accumulation locaux . . . 70
5.3.3 Calcul des flots d'accumulation globaux . . . 70

6 L'implantation parallèle de ParaFlow 73
6.1 Calcul de l'étape initiale . . . 74
6.1.1 Division des données . . . 74
6.1.2 Délimitation parallèle des cuvettes . . . 74
6.1.3 Calcul parallèle du graphe de connectivité . . . 79
6.2 Calcul parallèle hiérarchique des bassins versants . . . 82
6.2.1 Parallélisation de l'algorithme de Borůvka . . . 83
6.2.2 Implémentation dans ParaFlow . . . 84
6.3 Calcul parallèle des flots d'accumulations . . . 88
6.3.1 Écoulement de l'eau à l'intérieur des cuvettes . . . 88
6.3.2 Modélisation des cours d'eau entre les cuvettes . . . 90

7 ParaFlow : Résultats expérimentaux 97
7.1 Calcul parallèle des cuvettes . . . 98
7.2 Calcul parallèle de la hiérarchie . . . 98
7.3 Calcul parallèle des flots d'accumulations . . . 104

IV ParaFlow-OOC 109

8 Introduction de ParaFlow-OOC 111

9 Détails de ParaFlow-OOC 115
9.1 Calcul parallèle des directions d'écoulement . . . 115
9.2 Calcul parallèle des cuvettes . . . 116
9.3 Calcul parallèle du graphe de connectivité . . . 117
9.4 Calcul parallèle hiérarchique des bassins versants . . . 119
9.5 Tests . . . 119

V CONCLUSION ET PERSPECTIVES 121

10 Conclusion 123

11 Perspectives 127

Bibliographie 129

Table des figures

1.1 Lidar scanner 14
1.2 Architecture SISD (Single Instruction Single Data) 16
1.3 Architecture SIMD (Single Instruction Multiple Data) 17
1.4 Architecture MISD (Multiple Instruction Single Data) 17
1.5 Architecture MIMD (Multiple Instruction Multiple Data) 18
1.6 Modèle de programmation de SPMD 18
1.7 Modèle de programmation de "Data-Parallel" 20
1.8 Modèle de programmation à mémoire partagée 21
1.9 Modèle de programmation à passage de messages 22

2.1 Analyse de bassin visuel basée sur la ligne de mire 27
2.2 Exemple de bruits (puits et plateau) dans un MNT 29
2.3 Modèle D8 (Deterministic Eight-neighbor en anglais) 32
2.4 Modèle D∞ pour le calcul des directions d'écoulements 34
2.5 Pipeline d'analyse de TerraStream à partir de nuages de points . . . 37

3.1 Segmentation d'Images dans la Morphologie Mathématique 42
3.2 Simulation d'immersion dans la LPE 43
3.3 Simulation de pluie dans la LPE 45

5.1 Exemple de graphe G connexe, avec un MST (traits pleins). 60
5.2 Illustration de l'algorithme de Borůvka 62
5.3 Calcul des cuvettes et du graphe de connectivité GC^0 63
5.4 Calcul de la hiérarchie : Premier niveau 65
5.5 Calcul de la hiérarchie : deuxième niveau 66
5.6 Calcul des cuvettes et du graphes des connectivités avec règle Ω . . . 68
5.7 Hiérarchie avec règle Ω : Première étape 69
5.8 Hiérarchie avec règle Ω : Deuxième étape 69

6.1 Calcul parallèle des directions d'écoulements 76
6.2 Calcul parallèle local des bassins versants aux *Proc. i* 78
6.3 Graphe de dépendances GDG global 79
6.4 Calcul de ${GC^0}_i$ dans un sous-domaine D_i 81
6.5 Finalisation ${GC^0}_0$ correspondant au *Proc. 0* 82
6.6 Algorithme de Borůvka dans le calcul parallèle des cuvettes 84
6.7 Propagation FA à partir d'une cuvette A to son parent B 91

7.1 Plate-forme MIREV au LIFO 97
7.2 La Loire en France avec des cuvettes calculées par ParaFlow 98
7.3 Comparaison entre ParaFlow et GRASS sur une partie de la Loire . 99
7.4 Bassin versant de la Loire en France 100

7.5 Bassin Versant du Niger 100
7.6 Les bassins versants en Europe (SRTM Europe 90m) 101
7.7 Bassins versants en Asie : La détaille du Mékong 102
7.8 Accélération relative en fonction du nombre de processeurs ($|P|$) . . 104
7.9 Cours d'eau principaux en Loire-Bretagne (France) 105
7.10 Cours d'eau principaux du continent européen 105
7.11 Flots d'accumulations d'un MNT sur trois plateformes 106
7.12 Accélération relative en fonction du nombre de processeurs $|P|$ . . . 107
7.13 Comparaison du temps d'exécution pour MNT_LB4 entre TerraFlow et ParaFlow 108

8.1 Modèle de E/S en deux niveaux 112
8.2 Modèles de disques parallèles : deux niveaux 112
8.3 Modèles de disques parallèles : multiples niveaux hiérarchiques . . . 113

9.1 Découpage des domaines en dalles 116

Liste des tableaux

1.1 Classification des architectures parallèles selon Flynn 16

7.1 Nombre de sommets à chaque niveau de la hiérarchie 103
7.2 Temps d'exécution (en secondes) pour de petits MNT 103
7.3 Temps d'exécution (en secondes) pour de gros MNT 103
7.4 Temps d'exécution de calcul parallèle des flots d'accumulations (en secondes) 106
7.5 Comparaison de temps d'exécution (en secondes) du MNT_LB4 entre TerraFlow et ParaFlow 107

Première partie

INTRODUCTION

L'évolution des Systèmes d'Informations Géographiques (SIG) et de leur capacité à répondre à des problématiques environnementales est étroitement liée à l'évolution des concepts et des technologies informatiques, aux données géolocalisées disponibles (information géographique) et à notre capacité à exprimer des besoins clairs sur des approches scientifiques transversales, globales et systémiques comme par exemple dans le cadre de la résolution de problématiques liées à l'eau et l'environnement.

Depuis plus de 30 ans maintenant, l'essor des SIG s'est construit sur la base de ce triptyque, bénéficiant tour à tour des avancées des moyens informatiques, de l'accessibilité à l'information géographique ou en fonction de nos besoins ou de notre manière d'appréhender les problématiques environnementales. Actuellement, le contexte est plus que favorable. En effet l'information géographique est en plein essor, d'importants programmes et initiatives ont permis d'augmenter la quantité de données disponible ainsi que leur qualité (traçabilité, interopérabilité,...). De plus les problématiques environnementales se posent et s'expriment de manière globale et transversale, le besoin en terme d'outil d'aide à la décision n'a jamais été aussi fort. Ces deux points sont solidement assis sur un contexte juridique et organisationnel fort, relayés par une demande importante des pouvoirs publics et des gestionnaires de territoire, voir même du citoyen. Ces derniers veulent "voir" les données, percevoir et comprendre les phénomènes, imaginer et concevoir de nouveaux modes de gestion ou d'aménagement du territoire.

D'un point de vue strictement informatique, depuis l'avènement des systèmes d'exploitation type "Windows" et des SIG dit "bureautiques", les logiciels SIG n'ont finalement que peu évolué depuis 10 ans. Les quelques évolutions notables se sont exprimées en termes de facilité d'accès (notamment avec Internet) et de plus grande intégration dans les systèmes d'exploitation. Du point de vue de la capacité de traitements, les logiciels SIG sont toujours et encore complètement liés à la puissance de calcul des processeurs actuels.

Les données et les méthodologies existent, les demandes et les besoins sont forts mais les réponses apportées par les SIG actuels sont rarement appropriées, mais surtout la plupart du temps bridées par les capacités de calculs offerts par un ordinateur de bureau. Le sujet de la présente thèse est donc d'arriver à faire sauter ce verrou technologique afin de proposer des solutions qui soient à la hauteur de la qualité des données actuellement disponibles et aux demandes des utilisateurs finaux des SIG. La réponse à ces enjeux passera par l'utilisation des moyens de calculs puissants et peu coûteux que fournissent les grappes d'ordinateurs.

L'utilisation de grappes d'ordinateurs pour faire du calcul intensif à bas coût sur des gros volumes de données est une idée qui est apparue au milieu des années 90 avec le projet Beowulf impliquant notamment la NASA. L'augmentation croissante des capacités de calcul et de stockage des ordinateurs type PC à coût quasi constant et l'avènement des réseaux très haut débit ont renforcé la pertinence de cette architecture de machine parallèle. Les deux principales qualités des grappes de PC sont leur faible coût par rapport aux machines parallèles dédiées telles celles proposées par Cray d'une part et leur très grande évolutivité d'autre part. En effet il est très facile d'augmenter leur puissance de calcul en y rajoutant des nœuds par exemple.

Le revers de la médaille est que les grappes de PC ont une architecture fortement découplée et que notamment les nœuds de la grappe n'ont pas de mémoire partagée ce qui complique singulièrement leur programmation. La difficulté principale est de s'assurer que le coût des communications entre les nœuds de la grappe pour effectuer le calcul en parallèle n'est pas supérieur au gain obtenu par la distribution des calculs sur ces nœuds.

Depuis les années 90, de nombreux travaux ont été effectués afin d'essayer d'exploiter au mieux les ressources des grappes d'ordinateurs. La plupart des applications développées sur ce type d'architecture l'ont été de manière adhoc en utilisant des bibliothèques de communication type MPI. Ce type de développement a pour inconvénients principaux de nécessiter une grande expertise de la part des programmeurs et de produire des codes peu réutilisables. C'est pour cette raison que l'étude d'outils généraux pour la programmation de machines distribuées fait encore l'objet de recherches intensives dans le domaine de l'informatique répartie. Parmi les solutions existantes, on retrouve des approches qui visent à simuler une mémoire partagée afin de pouvoir réutiliser les algorithmes connus pour les machines parallèles dédiées. Une autre solution qui est très largement étudiée est la programmation par composant, c'est à dire que l'application est décrite sous la forme de composants qui effectuent certaines tâches et qui peuvent communiquer entre eux. L'avantage de cette approche est de permettre une grande réutilisabilité des composants mais le principal inconvénient reste la difficulté à optimiser les communications entre ces derniers.

L'objectif principal de la thèse est de proposer une plateforme calcul répartie à base de grappes d'ordinateurs, s'appelle ParaFlow, pour la parallélisation de traitements sur des gros volumes de données géo-référencées notamment dans le cadre de la délimitation des bassins versants des grands fleuves et le calcul des flots d'accumulation. L'idée consiste à définir des techniques pour traiter des modèles numériques de terrain (MNT) pouvant comporter plusieurs milliard de points en tirant partie de la puissance de calcul et des capacités de stockages des grappes d'ordinateur. Un autre objectif est d'essayer de tirer partie du maximum d'information contenue dans le MNT initial sans le modifier. Dans ce cadre, nous avons développé une technique de calcul parallèle d'une forêt d'arbres couvrants minimums représentant le parcours de l'eau de chaque point du MNT vers la mer. Cette technique débute par une délimitation des cuvettes (ensemble de points allant vers le même minimum local) contenues dans le MNT. Ensuite une hiérarchie de déversement des cuvettes les unes dans les autres est construite jusqu'à obtenir les bassins versants des fleuves. Cette construction est dirigée pour que l'eau prenne le chemin le moins couteux pour aller vers la mer et pour que les bassins versants de deux fleuves différents ne soient pas fusionné. Par ailleurs la technique utilisée permet aussi de tenir compte de l'absence d'information au bord du MNT. La technique a été implémentée en C++ avec la librairie de communication MPI et des tests ont été effectués sur des MNT allant jusqu'à 10 milliards de points. Les résultats obtenus en terme performance montrent l'extensibilité de la méthode (c'est à dire que plus on a d'ordinateurs à disposition plus le calcul est rapide). De plus sur le plan géologique, les résultats sont tout à

fait cohérents par rapport aux délimitations des bassins versants (manuelles) qui ont été effectuées notamment en Europe. Concernant la partie flots d'accumulation, c'est à dire déterminer combien de points du MNT se déversent en chaque point, ce calcul devient très couteux sur des MNT de très grande taille et demande donc une implémentation parallèle efficace qui puisse passer à l'échelle. Notre plateforme ParaFlow développée et implémentée se montre aussi extensible et cohérente vis à vis des résultats obtenus par des logiciels de traitement de MNT reconnus (ArcGIS ou GRASS).

Enfin la dernière partie du travail de cette thèse consiste à étendre les résultats obtenus précédent en utilisant des techniques dites "out-of-core" lorsque le problème à traiter n'entre pas en mémoire centrale des ordinateur de la grappe de PC. L'idée consiste à désigner un autre parallèle algorithme en utilisant la mémoire externe, par exemple la disque dur. Ces algorithmes doivent être optimisées pour chercher efficacement et faire des accès aux données stockées dans la mémoire externe lente. Les premiers résultats obtenus par cette extension sont tout à fait encourageants.

Deuxième partie

PROBLÉMATIQUES

Chapitre 1

Gros volumes de données et Calcul parallèle de haute performance

Sommaire

1.1 Gros volume de données 14
1.2 Calcul parallèle de haute performance 14
1.2.1 Introduction 14
1.2.2 Classification des Architectures parallèles selon Flynn 15
1.2.3 Classification des Architectures parallèles selon la mémoire . 18
1.2.4 Modèles de Programmation parallèle 20
1.2.5 Exemples de librairies facilitant la parallélisation 22
1.2.6 Conclusion 23

1.1 Gros volume de données

Les géo-sciences doivent gérer des volume de données de plus en plus gros, soit que les études portent sur des territoires de plus en plus vastes, soit que les appareils de mesures soient de plus en plus précis, comme les lasers aéro-portés illustrés dans **Figure 1.1**. Ces appareils permettent de construire des modèles numériques de terrains (en abrégé MNT) de plus en plus fins. À titre d'exemple, pour des raisons pratiques de capacité de traitement, les MNT couramment utilisés en géomatique travaillent au pas de 50 mètres alors que l'on est capable de générer des MNT au pas de 1 mètre. Cela signifie que l'on passe, pour un territoire de 100 km^2, d'un modèle à 4 millions de points à un modèle à 10 milliards de points. Cette explosion des volumes de données à traiter nécessite des moyens de calculs bien plus importants que ceux utilisés jusqu'alors.

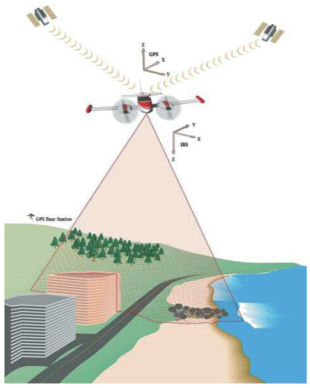


FIGURE 1.1 – Lidar scanner

Ces données initiales peuvent être utilisées pour la visualisation, mais aussi pour l'extraction d'informations. Si nécessaire, ce processus peut être répété afin de raffiner le résultat ou bien d'en obtenir un nouveau. Cette boucle d'analyse n'est concevable que lorsque chacune des itérations est réalisée dans un temps raisonnable. Les machines séquentielles standards et les logiciels SIG ne sont pas adaptés pour de telles spécifications. Il en résulte que seule l'utilisation de machines parallèles et l'adaptation du code peut permettre d'espérer véritablement passer à l'échelle.

1.2 Calcul parallèle de haute performance

1.2.1 Introduction

Depuis l'apparition des ordinateurs, la puissance de calcul n'a fait que croître afin de résoudre des problèmes de complexités croissantes. Le calcul scientifique a connu une évolution rapide. Il est largement utilisé et se développe dans tous les domaines, tels que, la prévention météo, la simulation scientifique, mécanique, aérodynamique, électrique, biologique, etc...

La puissance de calcul repose sur deux grandes notions :

– la *latence de traitement* ;
– le *débit de traitement*.

La première notion, la latence de traitement, représente le temps nécessaire pour l'exécution d'un traitement tandis que le débit de traitement représente le nombre de traitements exécutables par unité de temps. Ces deux notions peuvent être indépendantes. L'augmentation de la puissance de calcul consiste dans la réduction de la latence de traitement ou dans l'augmentation du débit de traitement. La réduction de la latence de traitement est plus difficile à obtenir que l'augmentation du débit de traitement. De plus, la puissance de calcul dépend aussi de la capacité et de l'organisation de la mémoire d'un ordinateur. Dans le modèle séquentiel, un programme est exécuté par un unique processus. Ce processus a accès à la mémoire du processeur. Plusieurs applications, telles que la simulation, la modélisation et l'optimisation numérique, requièrent des ensembles de données (météo, SIG, marketing, etc) dont la taille est supérieure à la capacité d'adressage d'un ordinateur séquentiel. De plus, les exigences du temps de calcul dans ces applications sont importantes. La solution unique, afin de surmonter ces problèmes, est le parallélisme. Certaines organisations d'architectures parallèles permettent donc d'adresser plus de mémoire que des architectures séquentielles.

Historiquement, l'évolution des architectures parallèles a commencé à partir de l'architecture séquentielle de Von Neumann qui offre un flux de contrôle unique dont les instructions exécutent séquentiellement les opérations sur des opérandes scalaires. Une révolution informatique c'est produite lorsque le calcul scientifique basé sur les machines de type "mainframes" a été remplacé par celui basé sur le réseaux. Dans les années 1970s, l'introduction des systèmes vectoriels a marqué le début des supercalculateurs modernes. De nombreux ordinateurs parallèles commerciaux sont ensuite apparus dans les années 1980s et 1990s. Ils peuvent être généralement classés en deux catégories principales : les systèmes à mémoire partagée et les systèmes à mémoire distribuée. Au début des années 1990s, alors que les systèmes multiprocesseurs vectoriels ont atteint leur plus large diffusion, une nouvelle génération de machines massivement parallèles a été introduite sur le marché. Ces systèmes se sont avérés autant ou plus performants que les systèmes vectoriels. Le nombre de processeurs peut atteindre une centaine voir un millier de processeurs dans un système massivement parallèle. Les calculs scientifiques, liées à ces architectures de machines, deviennent de plus en plus ambitieux. Mais les algorithmes, les paradigmes et les méthodes doivent évoluer pour pleinement tirer partie des points forts de ces architectures.

Nous présentons ici un bref panorama des architectures parallèles, de leurs contraintes et des moyens de les programmer avant de conclure sur nos choix.

1.2.2 Classification des Architectures parallèles selon Flynn

Une architecture parallèle est essentiellement un ensemble de processeurs qui coopèrent et communiquent. Les premières architectures parallèles furent des réseaux d'ordinateurs et des machines vectorielles faiblement parallèles (IBM 360-90

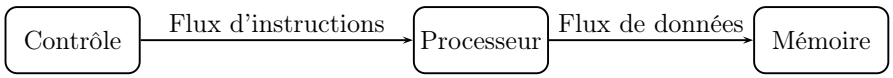


FIGURE 1.2 – Architecture SISD (Single Instruction Single Data)

vectoriel, IRIS 80 triprocesseurs, CRAY 1 vectoriel, etc).

La classification la plus utilisée est historique, elle est connue sous le nom de classification de Flynn (voir [Flynn 1972]). Elle classe les architectures en fonction des relations entre les unités de traitement et les unités de contrôle. Les unités de traitement calculent chacune un flux de données (Data Stream en anglais, ou DS). Les unités de contrôle déroulent chacune un programme qui consiste en un flux d'instructions (Instruction Stream en anglais, ou IS). Chacun de ces types peut avoir un seul des deux états possibles : mono ou multiples. On peut trouver cette classification dans la **Table 1.1**.

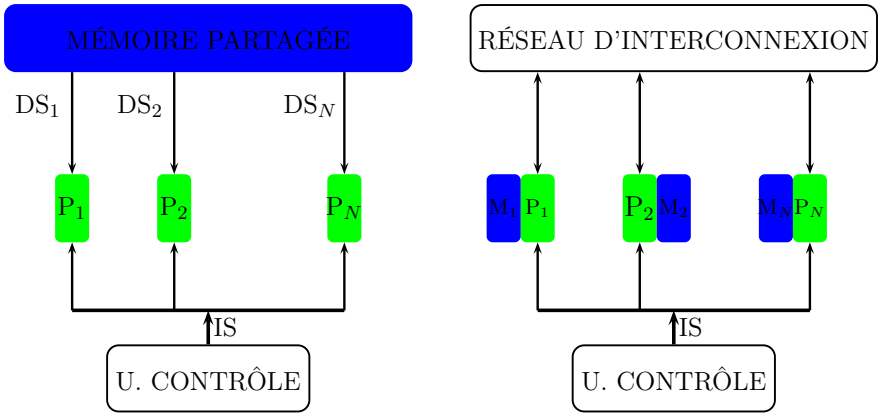
Flux	mono flux d'instructions	multiples flux d'instructions
mono flux de données	SISD	MISD
multiples flux de données	SIMD	MIMD

TABLE 1.1 – Classification des architectures parallèles selon Flynn

L'architecture SISD (Single Instruction Single Data en anglais) est une architecture séquentielle. Elle correspond au modèle classique de Von Neuman illustrée dans **Figure 1.2**. Le traitement n'est effectué qu'avec un seul flux d'instructions sur un seul flux de données. Cette architecture peut être combinée avec la technique du pipeline afin de permettre l'exécution de plusieurs instructions. L'idée consiste à subdiviser les instructions en une série d'opérations plus fines et chacune des opérations peut être exécutée dans une seul phase de pipeline à moment donné.

La deuxième architecture (voir **Figure 1.3**) s'appelle SIMD (Single Instruction Multiple Data en anglais). Une seule unité de contrôle gère le séquencement du programme pour plusieurs unités de traitement. En pratique, les unités de traitement fonctionnent de manière synchrone et reçoivent les mêmes instructions en même temps. Comme chaque unité de traitement calcule sur un flux de données différentes, la même opération est appliquée à plusieurs données simultanément. Cette architecture est ainsi capable de traiter plusieurs données à la fois. On classe dans cette catégorie également les machines vectorielles, les réseaux cellulaires et les réseaux systoliques.

L'architecture MISD (Multiple Instruction Single Data en anglais) peut exécuter plusieurs instructions en même temps sur un seul flux de données (voir **Figure 1.4**). Le mode pipeline d'exploitation du parallélisme de contrôle correspond assez bien à cette classe. Le pipeline du type MISD correspond à un enchaînement de macro-



(a) SIMD avec mémoire partagée (b) SIMD avec mémoire distribuée

FIGURE 1.3 – Architecture SIMD (Single Instruction Multiple Data)

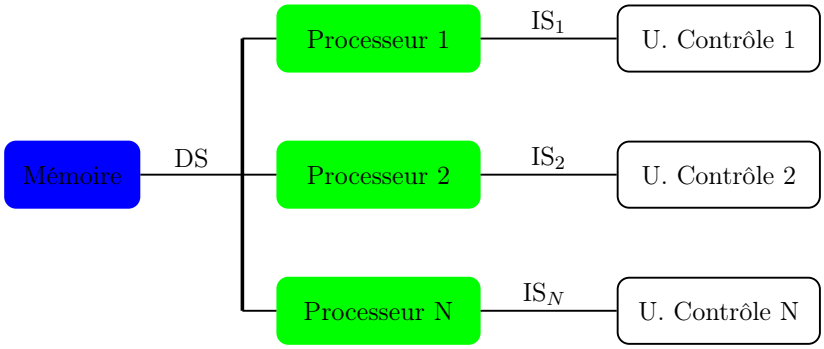
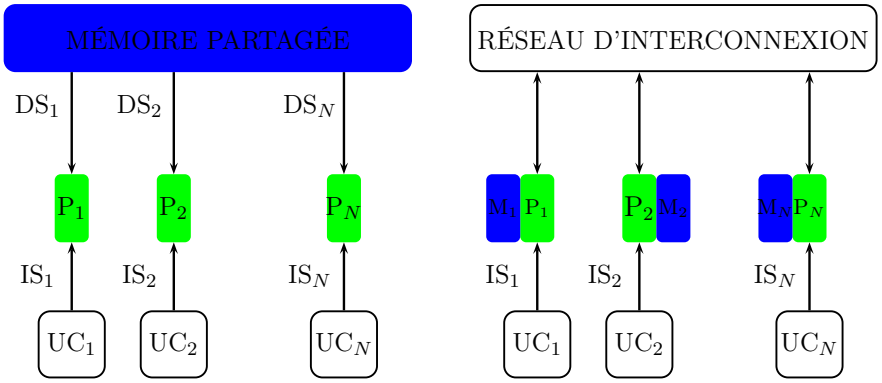


FIGURE 1.4 – Architecture MISD (Multiple Instruction Single Data)

blocs, chacun ayant sa propre unité centrale.

L'architecture MIMD (Multiple Instruction Multiple Data) est l'architecture parallèle la plus utilisée. Elle est constituée de plusieurs processeurs et plusieurs modules de mémoires reliés entre eux par le réseau d'interconnexion. Cette architecture contient plusieurs unités de contrôle et chacune peut gérer une ou plusieurs unités de traitement. À tout moment, elle permet à différents processeurs d'exécuter des instructions différentes sur différentes données.

Dans l'architecture de MIMD, le modèle de programmation SPMD (voir Figure 1.6) (Single Program Multiple Data en anglais) est le plus commun pour la programmation. Un seul programme doit être écrit et sera exécuté par tous les processeurs. Pendant l'exécution, les processeurs travaillent sur des données différentes et il faut ainsi écrire les parties correspondantes aux émetteurs et celles correspondantes aux récepteurs des communications au sein d'un même programme.



(a) MIMD avec mémoire partagée (b) MIMD avec mémoire distribuée

FIGURE 1.5 – Architecture MIMD (Multiple Instruction Multiple Data)

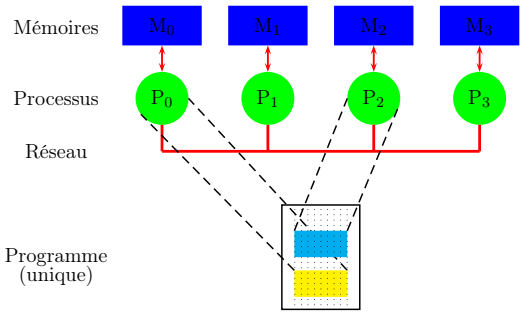


FIGURE 1.6 – Modèle de programmation de SPMD

1.2.3 Classification des Architectures parallèles selon la mémoire

La taxonomie de Flynn ne caractérise pas l'organisation de ma mémoire vis-à-vis des processeurs. Or la réalisation de mémoires rapides de grandes tailles est nécessaire pour les architectures parallèles dédiées à la haute performance.

Il existe deux grands types des machines parallèles : les machines à mémoire partagée et les machines à mémoire distribuée.

1.2.3.1 Machines à mémoire partagée

Les machines à mémoire partagée permettent de réaliser le parallélisme de données et de contrôle. Le programmeur n'a pas besoin de spécifier la distribution des données sur chaque processeur. Il définit seulement la partie du programme qui doit être parallélisée (directives) et doit gérer les synchronisations.

Dans une machine à mémoire partagée, les données d'une application sont toutes placées dans la mémoire commune, un processus p placé sur un processeur P a ainsi la possibilité physique d'accéder aux données d'un autre processus q placé sur un autre processeur Q. Cette propriété est très utile car elle permet aux processus d'échanger des informations à travers des variables partagées. Les processeurs

communiquent entre eux via la mémoire globale et par conséquent le temps de communication entre deux processeurs différents ne dépend pas de la position du processeur dans le réseau. Chaque processeur peut avoir une mémoire locale suffisamment grande pour le traitement local et le stockage temporaire. La cohérence entre ces mémoires locales devra être gérée par le hardware, le software et parfois l'utilisateur.

En se basant le temps d'accès à la mémoire, il existe deux classes principales d'architectures parallèles à mémoire partagée. La première classe, telle que les machines dual/quad core x86 (PowerPC, SPARC, etc), se base sur l'organisation de type UMA (Uniform Memory Access en anglais). Cette architecture est constituée de plusieurs processeurs identiques connectés à une unique mémoire physique (de l'ordre de 2 à 4 processeurs). Tous les processeurs accèdent à n'importe quel point de la mémoire en un temps uniforme. Cette architecture est implémentée pour les machines SMP (Symmetric Multi processors en anglais). La deuxième concerne sur l'organisation de type NUMA (Non-Uniform Memory Access en anglais) dans laquelle l'architecture parallèle est constituée de plusieurs processeurs connectés à plusieurs mémoires distinctes s'appelant bancs de mémoires. Ces processeurs sont reliées entre elles par des mécanismes matériels (jusqu'à 2048 processeurs). Le temps d'accès à la mémoire globale n'est pas uniforme, il dépend du placement des données dans celle-ci.

Le point fort des machines parallèles à mémoire partagée est la simplification au niveau de programmation et le partage des données entre les tâches est aussi rapide. La difficulté principale de cette architecture a d'accès à la mémoire (conflits d'accès à la mémoire) qui limite le nombre processeurs. Le programmeur est responsable de la validité des synchronisations À partir de quelque dizaine de processeurs, les performances commencent à se dégrader.

1.2.3.2 Machines à mémoire distribuée

Dans la machine à mémoire distribuée, chaque processeur possèdes sa propre mémoire locale et il n'a pas accès à celle des autres. Il exécute ses instructions sur ses données. L'accès à la mémoire du processeur voisin se fait par l'échange de messages à travers le réseau. Pour cela, les algorithmes utilisés sont implémentés de telle manière de minimiser les coûts de communications.

Ces communications entre les processeurs sont réalisées par l'appel à des fonctions de bibliothèque standard : PVM (Parallel Virtual Machine) ou MPI (Message Passing Interface) via un réseau d'interconnexion qui relie physiquement les processeurs entre eux. Le coût de communication entre deux processeurs différents dépend de la position de ces deux processeurs dans le réseau.

La machine à mémoire distribuée a un grand potentiel de puissance de calcul parce qu'il n'y a guère de limitation pratique au nombre de processeurs. Chaque processeur a aussi un accès rapide à sa propre mémoire De plus, ce type de machine peut être réalisé via la conception de grappe des machines (cluster en anglais) et

ensuite de grille de ces machines.

Le premier point faible de la machine à mémoire distribuée est qu'elle est plus difficile à programmer que la machine à mémoire partagée. Le programmeur devra gérer lui-même les communications. Cela pourrait engendrer des risques d'erreurs. La complexité des algorithmes est source de perte de performances. Les performances seront parfois possibles qu'au prix d'un réseau d'interconnexion coûteux. L'administration du système sur une grappe des machines doit être pris en compte dans le coût global.

1.2.4 Modèles de Programmation parallèle

1.2.4.1 Modèle Data-Parallel

L'un des modèles d'algorithmique parallèle le plus simple est le modèle à parallélisme de données (data-parallel en anglais). L'idée de ce modèle consiste à subdiviser un problème global en tâches se basant sur le partitionnement des données. Chaque tâche, associée à une partie de données, effectue des mêmes opérations (voir **Figure 1.7**). Le programmeur devrait garantir l'équilibre de charge des données.

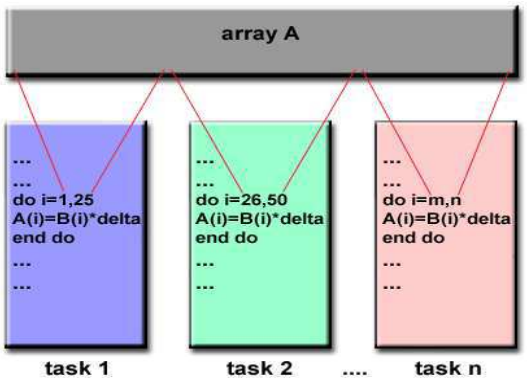


FIGURE 1.7 – Modèle de programmation de "Data-Parallel"

La programmation data-parallel consiste à rédiger un programme avec des structures de données parallèles de type tableau. Les algorithmes de ce modèle peuvent se baser sur les deux paradigmes : le paradigme de l'espace d'adresse commune et le paradigme de passage de messages. Dans un paradigme à passage de messages, la partition de l'espace d'adresse peut permettre un meilleur contrôle du placement, et peut alors offrir un meilleur manipulation de la localité. D'autre part, l'algorithme, dans le paradigme de l'espace d'adresse commune, peut être facilement programmé, surtout si la distribution des données est différente dans les différentes phases de l'algorithme (voir [Grama 2003]).

Ce modèle s'adapte bien aux problèmes dont le type de données est un tableau d'éléments, tels que des matrices, des images, etc...

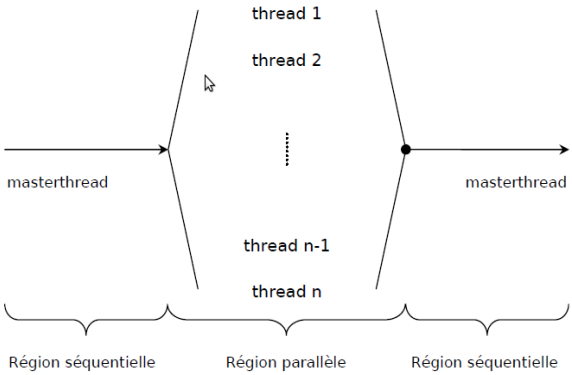


FIGURE 1.8 – Modèle de programmation à mémoire partagée

1.2.4.2 Modèle concurrent à mémoire partagée

Le modèle de programmation parallèle à mémoire partagé est le modèle le plus simple à comprendre parce qu'il est similaire à la programmation des systèmes d'exploitation. La programmation dans ce modèle se fait à travers des extensions aux langages de programmation existants, les systèmes d'exploitation, et les bibliothèques. Ils consistes à trois types principaux de routines qui permet :

1. la création de la tâche ;
2. la communication entre les tâches ;
3. la synchronisation.

Dans le modèle concurrent à mémoire partagée, les tâches se partagent un espace d'adressage commune dans lequel elles peuvent lire et écrire de manière asynchrone (voir **Figure 1.8**). Quelques mécanismes, tels que le blocage ou des sémaphores, peuvent être utilisés pour le contrôle d'accès à l'espace d'adressage commune.

1.2.4.3 Modèle concurrent à passage de messages

Le modèle de programmation à passage de messages a été conçu pour les architectures parallèles ne possédant pas de mémoire partagée. L'absence de mémoire partagée engendre deux conséquences importantes pour le programmeur. Dans ce modèle (voir **Figure 1.9**), la mémoire est fragmentée et le programme n'a pas une vision complète de l'espace mémoire. Cette répartition des données sur les différents processeurs doit être gérés par le programmeur. Chaque processus exécute éventuellement des parties différentes d'un programme. Toutes les variables du programme sont privées et résident dans la mémoire locale de chaque processus.

Les processus communiquent par envois explicites de messages contenant des données calculés par certains et utiles à d'autres. Les messages contiennent en général des données.

À la base, un système d'échange de message consiste en :

– un schéma d'adressage qui repère les processus par rapport aux autres.

– deux primitives de base(par exemple send/receive)

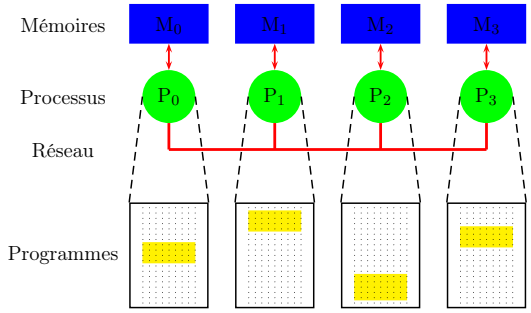


FIGURE 1.9 – Modèle de programmation à passage de messages

1.2.4.4 Modèle Multi-Threads

Dans le modèle multi-threads, un modèle très proche du modèle à mémoire partagée présenté dans **Section 1.2.4.2**, un processus est définie comme un programme en cours d'exécution d'un séquence des instructions. Il peut être décomposé en ce qui s'appelle threads, un fil d'instructions à l'intérieur d'un processus. Les threads d'un même processus se partagent l'espace d'adressage du processus. Ils possèdent leur propre compteur ordinal, leur propre pile et sont ordonnancés par le système d'exploitation concurrent. les threads ont aussi un cycle de vie semblable à celui d'un processus.

Ce modèle s'adapte bien à la programmation du côté serveur pour des applications client-serveur. Dans ces applications, un serveur doit simultanément répondre aux requêtes de multiples clients. De plus, le nombre de clients varie dans le temps. C'est-à-dire que les clients se connectent au serveur et se déconnectent dynamiquement. Dans certains cas, les programmes utilisant des threads sont plus rapides sur les machines comportant plusieurs processeurs. En plus, le partage de certaines ressources entre les threads permet une communication plus efficace entre les différents threads d'un processus.

La programmation utilisant des threads est toutefois plus difficile, l'accès à certaines ressources partagées doit être restreint par le programme lui-même. De plus, elle repose des questions concernées sur les temps de création de processus et de commutation de contexte. Ensuite, une exigence nécessaire, pour la partage de données communes entre les processus, n'est pas assurée. Cela est très importante dans plusieurs d'applications. Enfin, le programmeur n'a pas de moyen efficace pour gérer l'ordonnancement des processus.

1.2.5 Exemples de librairies facilitant la parallélisation

1.2.5.1 MPI (Message Passing Interface)

En juin 1994, le forum MPI (Message Passing Interface) définit un ensemble de sous-programmes de la bibliothèque d'échange de messages MPI. Une quarantaine

d'organisations y participent. Cette norme, MPI-1 (Le cœur de la bibliothèque), vise à la fois la portabilité et la garantie de bonnes performances. En juillet 1997, la norme MPI-2 est publiée pour l'extension et le support C++ et f90. Elle permet la gestion dynamique de processus, la copie mémoire à mémoire, la définition de types dérivés et MPI-IO.

Une application MPI est un ensemble de processus qui exécutent des instructions indépendamment les unes des autres et qui communiquent via l'appel à des procédures de la bibliothèque MPI. MPI nous permet de gérer l'environnement d'exécutions, les communications point à point, les communications collectives, les types de données dérivées, la topologie d'interconnexion des processus (grilles, arbres,...), et les groupes de processus et les groupes de communicateurs.

Parmi l'ensemble des implémentations de MPI, le trois grandes distributions les plus populaires sont :

- MPICH (http://www.mcs.anl.gov/research/projects/mpich2/)
- Open-MPI (http://www.open-mpi.org/)
- LAM-MPI (http://www.lam-mpi.org/)

1.2.5.2 OpenMP (Machines à mémoire partagée)

OpenMP (http://openmp.org/wp/) amène aujourd'hui une interface standard de haut niveau pour une programmation parallèle de type SPMD (Single Program Multiple Data) sur les machines à mémoire partagée ou au moins virtuellement partagée. OpenMP se base sur les techniques du multi-threading, elle est considérée comme l'un des grands standards au service du calcul scientifique.

OpenMP peut être supportée par plusieurs langages (C, C++ et Fortran) et disponible sur plusieurs plate-formes (Linux, Windows, OS X, ...). Les standards d'OpenMP datent de 1997, ceux d'OpenMP2 de 2000. OpenMP est un ensemble de procédures et de directives de compilation identifiées par un mot clef initial $OMP visant à réduire le temps de restitution lors de l'exécution d'un programme sans en changer la sémantique. Elle est aussi utilisée afin de paralléliser un code sur une architecture SMP (Shared Memory Processors en anglais).

1.2.6 Conclusion

Dans le cadre de ce mémoire, le scénario d'utilisation pourrait être celui d'un hydrologue qui réalise des calculs sur sa propre grappe ou bien sur une grappe distante quand les travaux sont trop lourds ou que celui-ci ne dispose que d'une machine séquentielle de bureau. Le choix d'une grappe est celui de la simplicité et de l'économie. Les machines qui la composent peuvent être relativement standards. Toutefois nous aurions pu aller jusqu'à choisir une machine SMP puissante comme il en existe de plus en plus sur le marché. La programmation en aurait été simplifiée et le coût amoindri. Toutefois nous avons choisit d'orienter nos travaux vers les grappes de machines (machines parallèles à mémoire distribuée). Il est apparu que cette architecture est meilleure candidate eu égard à ses qualités d'extensibilité.

Compte tenu du fait que les tâches peuvent être modestement grandes et lancées en local ou bien très lourdes, portant sur de vastes données, et lancées à distance sur un centre de calcul, il importe donc que la puissance de la machine puisse être étendu en fonction des besoins sans que le modèle de programmation ni les algorithmes ne soient remis en cause.

Nous complétons le choix de cette architecture par l'utilisation de la bibliothèque MPI qui est devenue un standard et dont les implémentations sont efficaces et largement répandues.

Chapitre 2

Analyse Hydrologique des Terrains

Sommaire

2.1 Modélisation de l'écoulement de l'eau 28
2.1.1 Notion de pente 28
2.1.2 Modification d'homotopie 29
2.2 Méthodes Séquentielles 30
2.2.1 Mono direction d'écoulements 31
2.2.2 Multiples directions d'écoulements 33
2.2.3 Plate-forme TerraFlow et TerraStream 35
2.3 Parallélisation de modèles de directions d'écoulements . . . 38
2.4 Conclusion 39

Le *Système d'Information Géographique* (en abrégé SIG) est un système informatique permettant, à partir de diverses sources numériques, de rassembler, d'organiser, de gérer, d'analyser et de combiner, d'élaborer et de présenter des informations localisées géographiquement, contribuant notamment à la gestion de l'espace. L'information géographique se définit comme l'ensemble des objets à référence spatiale dont une caractéristique essentielle est l'aspect multi-source et multimédia. La représentation des objets spatiaux utilise des concepts issus de la géométrique euclidienne pour la localisation absolue des objets, mais aussi de la théorie des graphes et la topologie afin d'étudier leurs positions relatives et leurs relations spatiales. Il existe principalement deux types de représentations : *vecteur* et *raster*.

1. Type de vecteur : les limites des objets spatiaux sont décrites à travers leurs constituants élémentaires, tels que les points, les arcs et les polygones. Chaque objet spatial est doté d'un identifiant qui permet de le relier à une table attributaire.
2. Type de raster : la réalité est décomposée en une grille régulière et rectangulaire, organisée en lignes et en colonnes, chaque maille de cette grille ayant une intensité de gris ou une couleur. La juxtaposition des points recrée l'apparence visuelle d'une carte. Par exemple une forêt sera "représentée" par un ensemble de points d'intensité identique.

Les analyses à partir d'un relief topographique se basent sur ce qui s'appelle un *Modèle Numérique de Terrain* (en abrégé *MNT*) à travers la notion d'altitude. C'est

une source d'information nécessaire à la visualisation, l'analyse et la modélisation de phénomènes reliés au relief et en particulier l'extraction des paramètres du relief. Le MNT est une expression numérique de la topographie, sous forme matricielle ou vectorielle. Il peut être sauvegardé sous l'une des trois structures de données suivantes : une grille rectangulaire, une structure de découpage à base d'éléments irréguliers ou *TIN* (*Triangulated Irregular Network* en anglais), ou bien une structure basée sur des contours. La structure de grille rectangulaire pour un MNT est la plus utilisée dans la plupart des applications (voir [Moore 1991, Wise 1998]). C'est à grâce à cette richesse que les MNT sont utilisés dans de multiples domaines nécessitant une connaissance approfondie du relief, tels que la visualisation de fins (appliquée dans le cadre militaire et dans les opérations du génie civil), l'analyse et la modélisation des phénomènes liés au relief topographique (le microclimat, la circulation des vents locaux, l'urbanisme, la prédiction de la crue et surtout l'hydrologie).

Les MNT sont utilisés dans de multiples domaines, tels que le tourisme, les transports, les télécoms,... ainsi qu'en hydrologie.

L'*Hydrologie* est une science de la terre qui s'intéresse aux cycles de l'eau. À ce titre elle s'intéresse aux échanges hydriques entre l'atmosphère, la surface et le sous-terrain. Une des branches de cette science focalise sur l'analyse du parcours de l'eau en surface, c'est *l'hydrologie de surface* qui permet par exemple de caractériser le ruissellement, l'érosion, la position des cours d'eau ou les inondations. Ces phénomènes jouent un rôle important dans la gestion des ressources et l'aménagement du territoire. Bien sûr, lors d'analyses complètes et fines, il est difficile de dissocier ces phénomènes de surface de leurs liens avec l'atmosphère et le sous-sol. Mais la réduction de leur étude à des aspects purement et seulement liés à la topographie apporte une vision peut-être schématique mais déjà importante pour l'hydrologue. C'est particulièrement vrai pour les vaste étendues à l'échelle d'un continent où il est aisé d'obtenir par imagerie satellite des informations de surface relativement précises mais où il est difficile de réunir des informations de sous-sol pertinentes à cette échelle.

Un des aspects les plus importants dans la gestion et l'exploitation des ressources en eau est le réseau hydrographique des bassins versants. Le terme désigne aussi l'ensemble des cours d'eau d'une région donnée, organisés en bassins hydrographiques (ou bassins versants). Un bassin versant est une zone associée à un minimum telle qu'une goutte d'eau tombant en un point de cette zone descende au final jusqu'au minimum. À partir de cette définition, un bassin versant est comme la totalité de la surface topographique drainée par un cours d'eau. Il est entièrement caractérisé par son exutoire, à partir duquel nous pouvons tracer le point de départ et d'arrivée de la ligne de partage des eaux (LPE) qui le délimite.

La cartographie du réseau hydrographique à partir d'un MNT a un double objectif : le premier est la détermination des descripteurs de la topographie (pentes, crêtes, vallées, etc.) des bassins versants et de leurs paramètres hydrologiques afin de cartographier les zones de récupération des ruissellements et le deuxième est la modélisation synthétique de la géomorphologie des bassins versants pour une meilleure connaissance de l'occupation du sol (voir [Nanée 2004]). Cette modélisation permet

de prévoir la réponse des bassins versants à des évènements pluvieux et de connaître l'évolution au cours du temps de ses caractéristiques des écoulements. En outre le réseau hydrographique constitue un indicateur importants afin de comprendre en fonction de la topographie, le parcours des carbones, des nutriments et des sédiments.

Nous avons vu que la détermination des bassins versant est une phase importante de l'analyse hydrologique. Elle est implémentée dans la plupart des SIG. Mais ce type d'analyse trouve aussi d'autres applications telles que l'analyse de bassin visuel. Cette méthode d'analyse spatiale vise à déterminer des endroits visibles à partir d'un point d'observation en se basant sur la ligne de mire. Cette analyse nous donnera un carte binaire : visible et invisible (voir Figure 2.1). Elle est largement utilisée dans les applications pratiques. Elle permet d'analyser l'impact visuel d'un nouveau bâtiment ou d'un développement urbain ou industriel(voir [Fisher 1996]), ainsi qu'à localiser le meilleur site pour une tour d'observation par exemple pour les incendies dans la forêts (voir [Lee 1991]) ou les relais de communications (voir [Floriani 1994]). Bien entendu la qualité de l'analyse de bassin visuel dépend du modèle de données utilisé (DEM ou TIN) et de la précision des données d'élévation.

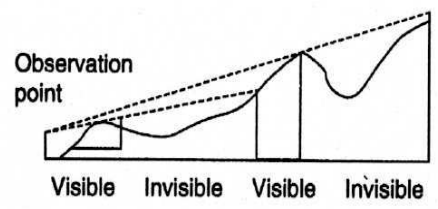


FIGURE 2.1 – Analyse de bassin visuel basée sur la ligne de mire

Si ce type d'analyse se trouve aussi implanté dans les SIG, dans ce document nous nous concentrerons sur l'aspect hydrologique des bassins versants.

La majorité des méthodes se basent sur le concept de simulation d'écoulements d'eau (voir [O'callaghan 1984, Jenson 1988, Martz 1988, Morris 1988]) sur un relief topographique en utilisant un *Modèle Numérique de Terrain* (en abrégé MNT) comme entrée pour ce processus et en faisant l'hypothèse que des fleuves s'écoulent plus probablement dans des thalwegs et ne traversent pas les crêtes. Trois étapes sont classiquement utilisées (voir [Jenson 1988]). Ce sont des fonctionnalités implantées dans les *Systèmes d'Information Géographique* courants tels que *GRASS* ou *ARCGIS*.

Étape 1 : Définition d'un raster en modélisant l'écoulement d'eau sur chaque pixel dans le MNT. Cela produit une forêt d'arbres disjoints dont la racine est un minimum local.

Étape 2 : Définition d'un autre raster afin de coder des flots d'accumulations. Les flots d'accumulations représentent en chaque point p d'un terrain le nombre de points dont l'eau, si elle tombe dessus, passe par p avant de rejoindre un exutoire

global. Une valeur forte de flot d'accumulation en un point est le signe du passage probable d'un cours d'eau en ce point.

Étape 3 : Segmentation des pixels appartenant au réseau hydrographique probable et des autres, en fixant un seuil sur la valeur des flots d'accumulation.

Dans le cadre de cette thèse, nous nous concentrons sur l'analyse de la ligne de partage des eaux à partir d'un MNT qui conditionne la détermination des bassins versant ainsi que sur le calcul des flots d'accumulation qui permet d'extraire le réseau hydrographique probable. Ces deux points sont à la base de nombreuses analyses dans les SIG en hydrologie. En outre ces calculs dépendent de facteurs non locaux en chaque points du MNT ce qui rend leurs durées de traitement extrêmement sensibles à la taille totale des MNT sur lesquels elles portent.

2.1 Modélisation de l'écoulement de l'eau

L'hydrologie s'intéresse particulièrement aux problèmes de pentes (*slope* en anglais) et d'orientation où la trajectoire d'écoulements est localement déterminée à la ligne de plus grande pente. Ce paramètre, la pente, est ainsi une des données fondamentales à partir de laquelle le réseau hydrographique est extrait. La pente S mesure le taux de changement d'élévation dans les différentes directions. Fondée sur la la gravité, une hypothèse classique en géomorphologie consiste à privilégier les chemins descendants les plus raides afin de déterminer le parcours de l'eau sur un relief topographique. On en déduit une hydrographie probable.

La pente peut être calculée en utilisant les directions des chemins descendants les plus raides vers un des quatre voisins (4-connectivité, les voisins cardinaux seuls) ou les huit voisins (8-connectivité, voisins diagonaux pris en compte).

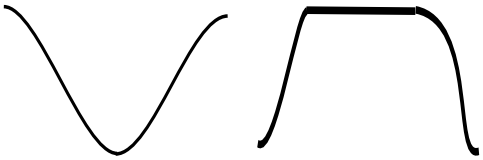
2.1.1 Notion de pente

2.1.1.1 Extraction des paramètres primaires

L'analyse de terrains, à partir d'un relief représenté par un MNT, commence par le calcul des dérivées. Les dérivées d'un relief influencent directement le calcul des directions d'écoulements d'eau. Elles permettent de définir les cartes de pentes et azimuts (paramètres primaires) ou les cartes de courbures (paramètres secondaires). Mais il existe d'autres calculs plus complexes permettant de construire des cartes hydrographiques incluants des notions telles que les crêtes, les talwegs, les bassins versants (voir [Martz 1988, O'callaghan 1984]) ou encore le réseau hydrographique (voir [Fairfield 1991]).

2.1.1.2 Extraction des paramètres secondaires : Courbure d'une surface

Les applications de SIG calculent souvent la courbure d'une surface en chaque cellule. Ce type de calcul consiste à déterminer si cette surface est convexe ou concave en se basant sur la courbure profile (la courbure d'une surface dans la direction de



(a) Puits (b) zone plate ou plateau

FIGURE 2.2 – Exemple de bruits (puits et plateau) dans un MNT

la pente) et la courbure de plan (la courbure d'une surface perpendiculaire à la direction de la pente). La courbure profile mesure le taux de changement du gradient potentiel, c'est un point important pour la caractérisation des changements dans la vitesse d'écoulement et dans les processus de transport de sédiments. Cette valeur est négative pour une pente qui accélère (correspondant à la surface convexe) et est positive pour une pente décroissante (correspondant à la surface concave).La courbure de plan mesure la convergence et la divergence topographiques. Cette valeur est négative pour les flux divergents (sur les crêtes) et positif pour les flux convergents (dans les vallées).

2.1.2 Modification d'homotopie

La première étape consiste à modifier le MNT d'entrée afin d'éliminer le bruit, les erreurs et les incohérences topographiques [Chorowicz 1992, Tribe 1992]. Ce mécanisme constitue ce que l'on désigne par *modification d'homotopie*.

Un point important est que les MNT comportent une multitude de dépressions dans les données représentées. Il s'agit de les cuvettes (ou puits) d'où l'eau ne peut pas sortir vers l'exutoire global. Les puits sont formés par une cellule ou un groupe de cellules adjacentes entourés des cellules plus hautes (voir **Figure 2.2(a)**). Il existe aussi des zones d'absence de données : les trous. Il y aussi les zones plates (ou plateaux) qui sont formées par au moins de deux pixels ayant la même hauteur (voir **Figure 2.2(b)**). Tous ces artefacts perturbent potentiellement les algorithmes de détermination des bassins versants ou de LPE. La présence de ces zones pose un problème de discontinuité du réseau hydrologique : l'eau stagne dans les zones de dépression et se disperse dans les plateaux (voir [Martz 1998]). C'est une des grandes difficultés pour la définition des directions d'écoulements sur des grilles en raster. En conséquence, les auteurs proposent souvent de les supprimer lors d'un pré-traitement par exemple en comblant les trous, les cuvettes et les plateaux. Toutefois, ce type de traitement peut introduire de nouvelles cuvettes, ce processus est alors itéré jusqu'à ce que tous les bruits soient traités ce qui peut être coûteux.

Afin de résoudre ce problème, il existe une méthode manuelle et un grand nombre d'algorithmes automatiques. L'avantage de la méthode manuelle est d'éviter les artefacts de mise en œuvre lors de l'ajustement des directions d'écoulements pour des pixels appartenant aux dépressions (puits, fosses,..). Ce processus est très coûteux en temps lorsqu'il existe un grand nombre de dépressions dans la grille en raster. De

plus, des résultats d'écoulements sont frappés d'empirisme.

L'autre voie, qui est majoritaire, tente de résoudre les puits en essayant de les éliminer dans le MNT. C'est réalisé en ajustant les valeurs dans une grille de telle manière qu'il existe toujours un chemin descendant pour toutes les cellules. L'idée consiste à augmenter la hauteur des cellules appartenant aux puits (comblement des puits ou comblement des dépressions), ou à réduire la hauteur des pixels au bord des puits (la technique de "carving" ou de "breaching" en anglais), ou à combiner les deux techniques. Ces techniques sont à rapprocher des techniques similaires en analyse d'image et qui tendent à obtenir des images lower complete.

La première méthode proposée consiste à appliquer un filtre passe-bas pour le lissage d'un MNT, par exemple la moyenne (voir [Mark 1984]) ; cette méthode ne peut pas éliminer tous les puits dans un MNT, pire, elle peut aussi ajouter de nouveaux puits dans les vallées étroites. (voir [Grimaldi 2007]).

D'autres méthodes essayent de réduire la présence des puits par des pré-traitements des MNT en utilisant une information vectorielle contenant l'information des drainages hydrographiques. Cette opération peut être utilisée pendant l'interpolation ou la création des grilles MNT (voir [Hutchinson 1989]). L'idée commune est de changer de MNT en utilisant un drainage appelé *"Stream Burning"* (voir [Saunders 1999, Maidment 2002]). Le MNT est "creusé" en fonction du drainage dans l'espoir de forcer le trajet de l'eau là où elle est censée passer. Remarquons que cette méthode est seulement appliquée lorsque nous avons eu un vecteur de données hydrographiques et que ces données doivent être compatibles aussi avec le MNT en cours de traitement. Cette méthode ne peut pas complètement éliminer la présence des puits dans un MNT. La méthode de comblements des puits a été développée dans [Jenson 1988] ; l'idée consiste simplement à augmenter la hauteur des cellules dans un puits. Cela est réalisé jusqu'à ce que toutes les dépressions soient comblées et qu'il existe toujours un chemin descendant dans le réseau hydrographique pour chacune des cellules dans un MNT. Elle a aussi été implémentée dans des outils de comblement de ESRI ArcGIS (`http://www.esri.com/software/arcgis/index.html`). Une autre méthode a aussi été proposée par Planchon et Darboux (voir [Planchon 2002]). L'idée consiste à inonder toute la surface d'un MNT en assignant un niveau d'eau plus haut pour chaque cellule dans le MNT et à supprimer la quantité d'eau superflue à chaque cellule. Cette approche a été implémentée dans la plate-forme Terrain Analysis System [Lindsay 2005] (en abrégé TAS `http://www.uoguelph.ca/~hydrogeo/TAS/index.html`). Dans [Jones 2002], l'auteur a proposé une méthode d'élimination des puits en utilisant l'algorithme de PFS (priority-first-search en anglais) afin de chercher un exutoire pour chaque puits et ensuite ajuster la hauteur des cellules appartenant au chemin à partir de l'intérieur des puits vers cet exutoire.

2.2 Méthodes Séquentielles

Une des premières étapes pour inférer des informations en hydrologie à partir des MNT est de définir des directions d'écoulements de chaque cellule en utilisant un

modèle. Les directions d'écoulements définies en tout endroit du terrain sont utilisées afin de modéliser un chemin global de l'eau, des sédiments, des contaminants, des nutriments s'écoulant sur un relief. La direction d'écoulement est un paramètre clé dans la topographie des bassins versants. À partir de cela, quelques attributs peuvent facilement être extraits, tels que la ligne de crête, la zone de drainage ou encore le réseau hydrographique. Depuis les années de 1990, plusieurs modèles ont été proposés pour ce type de calculs. Ils sont classés en deux types de méthodes : mono direction d'écoulements (Single Flow Direction en anglais et en abrégé SFD) et multiples directions d'écoulements (Multi Flow Directions en anglais et en abrégé MFD).

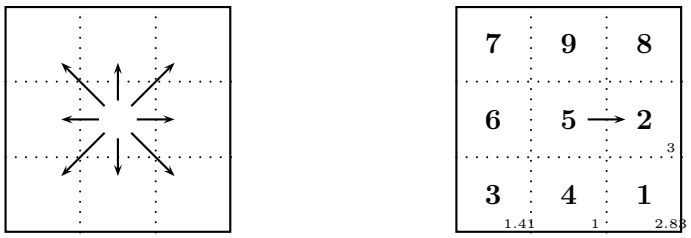
2.2.1 Mono direction d'écoulements

La méthode mono direction d'écoulements permet à l'eau arrivant sur un pixel p dans un relief de ne s'écouler que vers un seul voisin q plus bas que p. Le choix de q dépend des modèles mis en œuvre, par exemple, un voisin q en suivant la plus grande pente pour le modèle D8 [O'callaghan 1984] (Deterministic Eight-neighbor) ou un voisin plus bas aléatoire pour le modèle Rho8 (Random eight-node en anglais) (voir [Fairfield 1991]). Elle implémente seulement des directions d'écoulements primaires pour l'emplacement de l'eau sur un relief en considérant approximativement une surface comme un point ; un tube de 2D comme une ligne (voir [Moore 1996]). Cette méthode a aussi été largement utilisée par d'autres auteurs (voir [Band 1986, Jenson 1988, Mark 1988, Marks 1984, Martz 1992, Morris 1988, Tarboton 1988]). L'approche mono direction d'écoulements produit un résultat satisfaisant sur des surfaces concaves dans lesquelles les écoulements de l'eau convergent. De plus, les calculs de détermination des flots accumulations sont simples et rapides pour les grands MNT.

Le modèle D8 est le plus simple, il a été introduit par O'Callaghan et Mark (voir [O'callaghan 1984]). Ce modèle est aussi implémenté dans la plupart des plate-formes de SIG grâce à ses simplifications et surtout à son efficacité en terme de calcul. La direction d'écoulements principale d'un pixel p est déterminée en choisissant le voisin q plus bas que p dont la pente pour l'atteindre est la plus grande (voir **Figure 2.3(b)**). Moore nous propose la formule ci-dessous (voir [Moore 1996]) pour ce calcul. Les flots d'accumulations de chaque pixel p du MNT sont ensuite déterminés par le nombre de pixels qui sont drainés vers p (voir [Jenson 1988]). Les pixels dont les flots d'accumulations sont égaux à zéro appartiennent à la ligne de partage des Eaux. Un seuil pré-défini par l'utilisateur est choisi afin d'extraire le réseau hydrographique. Il concerne les pixels dont des flots d'accumulations sont plus grands que ce seuil.

$$f = 2^{J-1} \tag{2.1}$$

où :
J = i tel que :



(a) Directions d'écoulements possibles

(b) Direction d'écoulements au pixel central

FIGURE 2.3 – Modèle D8 (Deterministic Eight-neighbor en anglais)

$$\max_{i=1\to 8}\{\phi(i)\left\|\frac{z_9 - z_i}{\lambda}\right\|\} \tag{2.2}$$

et

- $\phi(i) = 1$ pour les voisins cardinaux, $\phi(i) = \frac{1}{\sqrt{2}}$ pour les voisins diagonaux.
- λ est la taille d'un pixel. Elle est définie dans l'entête d'un MNT
- z sont des élévations correspondant aux 9 pixels dans une sous-matrice 3×3 (z_1, z_2,... ,z_9).

À cause du choix un voisin plus bas pour la direction d'écoulement, Cette algorithme produit souvent beaucoup de cellules dont les flots d'accumulation sont égaux à zéro et qui ne correspondent pas à la LPE en réalité. De plus, sur les zones topographiques divergentes ou planaires, ce modèle simplifie les directions d'écoulements et alors produit des directions d'écoulements parallèles. En réalité, ces directions d'écoulement devraient être considérées comme de la dispersion.

Un autre modèle SFD, le Rho8, a été proposé par Fairfield et Leymarie (voir [Fairfield 1991]) en se basant sur des choix stochastiques pour les directions d'écoulements. Le choix dans Rho8 se fait parmi les voisins les plus bas suivant une loi aléatoire. Ce modèle fonctionne quasiment comme le modèle D8 et commence d'abord à identifier tous les voisins plus bas d'un pixel considéré. Il calculera ensuite la pente de gradients entre chacune des directions et assigne la direction d'écoulements d'un pixel vers ces pixels candidats en utilisant des nombres stochastiques qui ont été calculés. Les flots d'accumulations de tous les pixels dans le MNT sont déterminés comme le modèle D8. Cependant, pour un même MNT, des réseaux de directions d'écoulements différents seront produits à chaque fois que l'algorithme de Rho8 sera réalisé. Le modèle *Rho8* est à même de résoudre le problème des flux parallèles sur des surfaces planaires existant dans le modèle D8. Le grave défaut de ce modèle est qu'il est intrinsèquement impossible de reproduire ses résultats à partir d'un même jeu de données.

Il existe encore un modèle de mono direction d'écoulement proposé par Léa (voir [Lea 1992]). Ce modèle est basé sur un vecteur local d'azimuts (*aspect vector* en anglais) dans la direction descendant et sur une surface associée à chaque cellule

dans le MNT afin de calculer la direction d'écoulements. Intuitivement, le flux associé à une cellule c suit la course d'un ballon roulant sur cette surface à partir du centre de la cellule. Cette surface est construite en déterminant l'élévation des quatre coins de la cellule. Ces élévation sont obtenues en interpolant les élévations des pixels voisins de c. Ce modèle permet de distribuer, à une cellule c dans une grille, une quantité de flux (direction d'écoulement) dont la valeur est mesurée comme un angle variant continuellement de 0 à 2π. Ce système est plus fin que les précédents mais à très grande échelle il n'est pas certain qu'il présente une amélioration importante. Par contre il est plus coûteux en terme de calcul.

2.2.2 Multiples directions d'écoulements

L'approche par multiples directions d'écoulements a également été suggérée dans certaines études (voir [Freeman 1991, Quinn 1991, Holmgren 1994, Pilesjo 1996, Tarboton 1997]) afin de résoudre des problèmes de la méthode SFD. L'idée consiste à déterminer, à chaque cellule, la distribution d'une quantité de flux vers tous ses voisins plus bas en fonction de ses pentes. Le point fort de cette méthode à flux multiples est sa capacité à analyser simultanément des directions d'écoulements (ou flux) parallèles, convergents et divergents. Elle est bien adaptée et pour analyser les terrains comportant des surfaces de type plateaux et des surfaces convexes.

Le premier modèle, appelé FD8 (Fractional Deterministic Eight-Neighbor en anglais), a été proposé et développé par Quinn (voir [Quinn 1991]). Afin de calculer les flots d'accumulations à chaque pixel, une quantité d'eau d'un pixel p se distribue proportionnellement vers ses voisins plus bas. Cette proportion est calculée en fonction de la pente en utilisant la formule ci-dessous :

$$f_i = \frac{f_{total} \times \Delta(z_i)}{\sum_{j=1}^{9} \Delta(z_j)} \tag{2.3}$$

où

- $\Delta(z_i)$ est un changement d'élévation entre p et son voisin i plus bas que p. Si i est plus haut que p alors $\Delta(z_i) = 0$
- f_{total} est la totalité des flots d'accumulations au pixel p,
- f_i est la partie de f_{total} qui contribue au voisin i plus bas que p.

Le modèle FD8 peut reconnaître que l'écoulement sur la surface convexe, par exemple des versants (hillslope en anglais), est divergent, tandis que l'écoulement des pixels appartenant aux vallées est convergent. Il est à même de réduire les effets indésirables du modèle de D8 sur les zones plates comme par exemple les directions d'écoulements parallèles.

Le modèle FD8 peut toutefois lui aussi produire des écoulements indésirables (voir [Seibert 2007]) sur des versants convergents. Et afin de réduire ces effets, Holmgren (voir [Holmgren 1994]) a proposé une fonction empirique alternative en se basant sur la pente de gradients.

$$f_i = \frac{(tan\ \beta_i)^x}{\sum_{j=1}^{8}(tan\ \beta_j)^x} \tag{2.4}$$

où

- i, j = directions d'écoulements vers les huit voisins,
- f_i : proportion d'écoulements dans la direction i,
- $tan\ \beta_i$: la pente entre le pixel central et son voisin dans la direction i
- x : une variable d'exposant.

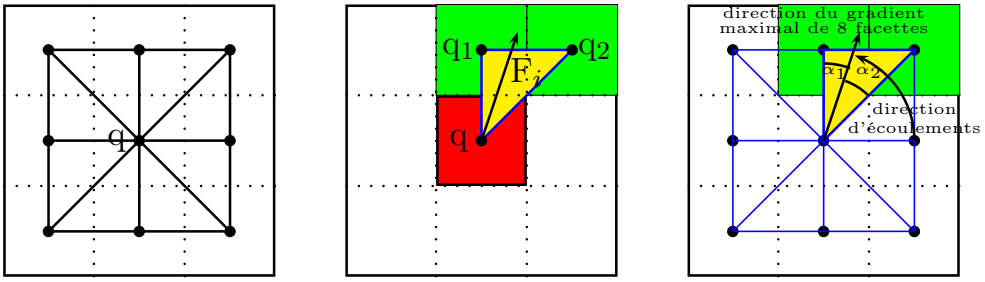
Dans l'équation 2.4, en choisissant l'exposant $x = 1$, l'équation 2.4 devient l'équation 2.3 proposée par Quinn (voir [Quinn 1991]) ; et lorsque x $\rightarrow \infty$, l'équation 2.3 modélise le modèle mono direction d'écoulements D8. En faisant varier l'exposant x depuis 1 vers ∞, la distribution d'écoulement de cette équation change progressivement de multiple en mono direction d'écoulements. Les expérimentations dans la contribution de Holmgren (voir [Holmgren 1994]) montrent que la valeur idéale d'exposant doit être choisie entre quatre et six. Les flots d'accumulations d'un pixel p dans le MNT sont obtenus en sommant des fractions des flots d'accumulations des voisins drainant vers p.

Un autre modèle D∞ (en anglais Deterministic Infinity) a été proposé par Tarboton (voir [Tarboton 1997]). Ce modèle calcule des directions d'écoulements en faisant un compromis entre plusieurs modèles précédemment proposés, par exemple D8 (voir [O'callaghan 1984]), ou bien le modèle aspect-dérivé de Léa (voir [Lea 1992]). L'idée de ce modèle consiste à associer à chaque pixel q (q correspond à la zone en rouge sur la Figure 2.4(b)) une valeur entre 0 et 2π pour le calcul des directions d'écoulements en examinant les huit facettes F_i (voir **Figure 2.4(a)** triangulaires d'un pixel q. Chaque facette F_i (voir **Figure 2.4(b)**) aura un sommet au centre de q (q en rouge) et deux autres sommets localisés au centre de deux voisins de q (deux voisins q_1 et q_2 en vert). C'est un modèle à multiples directions d'écoulements permettant d'assigner une direction d'écoulements de manière continue en fonction de la valeur d'un angle variant de 0 à 2π. De plus, l'eau d'un pixel q est seulement distribué à deux voisins maximum plus bas que q. Ceci permet d'éviter la dispersion des directions d'écoulements sur des surfaces convexes.

La **Figure 2.4(c)** illustre la manière d'assigner des directions d'écoulements dans le modèle de Tarboton. La direction d'écoulements d'un pixel q est assigné par la valeur d'un angle, noté direction d'écoulement dans la figure. Elle est déterminée en se basant sur la direction du gradient maximal parmi les huit facettes triangulaires. Ici elle est orientée vers les deux pixels colorés en vert. La proportion des écoulements vers chacun des deux pixels en vert est déterminée par deux ratios :

$$p_1 = \frac{\alpha_1}{\alpha_1 + \alpha_2}\ et\ p_2 = \frac{\alpha_2}{\alpha_1 + \alpha_2} \tag{2.5}$$

où α_i est mesure un angle.



(a) huit facettes (b) direction du gradient d'une facette F_i (c) Calcul des directions d'écoulements

FIGURE 2.4 – Modèle D∞ pour le calcul des directions d'écoulements

Les flots d'accumulations d'un pixel p sont calculés par sommation de la valeur de flot propre à p et de celles des flots d'accumulations de ses voisins v qui ont des fractions drainées vers p au prorata de la quantité d'eau de v répartie vers p. Ce type de calcul est réalisé récursivement, son principe est illustré dans **Algorithme 1**.

Algorithme 1: Calcul des flots d'accumulations dans le modèle de $D\infty$

```
1  procedure FA (pixel p)
2  begin
3      if (FA(p) n'est pas déterminé) then
4          fap = 1 ;                                  // une zone a un seul pixel
5          foreach (v est un voisinage de p) do
6              proportion = proportion de v drainant vers p basée sur l'angle ;
7              if (proportion > 0) then
8                  fav = FA(v) ;
9              end
10             fap = fap + proportion × fav ;
11         end
12     end
13     return fap;
14 end
```

Un autre modèle à multiples directions d'écoulements, appelé MFD-md, a été aussi proposé par Qin (voir [Qin 2007]). Afin de répartir le flux d'une cellule vers ses voisins plus bas, ce modèle se base sur les conditions topographiques locales au lieu du gradient maximum systématique du modèle proposé par Quinn (voir [Quinn 1991]). Afin de modéliser les flux, l'auteur a utilisé un schéma de partition de flux en choisissant un exposant variant de 1.1 à 10 pour la modélisation des flux divergents et un exposant plus grand pour la modélisation des flux convergents. Il reste que ce schéma demeure très similaire aux autres schémas déjà présentés [Freeman 1991, Holmgren 1994, Quinn 1995].

2.2.3 Plate-forme TerraFlow et TerraStream

Le projet TerraFlow (http://www.cs.duke.edu/geo*/terraflow/) offre un moyen de calcul des flots d'accumulations pour de grands terrains sur une machine séquentielle classique via des optimisations des opérations d'entrées/sorties (voir [Arge 2001c, Arge 2003a]). L'idée consiste à conserver les données initiales sans modifications et de ce concentrer sur le problème de calcul des directions d'écoulements. Les auteurs montrent que ce point de vue intuitif peut être formalisé dans un problème de routage entre les pixels. Par ailleurs, il est possible de transposer cet algorithme au niveau des pixels en un algorithme identique au niveau des cuvettes. (rappelons qu'une cuvette est un ensemble des pixels dont le chemin descendant conduit à un même pixel considéré comme un minimum local). Ce constat permet de résoudre rapidement le problème car la nouvelle structure de donnée qui manipule les cuvettes est beaucoup plus légère que le MNT initial. Les auteurs ont introduit un *graphe des cuvettes*, (watershed graph, en anglais). C'est un graphe non orienté dont les sommets correspondent aux cuvettes et dont les arrêtes représentent la connectivité entre deux cuvettes adjacentes. Les arrêtes sont pondérées par la valeur de l'altitude la plus basse sur la frontière entre les deux cuvettes.

Afin de représenter la zone extérieur au terrain, une cuvette virtuelle s'appelant la *cuvette extérieure* (outside watershed en anglais), est ajoutée dans le graphe des cuvettes. Un graphe d'écoulements entre des cuvettes est ensuite construit en tenant compte de la contiguïté et des étiquettes.Les auteurs recherchent un chemin s'écoulant vers la cuvette extérieure à partir de chaque cuvette.Cette approche peut construire des cycles qui doivent être éliminés par des calculs coûteux (voir [Jenson 1988, Peckham 1995]). Dans [Arge 2001a], il proposé une solution en simulant une inondations survenues sur un terrain comme dans la réalité lorsque le niveau de la mer monté. L'eau remplit progressivement les cuvettes et cela produit un chemin à partir de la cuvette extérieure vers les cuvettes intérieures. Cette approche est typiquement séquentielle et a été implémentée dans un logiciel appelé TerraFlow. TerraFlow a utilisé des algorithmes manipulant efficacement les mémoires externes (I/O-efficients algorithms en anglais)afin de réduire le temps de calcul des flots d'accumulations sur de gros terrains. Ce problème dans TerraFlow peut être résolu en O(N×logN) et O(Sort(N)) opérations d' entrées/sorties, où N est le nombre de pixels dans le terrain.

Cette approche est peu extensible. Pour un ensemble de données fixe, vouloir réduire le temps de calcul conduit à acheter une nouvelle machine séquentielle plus puissante. Cette approche ne tire pas partie des avantages des architectures parallèles distribuées qui sont facilement extensibles.

Le projet TerraStream (http://www.madalgo.au.dk/Trac-TerraSTREAM), améliore Terraflow en proposant de traiter le problème sous la forme d'un pipeline séquentiel. Chaque étages du pipeline ont été optimisés. Les quatre composants principaux manipulent de gros volumes de données en se basant sur des algorithmes manipulant efficacement les mémoires externes (I/O-Efficient en anglais). La **Fi-**

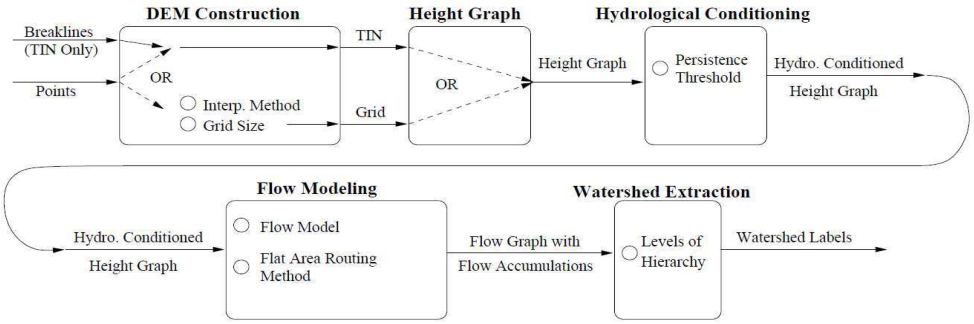


FIGURE 2.5 – Pipeline d'analyse de TerraStream à partir de nuages de points

gure 2.5 nous montre un schéma de l'organisation de ces quatre composants en pipeline de TerraStream.

Le premier composant dans TerraStream est la construction d'un MNT (voir [Agarwal 2006a, Agarwal 2005]) sous format TIN (Triangulated irregular network en anglais) ou sous format grille à partir d'un ensemble de N points $S = \{p(x, y, z) \in R^3\}$. Il vérifie la qualité de la grille du MNT en analysant la distribution de ses points. Pour la construction en grille, TerraStream utilise l'algorithme externe (I/O-Efficient en anglais) développé dans [Agarwal 2006a]. Cette construction se base sur un arbre quaternaire (quad-tree en anglais) pour la segmentation en combinant avec l'interpolation par spline régulière avec tension (voir [Mitasova 2005]). TerraStream construit une grille MNT en $O\left(\frac{N}{B} \times \frac{h}{log\frac{M}{B}} + SORT(T)\right)$ opérations de E/S. Pour la construction en format de TIN (TerraStream a choisi l'algorithme externe de type aléatoire (randomized I/O-Efficient algorithme en anglais) (voir [Agarwal 2005]) afin de construire une triangulation de Delaunay sous contrainte (voir [Chew 1989]). Cet algorithme utilise $SORT(N)$ opérations de E/S (voir [Danner 2007]) dans la plupart des applications réelles.

Dans l'analyse du MNT, afin d'éviter la construction d'algorithmes spécialisés pour manipuler les deux formats (grille et TIN), TerraStream a proposé une structure de données unique, appelée graphe de hauteur. Un autre module existant dans TerraStream nous permet de convertir un MNT sous format d'une grille ou de TIN en un graphe de hauteur avant d'analyser la LPE. À partir d'un graphe de hauteurs obtenu précédemment, la seconde phase de TerraStream réalise l'élimination de tous les puits non-signifiants en se basant sur la notion de *persistance topographique* (voir [Edelsbrunner 2000, Edelsbrunner 2001]) et sur un seuil choisi par l'utilisateur. Cet algorithme a été implémenté avec une complexité de $O(SORT(N) \times log\left(\frac{N}{M}\right))$. Par rapport à TerraFlow, TerraStream a améliore la façon dont les algorithmes modélisent les cours d'eau (calcul des directions d'écoulements et des flots d'accumulations). Cette phase est aussi réalisée dans la complexité $SORT(N)$ en terme d'opérations d'entrées/sorties. La complexité dans cette phase est similaire à TerraFlow. A partir des flots d'accumulations de tous les sommets, le réseau hydrographique (voir [O'callaghan 1984]) peut être aussi extrait dans la complexité $O(SORT(N))$

opérations de E/S. La dernière phase de TerraStream est le calcule de la la hiérarchie de la LPE. Elle distribuera une étiquette (voir [Verdin 1999]), à chaque pixel d'un MNT en utilisant l'algorithme externe développé par Arge (voir [Arge 2006]). Cet algorithme est réalisé avec une complexité de $O(SORT(N) + \frac{T}{B})$ opérations d'entrées/sorties, où T est un nombre total d'étiquettes. Ce type de calcul se base sur un graphe de directions d'écoulements en prenant en compte le type mono direction d'écoulements et les flots d'accumulations de chaque pixel.

Comme pour TerraFlow les auteurs ne semblent pas avoir envisagé la parallélisation de leur approche alors que la notion de pipeline suggère à minima une possibilité de distribution de chaque étape sur une machine distincte.

2.3 Parallélisation de modèles de directions d'écoulements

En hydrologie, la parallélisation de modèles de directions d'écoulements a été seulement étudiée dans la contribution de [Wallis 2009]. Les auteurs ont proposé une implémentation parallèle du modèle D8 en se basant sur une file d'attente afin d'éviter la récursion pendant le calcul des flots d'accumulations. Cette implémentation est réalisée sur un machine à mémoire distribuée. Elle répartit le MNT sur les processeur sous la forme de blocs de lignes. Des synchronisations sont nécessaires entre les processeurs afin de maintenir en cohérence les valeurs des pixels au bords des blocs distribuées.

Après avoir traité le MNT pour élimination des puits (voir **Section 2.1.2**), qui créeraient des cuvettes d'où l'eau ne sortirait pas vers son exutoire global, et des plateaux (voir [Garbrecht 1997]), qui introduiraient des ambiguïtés, l'algorithme de Tarboton est utilisé pour le calcul des directions d'écoulements (cf. **Algorithme 2**). Il s'agit d'un algorithme parallèle sans synchronisation ni communication. Chaque processeur traite son bloc de données indépendamment des autres.

Algorithme 2: Calcul parallèle des directions d'écoulements

```
1 Initialiser ;
2 for (∀p ∈ D_i) do
3 |   if (p n'est pas déterminié la direction d'écoulements) then
4 |   |   for (∀n est les voisins de p) do
5 |   |   |   sélectionner n tel que PENTE(p,n) est maximum et plus grand
  |   |   |   que 0 ;
6 |   |   end
7 |   end
8 end
9 F(i) ← direction vers n;
```

Afin de calculer des flots d'accumulations, les auteurs ont proposé une structure de données s'appelant le graphe de dépendances noté ici GD. Il s'agit d'un graphe

orienté encodant la dépendance de chaque pixel p vis-à-vis de ses voisins. Un arc allant de q vers p indique que le voisin q s'écoule vers p. Avec cette structure, il est aisé de connaître combien de pixels voisins s'écoulent vers p. Cette information est nommée *valeur de dépendance*, elle est stockée pour chaque pixel p, elle est notée : $GD[p]$. Il faut connaître les valeurs des flots d'accumulation de $GD[p]$ pixels voisins pour pouvoir calculer le flot d'accumulation de p. Chaque processeur gère non seulement un graphe de dépendances restreint à son sous-domaine mais aussi deux graphes tampons GDT_{haut} et GDT_{bas} afin de se communiquer des dépendances des pixels avec leurs voisins en haut et ceux en bas.

Chaque processeur crée sa propre queue Q_i. Initialement tous les pixels $p \in D_i$ dont la valeur de dépendances est égale à zéro ($GD_i[p] = 0$) sont insérés dans cette queue. La valeur du flot d'accumulation de ces pixel est connue, elle vaut 1.Cette phase est complètement parallèle et sans synchronisation. Le calcul des flots d'accumulation commence ainsi : chaque processeur trouve dans sa queue un pixel p, s'écoulant vers un voisin n. La dépendance de n dans le GD_i qui valait $GD_i[n] = \beta$ est alors mise à jour en la décrémentant de un. Cela signifie que la valeur du flot d'accumulation du pixel n est encore dépendante de la connaissance de $GD_i[n] = \beta - 1$ pixels. Si toutefois après décrémentation, $GD_i[n]$ est égal à zéro, alors son flots d'accumulation peut être calculé, c'est la somme des flots de ces voisins dans le GD_i plus 1. Puis le pixel n est inséré à la queue Q_i. Dans le cas où le voisin n de p n'appartient pas au domaine considéré D_i, la dépendance de n ne peut être décrémentée sur GD_i, la mise décrémentation est réalisée sur le GDT_{haut} ou le GDT_{bas}.Lorsque Q_i est vide, la communication des dépendances entre des processeurs voisins est réalisée. Chaque processeur met à jour son graphe de dépendances en fonction de deux graphes tampons reçus. Les pixels dont la dépendance est égale à zéro sont alors insérés dans Q_i. Le traitement des pixels dans Q_i est répété comme dans la phase précédente. L'algorithme parallèle s'arrête lorsque toutes les queues de tous les processeurs sont vides après avoir mis à jour le GD_i à partir de la communication des deux graphes de tampons.

Une contrainte importante de l'implémentation de ce code est que pour chaque processeur, la queue locale Q_i contient, à tout moment, tous les pixels de D_i dont la dépendance est égale à zéro.Ceci peut conduire à l'utilisation d'une grande quantité de mémoire pour stocker ces pixels dans la phase initiale.

2.4 Conclusion

Dans ce chapitre nous avons présenté les différentes méthodes proposées en hydrologie afin de calculer la directions de l'eau sur le terrain puis d'en déduire les flots d'accumulation. En présentant les méthodes séquentielles nous avons montré l'importance de la préparation des données afin de supprimer les incohérences liées aux données. Nous avons vu que ces modifications d'homotopie sont souvent nécessaires pour le bon déroulement des algorithmes. Les hydrologues prennent grand soin d'affiner l'analyse en proposant des schémas à multiple directions d'écoulements ou en

essayant de déduire les écoulement les plus réalistes possible même quand il n'existe par de données ou que les zones sont plates. Sans nier l'importance des ces raffinements nous proposons d'utiliser un modèle classique de type D8 afin de montrer la faisabilité d'une parallélisation efficace sans pour autant prendre partie pour l'un des raffinements chers au hydrologues. Nous pouvons aussi résoudre le problème des zones plates ou vides en choisissant arbitrairement une direction à privilégier tout en laissant la possibilité aux hydrologues d'implanter la méthode qui leur paraît la plus pertinente au sein de notre solution.

C'est une solution de type D8 qui est proposée dans le SIG GRASS avec les implémentations de TerraFlow et TerraStream qui tentent en particulier de relever le défi posé par les données massives. Nous retenons de ces approches l'idée de l'utilisation d'un graphe de cuvettes permettant d'alléger considérablement la charge de calcul, toutefois ces approches sont axées sur les machines séquentielles.

Les tentatives parallélisation de ces problèmes sont rares et se heurtent à la quantité d'espace mémoire nécessaire pour stocker les structures de données indispensables pour le calcul des flots d'accumulation en chaque pixel. Il apparaît donc intéressant de coupler une parallélisation efficace avec l'utilisation d'un graphe de cuvette afin de réduire l'empreinte mémoire .

Enfin il apparaît que les méthodes présentées déduisent les bassins versants à partir des flots d'accumulation et non l'inverse. Elles ne peuvent donc pas accélérer le calcul des flots d'accumulation en réalisant les additions des contributions des différents bassins versants ce qui pourrait constituer une piste intéressante. De même le graphe de cuvettes de TerraStream ne représente les bassins versants qu'au seul niveau des cuvettes, alors qu'il pourrait être étendu à de plus vastes bassins versants. Ce serait possible à condition d'être capable de rassembler les cuvettes entre elles. Pour réaliser ce type de manipulations il est intéressant d'étudier les méthodes de l'analyse d'image ce qui est réalisé dans le chapitre suivant.

Chapitre 3

Ligne de Partage des Eaux dans la Morphologie Mathématique

Sommaire

3.1 Ligne de Partage des Eaux (LPE) 42
3.1.1 LPE par simulation d'inondation 42
3.1.2 LPE par simulation de pluie 45
3.2 Parallélisation de la Ligne de Partage des Eaux 46
3.2.1 Parallélisation de la LPE dans le cadre de la mémoire distribuée 46
3.2.2 Parallélisation de la LPE dans le cadre de la mémoire partagée 49
3.3 Ligne de Partage des Eaux et Fusion hiérarchique de Région 49
3.3.1 Conclusion 51

3.1 Ligne de Partage des Eaux (LPE)

Dans le domaine de l'analyse d'image, qui est une branche de la morphologie mathématique, la segmentation d'images est un processus qui consiste à partitionner une image en régions connexes et disjointes présentant une homogénéité selon certains critères, par exemple les niveaux de gris ou une texture [Haralick 1985]) (voir **Figure 3.1**).

(a) Image à segmentation (b) Résultat désiré

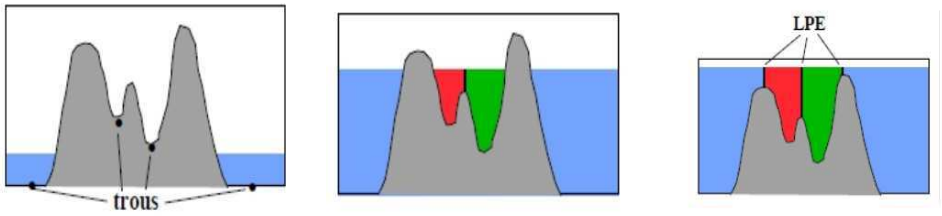
FIGURE 3.1 – Segmentation d'Images dans la Morphologie Mathématique

La *Ligne de partage des Eaux* (en abrégé LPE), proposée par Digabel et Lantuéjoul [Digabel 1978, Lantuéjoul 1978] pour la segmentation d'images, est un outil puissant et rapide. Elle a été aussi développée par Beucher et Lantuéjoul [Beucher 1979]. Elle est choisie pour la segmentation d'images en régions [Beucher 1993, Bieniek 2000, Vincent 1991]. Plusieurs algorithmes sont implémentés en se basant sur la simulation de l'inondation [Vincent 1991], et sur la simulation de pluie [Meijster 1998, Bock 2005, Osma-Ruiz 2007, Stoev 2000]. L'idée consiste à considérer une image en niveau de gris comme un relief topographique. Le niveau de gris d'un pixel de l'image est interprété comme son altitude dans le relief topographique. Cette analogie avec le problème du calcul des bassins versants des MNT conduit à penser que les idées qui ont prévalues pour résoudre le problème en analyse d'image pourraient être exploitées dans le domaine géo-scientifique.

3.1.1 LPE par simulation d'inondation

L'idée de la LPE par inondation (voir [Beucher 1979, Vincent 1991]), parfois appelée LPE par immersion, a été introduite pour la segmentation d'image par S. Beucher et C. Lantuéjoul [Beucher 1979]). Étant donnée une image F considérée comme un relief topographique, celle-ci est progressivement inondée niveau par niveau à partir de ses trous, appelés minima locaux (voir **Figure 3.2(a)**). L'eau montante de ses trous parcourt les bassins versants. Quand les eaux issues de deux minima différents se rencontrent en un lieu, une digue est placée afin d'empêcher ce mélange (voir **Figure 3.2(b)**). À la fin de l'immersion, l'ensemble des digues constituent la ligne de partage des eaux (en abrégé LPE) (voir **Figure 3.2(c)**).

L'idée de la plupart des implémentations consiste à détecter d'abord tous les



(a) (b) (c)

FIGURE 3.2 – Simulation d'immersion dans la LPE

minima locaux et à distribuer à chacun de ces minima une étiquette unique. L'algorithme simule ensuite l'inondation en utilisant une *file d'attente hiérarchique* (FAH) (voir [Beucher 1993, Meyer 1994]). Cet algorithme permet de rapidement construire la ligne de partage des eaux d'une image. Une FAH est un ensemble de N files d'attentes du type de FIFO (First In First Out en anglais) dont le niveau de priorité correspond à un niveau de gris, le nombre de N correspondant au nombre de niveau de gris. Un pixel de niveau de gris h est inséré dans la file d'attentes correspondante. Plus le niveau de gris est fort, moins la priorité est faible. Initialement, tous les minima dans une image sont détectés et associés à des étiquettes disjointes. Ces pixels minima sont aussi insérés dans les files d'attentes correspondantes.

L'algorithme commence par la file d'attente F de plus forte priorité. Un pixel p étiqueté par le label L_p est retiré de la file F. Ses voisins v dont l'étiquette est encore nulle sont identifiés. Ils reçoivent l'étiquette associée à $p : L_p$; puis sont insérés dans leur file d'attentes correspondantes. Idem pour les autres pixels dans la file F. Lorsque cette file est vide, elle est supprimée et la file d'attente suivante peut alors commencer à traiter ses pixels. Cela assure la progression séquentielle niveau par niveau de l'inondation.

Ce traitement s'arrête lorsque toutes les files sont vides. L'implémentation de la FAH est illustrée dans **Algorithme 3**. Cet algorithme est bien adapté bien à l'inondation en le combinant avec des marqueurs permettant l'élimination de la sur-segmentation dans la LPE, de plus, pour les pixels appartenant aux plateaux, il n'a pas de traitement particulier, il n'utilise que l'ordre de la FAH afin d'inonder ces pixels. Cet algorithme ne reproduit pas exactement le processus d'inondation qui a été défini plus haut (voir [Betser 2005, Beucher 2004]). Dans cette définition, à chaque niveau inondé, tous les pixels adjacents aux zones inondées doivent être traités en même temps et en parallèle. Cette condition n'est pas assurée dans cet algorithme. De plus, la propagation arbitraire sur les plateaux génère des erreurs sur la position de la LPE. C'est ce qu'on appelle le biais de la LPE. Une amélioration de cet algorithme a été aussi proposée afin d'éliminer ce biais (voir [Betser 2005, Beucher 2004]). Enfin, un défaut majeur de cette solution est qu'elle impose l'utilisation de beaucoup de mémoire pour les images volumineuses.

Algorithme 3: Algorithme FAH pour la LPE dans la morphologie

Input : Image de l'entrée f, Image de l'étiquette g
Output : Image de l'étiquette g

```
// Distribuer une étiquette unique pour un minimum ou marqueur
for (∀m ∈ Minima ou Marqueurs) do
    g[m] = Label_unique;
    m → FAH;
end

while (FAH ≠ ∅) do
    // extraire pixel x de la FAH
    x ← FAH;

    // déterminer les pixels voisins de x d'étiquette nulle
    N_g(x) = { voisins de x d'étiquette nulle};

    // Pour chaque voisin y d'étiquette nulle
    for (∀y ∈ N_g(x)) do
        // assigner à y la même étiquette que x
        g[y]= g[x];

        // insérer y dans la file d'attente de la FAH
        // de priorité correspondant au niveau de gris de y dans f
        index = f[y];
        y → FAH_index;
    end
end
```

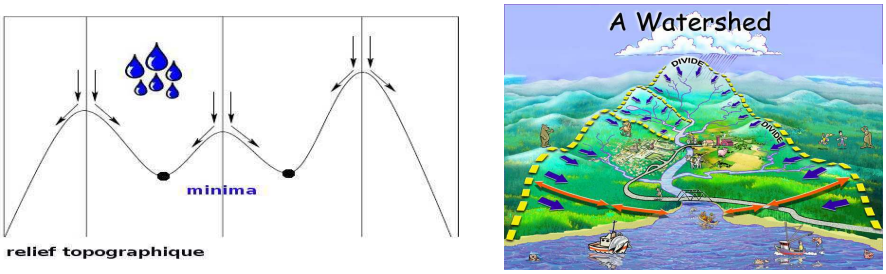
Luc Vincent et Pierre Soille ont proposé un algorithme plus rapide et souple que ceux existants précédemment pour un calcul de la ligne de partage des eaux dans le domaine de segmentation d'images [Vincent 1991]. Cet algorithme se base sur l'immersion d'un relief topographique. Il s'adapte bien aussi à plusieurs types de représentation d'une image, tels que les graphes et les images de 4-connectivité, de 6-connectivité et de 8-connectivité. L'idée consiste dans une analyse mettant en jeux deux phases. La technique de distribution proposée par E.J Isaac et R.C Singleton (voir [Isaac 1956]) est d'abord utilisée afin de trier les pixels d'une image dans l'ordre croissant de leurs valeurs de gris. Ce tri a une complexité linéaire. Vient ensuite une étape d'inondation. À chaque niveau h de l'inondation, les bassins versants associés aux minima sont de plus en plus élargis en calculant les zones d'influences géodésiques des minima parmi des pixels correspondant au niveau de gris h. Les pixels restants n'appartenant pas encore à un bassin versant sont de nouveaux minima détectés. A un niveau de gris donné, lorsque deux bassins versants

différents se rejoignent, les pixels de bordure reçoivent une étiquette particulière notée WSHED. Ces pixels sont les pixels de la Ligne de Partage des Eaux (en abrégé LPE). Ces pixels ne sont plus modifiés par la suite et la valeur WSHED est propagée lors des traitements ultérieurs. En cas de plateaux séparants des bassins versants, cela peut produire une LPE épaisse [Najman 2003, Rambabu 2007]. Ceci peut constituer un problème si le but final de l'étude porte moins sur les bassins versant que sur les LPE qui les séparent. Cela interdit pratiquement l'utilisation du résultat de l'algorithme de LPE pour effectuer une analyse de saillance sur l'image initiale.

L'implémentation se base sur une file d'attente FIFO qui permet d'éliminer la récursion naturelle de cette méthode (voir [Vincent 1991, Roerdink 2001]). Avant l'inondation, cette méthode n'a pas besoin une phase spécifique permettant de détecter tous les minima locaux. Le séquencement de la méthode par inondation dépend non seulement du nombre de niveaux de gris dans une images, mais aussi du nombre de pixels à chaque niveaux gris dans la phase de l'inondation. C'est une méthode de calcul de la LPE globale et intrinsèquement séquentielle. Elle est semble difficile à implémenter d'une manière parallèle.

3.1.2 LPE par simulation de pluie

Cette méthode est fondée sur le principe intuitif qu'une goutte de pluie tombant sur un relief topographique (voir **Figure 3.3(a)**) s'écoule vers le bas en choisissant son voisin le plus bas. Ce parcours de l'eau s'arrête lorsqu'il atteint un minimum local m_i. Un bassin versant CB_{m_i} correspondant à un minimum m_i est défini par un ensemble des pixels q du relief tels que le déplacement d'une goutte de pluie tombant en q fini en atteignant minimum m_i (voir **Figure 3.3(b)**).



(a) (b)

FIGURE 3.3 – Simulation de pluie dans la LPE

Pour choisir le voisin le plus bas, la plupart des implémentations de cette méthode se basent sur la pente la plus grande. Cette notion a été définie par Meyer (voir [Meyer 1994]). Étant donnée une image de niveau gris pondéré par la fonction $\mathrm{f} : \mathcal{D} \longrightarrow \mathbb{N}$, où $\mathcal{D} \subseteq \mathbb{Z}^2$ est le domaine global de cette image et $f(p)$ dénote le niveau de gris d'un pixel $p \in \mathcal{D}$; une grille E est un sous-ensemble de $\mathbb{Z}^2 \times \mathbb{Z}^2$.

Définition 1 *Une pente la plus grande (lower slope en anglais) de f à un pixel p*

est définie comme une pente maximale entre p et tous ses voisins plus bas que p. Elle est formalisée ci-dessous :

$$LS(p) = \max_{v \in N_G(p) \cup \{p\}} \left(\frac{f(p) - f(v)}{d(p,v)} \right) \tag{3.1}$$

où $N_G(p)$ est l'ensemble de voisins de p sur la grille $G = (V, E)$ et $d(p, v)$ est la distance associée à l'arête (p, v). La valeur de $d(p, v)$ est égale à 1 dans la plupart des cas, mais elle peut valoir $\sqrt{2}$ lorsque p et v sont diagonaux dans la 8-connectivité.

Une des difficultés principales dans cette approche est l'apparition des plateaux dans l'image. Le problème survient lorsque chaque pixel, qui est à l'intérieur d'un plateau, détermine son voisin plus bas à l'aide du calcul de la pente la plus grande. Quel voisin choisir ? Certains auteurs proposent de n'utiliser cette méthode que sur des images dans lesquelles il n'y a plus de plateaux. Dans ce type d'image, tout pixel qui n'est pas un minimum local a toujours au moins un voisin d'un niveau de gris inférieur strictement. Ce type d'image est qualifié en anglais de "lower complete image" [Soille 2004] ce que l'on pourrait traduire par image sans zone uniforme. On peut imaginer que cette propriété soit obtenue à l'issue d'une prétraitement spécifique. Après l'obtention d'une image à segmenter "lower-complete", chaque minimum local reçoit une étiquette unique. La simulation de pluie est ensuite réalisée en se basant sur les chemins descendants. L'algorithme de la simulation de pluie (voir [Moga 1995b]) consiste, pour chaque pixel p à tracer vers les minimum locaux m_i, un chemin descendant $\pi_{[p,m_i]}$. Tous les pixels $q \in \pi_{[p,m_i]}$ sont marqués par l'étiquette de m_i. Intuitivement, chaque goutte d'eau, suit son propre chemin et ne dépend pas d'autres gouttes voisines. On constate que cette méthode est intrinsèquement locale, elle est bonne candidate pour une parallélisation.

Plusieurs propositions ont été faites pour améliorer non seulement le temps de calcul, mais aussi l'utilisation de la mémoire (voir [Stoev 2000]). Elles ont été implémentées en ce basant sur une pile qui permet de tracer les chemins descendants. A l'opposé certains algorithmes (voir [Meijster 1998, Bock 2005]) sont basés sur des opérations portant sur des composantes connexes ("connected component operators" en anglais).

3.2 Parallélisation de la Ligne de Partage des Eaux

Lorsque les images à traiter sont volumineuses et nombreuses, les calculs aboutissant à la détermination des LPE sont longs et coûteux en terme de ressource mémoire. C'est la raison pour laquelle, le calcul parallèle de la LPE a été envisagé dans un certain nombre de contributions (voir [Bieniek 1997, Meijster 1996, Moga 1995a, Moga 1995b, Moga 1995c, Moga 1994, Moga 1995d, Meijster 1995, Moga 1997a, Moga 1998a, Moga 1997b, Roerdink 2001]). Il existe deux approches selon que l'on utilise une machine parallèle à mémoire distribuée ou à mémoire partagée.

3.2.1 Parallélisation de la LPE dans le cadre de la mémoire distribuée

La première approche, qui utilise des machines à mémoire distribuée, à été proposée par Moga (voir [Moga 1995a, Moga 1995b, Moga 1995c, Moga 1994, Moga 1995d, Moga 1997a, Moga 1998a, Moga 1997b]), ainsi que par Meijster et Roerdink (voir [Meijster 1995, Roerdink 2001]). Elle repose sur le principe de la subdivision du domaine. L'idée consiste à subdiviser une image F en imagettes F_i et chaque F_i est distribuée à un processeur P_i où l'étiquetage des pixels est réalisé par un algorithme séquentiel. Des points de communication entre des processeurs voisins sont insérés lorsque le résultat partiel du processeur P_i est dépendant de son voisin P_j. Quand il existe des plateaux non-minimum, ceux-ci doivent être traités afin d'obtenir une image "lower-complete" ce qui conduit à réaliser des communications très coûteuses (voir [Moga 1997a]). Le résultat global est obtenu par la fusion des résultats partiels en $log_2(|P|)$ étapes (voir [Bieniek 1996]) où $|P|$ est le nombre de processeurs.

La première implémentation parallèle de cette approche modélise l'inondation en utilisant une file d'attente hiérarchique (en abrégé FAH) (voir [Moga 1997a]). C'est une approche utilisant le modèle de programmation parallèle SPMD (Single Program Multiple Data en anglais). L'image est subdivisée en sous-images et chacune est distribuée à un processeur. Une zone étendue permet de gérer en local une copie des pixels au bords des sous domaines appartenant aux processeurs voisins. L'algorithme parallèle consiste en deux étapes : la détection des minima locaux et l'inondation via une file d'attente hiérarchique FAH.

La première étape consiste à d'abord classifier les pixels en deux groupes : les pixels non-minima et les pixels minima. Ce type de traitement peut être réalisé en un seul passage sur une sous-image. Chaque minimum reçoit une étiquette unique pour toute la distribution, ainsi, à terme, chaque composante connexe associée à un minimum aura une étiquette unique. La file d'attente hiérarchique est simultanément initialisée par des pixels minima entourés des pixels non-minima. Dans la deuxième étape, chaque processeur réalise l'inondation sur son sous-domaine D_i en se basant sur sa propre file d'attente hiérarchique comme dans la manière séquentielle afin de propager les étiquettes des minima aux autres pixels (voir [Beucher 1993]). À cause de la distribution d'une image en blocs, l'inondation est ainsi confinée dans un sous-domaine, de plus, les pixels appartenant à un même plateau non-minimum peuvent être partagés dans certains sous-domaines. Afin de propager l'inondation aux sous-domaines voisins et ainsi de la rendre globale, deux solutions ont été considérées.

Dans la première (voir [Moga 1994]), chacun des processeur P_i réalise l'inondation locale dans son sous-domaine D_i mais avec une synchronisation globale entre chaque niveau de gris. Cette méthode cadence globalement l'exécution et garantit que pour un niveau de gris donné, tous les pixels p appartenant au bord de tous les sous-domaines sont traités. Il y a un traitement particulier réalisé sur les pixels des bords qui possèdent des voisins dans la zone étendue qui sont plus bas ou à la même hauteur. A chaque niveau de l'inondation, une communication entre les processeurs voisins est réalisée afin d'échanger des étiquettes des pixels au bord des

sous-domaines. Cette phase est très coûteuse car elle s'applique sur l'ensemble des pixels des zones étendues. Par ailleurs les synchronisations globales impliquent inévitablement des temps d'attente car le volume des traitements à réaliser par niveau de gris n'a pas de raison d'être uniformément réparti.

Dans la seconde solution (voir [Moga 1997a]), chaque processeur P_i réalise d'abord localement l'inondation en fonction de tous les niveaux de gris dans son sous-domaine D_i sans aucune synchronisation. Après cela, la communication entre les processeurs voisins et la ré-inondation aux sous-domaines sont ensuite itérés jusqu'à la stabilité de la propagation des étiquettes. Cela réduit la quantité de la communication nécessaire pour la ré-inondation.

Une autre voie pour la parallélisation de la LPE consiste à étiqueter les composantes connexes relatives à la relation entre des pixels voisins. Dans [Moga 1998b], les auteurs présentent deux algorithmes parallèle implémentés sur le modèle de programmation parallèle SPMD (Single Program Multiple Data en anglais). Dans un premier temps, les minima sont détectés et distribués avec une étiquette unique ; l'image à segmenter est ensuite transformé en image "lower-complete" (voir [Moga 1997a, Roerdink 2001]) en se basant sur la distance géodésique de tous les pixels à l'intérieur d'un plateau non-minimum. Enfin, l'étiquetage des pixels non-minimum est respectivement réalisé en utilisant l'un des deux algorithmes qui sont décrits ci-après.

Le premier algorithme, basé sur la simulation de la pluie, réalise l'étiquetage sur les chemins descendants. Un chemin descendant $\pi_{(p,q)}$ peut seulement être résolu si et seulement si le pixel q a déjà une étiquette. Pendant l'étiquetage, chaque processeur utilise une file d'attente (FIFO Queue en anglais) afin de stocker tous les chemins descendants $\pi_{(p,q)}$ qui ne peuvent pas être résolus. Après avoir fini en local, les processeurs voisins communiquent afin d'échanger l'étiquette de tous les pixels au bord de chaque sous-domaines. Chaque processeur passe toute sa file d'attente afin de continuer à résoudre les chemins descendants $\pi_{(p',q')}$ non-résolu. Ce processus est réitéré jusqu'à ce que tous les pixels des sous-domaines soient étiquetés.

Le second algorithme se base sur l'inondation en utilisant la technique de "Hillclimbing" (en anglais) dans lequel chaque processeur P_i possède une file d'attente (FIFO Queue en anglais) qui lui est propre et qui est initialisée par les pixels minima détectés. À partir de ces minima stockés dans la file d'attente, une inondation est réalisée en utilisant la technique de "Hillclimbing" décrite à présent. Un pixel étiqueté p retiré de cette file d'attente voit son étiquette propagée à ses voisins q dans son sous-domaine ($q \in N_G(p) \cup D_i$) ; ces pixels q sont ensuite insérés dans la file d'attente afin de devenir de nouveaux candidats de propagation. Lorsque la propagation locale est finie, la file d'attente locale est vide aussi. Pendant la propagation des étiquettes, le chemin calculé peut parfois toucher des bords du sous-domaine. Dans ce cas, l'étiquette d'un certain nombre de pixels dans les sous-domaines adjacents doit être échangée afin de continuer la propagation. Le processeur P_i reçoit ainsi l'étiquette de tous les pixels appartenant au bord B_i de D_i. Après avoir reçu de nouvelles étiquettes des pixels $p \in B_i$, l'inondation est poursuivie dans D_i à partir de ces pixels p. Continuellement, ils sont insérés dans la file d'attente et propagent leurs étiquettes vers leurs voisins non étiquetés. Cette phase est itérée jusqu'à ce

que l'inondation globale soit finie.

Une autre parallélisation de la LPE de cette approche se base sur une condition locale (voir [Bieniek 1998, Bieniek 2000]) en la combinant avec des opérateurs de composantes connexes. Elle a été aussi considérée par Bieniek (voir [Bieniek 1997]) et Moga (voir [Moga 1997b]). Son idée (décrite dans [Bieniek 1997]) consiste à localement résoudre le problème de la LPE dans les sous-domaines sans synchronisation. Des étiquettes temporaires sont distribuées au pixels qui sont inondés à partir d'autres sous-domaines adjacents. Les connectivités au bord de chaque sous-domaine sont stockées dans un graphe (voir [Alnuweiri 1992]) ou une table équivalente (voir [Embrechts 1993, Johansson 1994]). Les étiquettes globales sont finalement calculées en fusionnant des résultats partiels en $log_2(P)$ étapes (où P est un nombre de processeurs à manipuler). Cette fusion se base sur une opération de réduction en utilisant l'algorithme Union-Find (voir [Tarjan 1983, Leighton 1992]) de Tarjan où la technique de compression de chemin a été proposée pour améliorer la performance de ce type de calcul.

Un autre algorithme, se basant sur des passages séquentiels (Sequential Scanning en anglais) pour le calcul parallèle de la LPE, est abordé dans [Moga 1995d]. Les auteurs reprennent l'idée du schéma d'inondation définie par F. Meyer, où la distance topographique (voir [Meyer 1993]) entre les minima et les autres pixels est calculée. La stratégie de parallélisation ce décompose en quatre étapes. La détection des minima dans un sous-domaine est d'abord réalisée. L'image est ensuite transformée en image "lower complete" afin d'éliminer les plateaux. C'est cette image qui est ensuite utilisée dans l'étape suivante qui consiste à étiqueter tous les minima puis, dernière étape, à inonder l'image afin de propager ces étiquettes en se basant sur la distance topographique. Ce calcul est itéré jusqu'à ce que la stabilité globale des étiquettes soit atteinte. Lors d'une étape de ce calcul, lorsque l'étiquette des pixels au bord du sous-domaine est changée, un nouveau raster peut être engendré qui influe sur la stabilité globale des étiquettes. La stabilité globale doit être détectée par un processeur maître. L'accélération du calcul est fonction du nombre de processeurs, cependant, cette augmentation n'est pas linéaire et fini par saturer. De plus, elle dépend du contenu de l'image, et surtout elle est sensible à la présence des plateaux non-minima.

3.2.2 Parallélisation de la LPE dans le cadre de la mémoire partagée

Une approche, correspondant à l'utilisation d'une machine la mémoire partagée, est proposée dans [Meijster 1996]. Elle procède par subdivision de la fonction à calculer en distribuant les minima locaux aux processeurs qui calculent chacun indépendamment leurs bassins versants. Les tailles des bassins versants ne peuvent être connus à l'avance. En conséquence, le principal inconvénient de cette approche est que le temps de calcul est difficilement prévisible car on ne maîtrise pas la charge de calcul de chaque processeur. De plus, l'accès concurrent à la mémoire partagée est

un goulot d'étranglement important lors du traitement de gros volumes de données.

3.3 Ligne de Partage des Eaux et Fusion hiérarchique de Région

Nous avons vu que la détermination de la ligne de partage des eaux est un outil largement étudié pour la segmentation d'images dans le domaine de la morphologie mathématique. Suivant son principe, chaque minimum produit une région ou une zone, appelé bassin versant. La difficulté vient du fait que les images réelles sont généralement bruitées, lorsqu'on calcule la LPE en se basant sur des gradients, cela aboutit à une forte sur-segmentation due à la présence de nombreux minima générés par le bruit. Afin de surmonter cette sur-segmentation, on a proposé plusieurs approches. La première approche est le filtrage. L'idée consiste à filtrer l'image originale afin de supprimer tous les minima non-significatifs. La deuxième approche consiste à choisir le nombre de minima locaux ainsi que le nombre de zones à mettre en évidence grâce à la LPE. Ici il est supposé que les caractéristiques des objets intéressés sont connues. Les minima inutiles sont effacés. Le choix est réalisé en sélectionnant des marqueurs automatiques ou manuels. C'est l'approche marqueurs (swamping en anglais). La troisième approche consiste à améliorer une segmentation initiale en fusionnant itérativement des paires de régions voisines : C'est la méthode de fusion de régions se basant sur le calcul d'une hiérarchie (voir [Beucher 2005, Hahn 2003, Cousty 2008]).

Le calcul de la hiérarchie, basé sur des valeurs d'extinction, se produit à un niveau différent pour chaque fusion de régions. Ceci est utile pour les approches de segmentation interactive parce que cette méthode offre une grande flexibilité. Les valeurs d'extinction peuvent être le résultat de mesures (voir [Grimaud 1992, Najman 1996, Vachier 1995]).

Dans [Meyer 1999, Meyer 2001a], Meyer a proposé une implémentation basé sur le calcul de la hiérarchie dans un graphe. Chaque nœud correspond à un bassin versant de la surface topographique. Si deux bassins versants sont voisins, c'est-à-dire que ces bassins versants ont une frontière commune (ligne de crêtes), il existe alors une arête qui relie ces deux nœuds dans un graphe. La valeur de cette arête est mesurée comme le gradient minimum des points de passages tout au long de cette frontière commune. Meyer a trouvé que toutes les informations utiles d'une hiérarchie peuvent être stockées dans une structure de données très condensée : l'arbre couvrant de poids minimum (en abrégé MST). Cela signifie que, sur un graphe, l'inondation suit toujours le chemin de la hauteur minimum. Cette considération conduit à un algorithme très efficace de segmentation hiérarchique (voir [Meyer 1999]) et a également été utilisé pour la segmentation interactive (voir [Zanoguera 1999]).

Beucher (voir [Beucher 1990, Beucher 1994]) a proposé une approche de segmentation hiérarchique particulièrement intéressante, s'appelant la cascade (waterfall en anglais). À partir du résultat du calcul des bassins versants, l'idée consiste, à chaque étape, à supprimer tous les contours des bassins versants qui sont complètement entourés par les autres plus élevés. Avec la suppression de ces contours, une partition

simplifiée est obtenue, ce processus peut être itéré. À la fin de la cascade, une région unique couvrant toute l'image est obtenue. Généralement, il existe moins de dix niveaux hiérarchiques produits par cette algorithme en cascade. Un algorithme en cascade efficace basé sur le graphe est aussi présenté dans [Beucher 2005].

Dans [Moga 1998c], les auteurs ont proposé un algorithme parallèle dans le calcul des bassins versants en résolvant le problème de «sur-segmentation». Cet algorithme consiste en deux étapes. La première étape concerne la délimitation des bassins versants de manière sur-segmentée en utilisant des algorithmes parallèles existants (voir [Moga 1995d, Moga 1997b, Moga 1998b]). La suite de cet algorithme parallèle réalise la fusion des bassins versants non-marqués avec ceux-marqués les plus proches, qui sont précédemment choisis, en se basant sur l'opération de contraint de la forêt d'arbres couvrant de poids minimum (constrained MSF operator en anglais).

3.3.1 Conclusion

Nous avons vu dans ce chapitre que la morphologie mathématique aborde elle aussi le problème de la détermination de la LPE. La méthode par inondation de Vincent et Soille permet de déterminer un ordre entre les différents bassin ce qui peut être intéressant pour être à même de regrouper les cuvettes. A l'opposé la méthode par simulation de pluie est propice à la parallélisation mais n'est fonctionnelle que sur des images «lower complete». Les parallélisations de ces algorithmes qui traitent les pixels et font appel à des synchronisations et des communications afin de mettre en cohérence les données sont peu économiques en utilisation mémoire et en temps calcul. L'approche qui consiste à utiliser un graphe portant sur des composantes connexes rappelle la méthode employée par TerraFlow et TerraStream dans le contexte de l'hydrologie. Combinée avec l'algorithme Union-Find de Tarjan elle est bien plus efficace en terme d'occupation mémoire et de rapidité. Résoudre le problème de la sur-segmentation conduit à créer une hiérarchie de cuvettes. Cela conduit à rechercher un MST sur l'image qui respecte des critères propres à l'analyse d'image afin de fusionner des cuvettes entre elles.

Chapitre 4

Conclusion

Lors de l'analyse de l'état de l'art en hydrologie, nous avons choisit de privilégier un modèle de type mono-direction de type D8. Nous nous abstenons d'imposer une phase de pré-traitement afin de ne traiter que des images de type «lower-complete». Nous avons noté que l'utilisation d'un graphe de cuvette est a même de minimiser à la fois l'empreinte mémoire et les calculs de l'algorithme. Par contre il nous semble que ces méthodes peuvent être améliorées pour calculer plus rapidement les flots d'accumulation et être parallélisées efficacement. Nous avons vu que dans la domaine de l'analyse d'image, des solutions ont été apportées pour la parallélisation du calcul des bassins versants. Il doit être possible d'adapter ce type de solution au domaine de l'hydrologie à condition de créer des critères qui ne sont plus ceux de l'analyse d'image tels que les gradients de niveaux de gris ou les regroupement par affinité de couleurs. Nous devons intégrer dans ce type de méthode les notions de hauteurs de passage de l'eau ainsi que celles de regroupement de bassins uniquement si ceux-ci appartiennent au même bassin versant d'un fleuve. Si ce critère est respecté il devrait être possible de s'appuyer sur la hiérarchie afin de calculer rapidement les flots d'accumulation. L'implémentation doit minimiser au maximum l'empreinte mémoire et, pour ce faire, choisir une représentation raster ou graphe selon ce que l'on doivent manipuler tous les pixels ou uniquement des composantes connexes. L'implémentation doit viser à minimiser au maximum les synchronisations et le besoin de communication.

Troisième partie

PARAFLOW : ANALYSE PARALLÈLE DE LPE EN MÉMOIRE PRINCIPALE

Chapitre 5

Principes de base de ParaFlow

Sommaire

5.1 Arbres Couvrants de Poids Minimum 60
5.1.1 Définition 60
5.1.2 Algorithme séquentiel de Borůvka 60
5.2 Calcul de MST adapté au traitement des MNT 61
5.2.1 Notations et définitions 61
5.2.2 Délimitation des cuvettes 62
5.2.3 Hiérarchie des bassins versants 64
5.2.4 Règle Ω 66
5.3 Calcul des flots d'accumulation 69
5.3.1 Objectifs 69
5.3.2 Calcul des flots d'accumulation locaux 70
5.3.3 Calcul des flots d'accumulation globaux 70

Dans ce chapitre, nous introduirons les principes généraux sur lesquels s'appuie la plateforme de calcul répartie, *ParaFlow*, qui a été développée dans le cadre de cette thèse. Cette présentation se fera indépendamment des problématiques de parallélisme qui seront décrites dans le chapitre suivant.

L'objectif de ParaFlow est de faire de l'analyse de terrains modélisés par des MNT sous forme de grilles régulières 2D. ParaFlow est composé d'une série de modules destinés à délimiter les bassins versants des fleuves et à extraire les réseaux hydrographiques à partir d'un MNT sans que celui-ci ne soit modifié. Les différents modules ont été conçus pour s'exécuter sur des machines parallèles à mémoire distribuée telles que les grappes de PC.

Les bases de la méthode s'appuient sur les techniques utilisées dans les domaines de recherches de l'hydro-géologie, du traitement d'images. Elles s'appuient sur le calcul d'un arbre couvrant de poids minimum (Minimal Spanning Tree en Anglais et abrégé en MST). En effet, le parcours de l'eau sur un terrain suit la loi de l'effort minimum. ParaFlow va diriger le calcul du MST de façon à pouvoir déterminer les bassins versants des principaux fleuves au sein du MNT. Ce sera le rôle notamment de la règle que nous avons appelée règle Ω. Cette règle permet de respecter la morphologie du terrain, de prendre en compte la mer ainsi que l'absence d'information aux bords du MNT ou dans les zones du MNT sans données.

Les MNT tels que ceux que l'on peut se procurer sur le site de l'USGS ou auprès de l'IGN, sont constitués d'une multitude de minuscules cuvettes. Ceci est dû au

fait que le MNT est une approximation du terrain réel où un point représente une surface qui peut aller de 250 m^2 à 1000 m^2. Cette incohérence globale du point de vue hydrologique est souvent traitée par des techniques de modifications du MNT (comme le stream-burning par exemple) pour le rendre compatible avec un écoulement naturel de l'eau (voir chapitre 2). ParaFlow lui va plutôt travailler avec ces cuvettes et calculer le trajet, de cuvette en cuvette, de l'eau vers la mer. L'arbre couvrant de poids minimum sera donc calculé sur un graphe constitué des cuvettes contenues dans le MNT de départ et dont les arêtes vont représenter l'*effort* nécessaire pour passer d'une cuvette à une cuvette voisine. Une fois le calcul de l'arbre terminé, ParaFlow pourra changer la direction des flux de certains points, en les forçant à remonter une pente pour sortir de la cuvette dont ils font partie. Enfin, il sera possible de calculer les flots d'accumulations, c'est à dire le nombre de points drainés par chacun des points du MNT.

ParaFlow est constitué de trois modules principaux dont chacun représente une étape dans l'analyse de la LPE à partir d'un MNT :

1. délimitation automatique des cuvettes ;
2. calcul parallèle du MST pour la délimitation des bassins versants des fleuves ;
3. calcul parallèle des flots d'accumulations.

Il est à noter que la première étape peut être vue comme la toute première étape du calcul du MST mais comme elle travaille au niveau des points du MNT (potentiellement plusieurs milliards), elle sera traitée de manière différente, notamment le graphe sous-jacent ne sera pas stocké dans une structure de données particulière mais restera sous forme de grille 2D. Cette étape est aussi la plus lourde en terme de calculs et de données. C'est au cours de cette étape que seront repérées les zones marquées comme Ω (c'est à dire la mer) et les zones marquées comme TROUS (c'est à dire sans données) dans les MNT.

Après avoir délimité les cuvettes du MNT, ParaFlow va calculer un graphe de connectivité entre ces cuvettes afin préparer l'étape suivante. Dans ce graphe, chaque cuvette représente un sommet et est identifiée par un numéro unique. Le poids des arêtes prendra en compte la hauteur minimale sur des frontières entre deux cuvettes voisines. L'ensemble des traitements de cette première étape sont entièrement traitées en parallèle. Ce travail [Hiep-Thuan 2009] a été publié à la conférence internationale *The* 2^{nd} *High Performance Computing and Applications 2009, HPCA 2009 à Shanghai en Chine.*

La deuxième étape de ParaFlow consiste à calculer un arbre couvrant de poids minimal en partant du graphe construit lors de la première étape. Ce calcul va se baser sur une parallélisation de l'algorithme de Borůvka. Elle va consister à construire une hiérarchie de graphes permettant de déterminer quelles sont les arêtes du MST dans le graphe initial. La construction du MST sera dirigée par la règle Ω qui va permettre de fusionner progressivement les cuvettes afin d'aboutir à la délimitation automatique des bassins versants réels des cours d'eau lors de l'avant dernière étape du calcul du MST. L'implémentation dans cette phase est aussi parallèle. Ce travail

(voir [Hiep-Thuan 2010]) a été publié à la conférence internationale "The International Conference on High Computing & Simulation" HPCS 2010 à Caen en France et a été sélection parmi les cinq meilleurs articles de la conférence. Une synthèse des ces travaux a été aussi soumise à un journal international, "*Concurrency and Computation : Practice and Experience*", CCPE.

L'étape finale dans ParaFlow porte sur le calcul des débits au sein du MNT. Ce calcul consiste à déterminer pour chaque point du MNT le nombre de points qui se déversent vers lui. Sur le graphe de directions d'écoulement, les débits des points ne sont représentés qu'au sein des cuvettes mais ne peuvent pas montrer les déversements entre les cuvettes. Afin d'obtenir un résultat cohérent avec la réalité, notre méthode dans cette phase consiste à d'abord déterminer les débits de tous les points p en calculant le nombre de points q dans une même cuvette que p qui se déversent vers p. Ensuite, grâce au MST calculé à l'étape précédente, ParaFlow propagera les débits entre les cuvettes dans cet arbre à travers les points de passages les plus bas. Ce travail (voir [Hiep-Thuan 2011]) a été publié à la conférence internationale, "*The International Conference on Computational Science ICCS-2011 à Singapour*".

Dans le suite, de ce chapitre nous allons principalement décrire la construction du MST et la règle Ω qui la dirige afin de bien expliquer les fondements de la méthode utilisée par ParaFlow. Le chapitre suivant lui décrira en détail, les algorithmes parallèles utilisés à chacune des étapes.

Après avoir présenté l'algorithme de Borůvka dans la section 5.1, nous décrirons la manière dont nous avons adapté cet algorithme pour le traitement des MNT section 5.2 enfin nous allons montrer comment utiliser les résultats précédents pour calculer les flots d'accumulation et déterminer la position des cours d'eau dans la section 5.3.

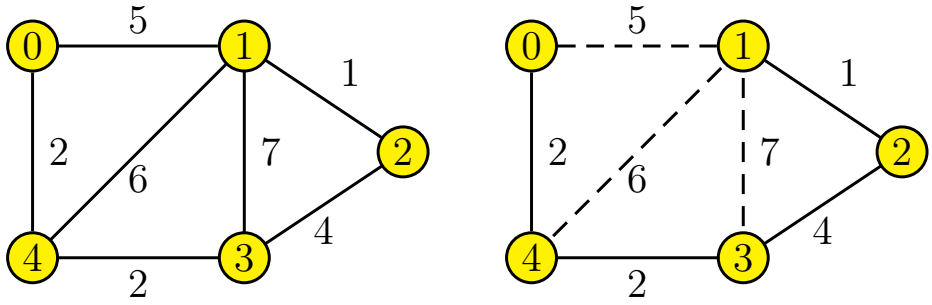
5.1 Arbres Couvrants de Poids Minimum

5.1.1 Définition

Supposons qu'on dispose un graphe $G = (V, E)$ non orienté connexe avec une fonction de pondération $w : E \to \mathbb{R}$. On souhaite alors trouver un sous-ensemble acyclique $T \subseteq E$ qui connecte tous les sommets de G et dont le poids total

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

soit minimum. Puisque T est acyclique et connecte tous les sommets, il forme un arbre, que l'on appelle arbre couvrant de poids minimum (MST).



(a) Graphe G non orienté (b) Arbre couvrant de poids minimum du G

FIGURE 5.1 – Exemple de graphe G connexe, avec un MST (traits pleins).

Le premier algorithme avait été obtenu en 1926 par Borůvka. Il existe encore deux algorithmes célèbres et chacun utilise plus particulièrement l'une des caractérisations des arbres pour trouver la solution : soit en considérant les arbres comme des graphes connexes avec le minimum d'arêtes (Algorithme de Prim), soit en considérant les arbres comme des graphes acycliques avec le maximum d'arêtes (Algorithme de Kruskal). Les deux algorithmes sont gloutons.

5.1.2 Algorithme séquentiel de Borůvka

Les graphes que l'on manipule sont donc des graphes pondérés. Un graphe pondéré G consiste en un ensemble de sommets, noté $V(G)$, associé à un ensemble d'arêtes pondérées, notée $E(G)$, tels que $E(G) \subseteq V(G) \times V(G) \times \mathbb{R}$. Pour une arête (v_1, v_2, w) de $E(G)$, w est le poids de cette arête. Pour une arête $e \in E(G)$ $w(e)$ désigne le poids de e. Le voisinage d'un sommet v de G est un ensemble $N_G(v) = \{v' \in V(G)\ tel\ que\ \exists(v, v', w)\ ou\ (v', v, w) \in E(G)\}$.

Un graphe H est un *mineur* du graphe G s'il peut être obtenu en contractant les arêtes d'un sous-graphe induit de G. Pour la contraction d'une arête (v_1, v_2, w) de G, nous supprimons l'arête (v_1, v_2, w) en fusionnant les deux sommets v_1, v_2 en un sommet z dans H. Dans notre contexte, nous prenons seulement en compte des

mineurs obtenus par des contractions d'arêtes à partir du graphe G (et non à partir des sous-graphes de G).

Pour définir plus formellement la classe des mineurs que nous allons utiliser, nous allons introduire quelques définitions. Soit G un graphe, $\overline{V}$ une partition de $V(G)$. L'ensemble des poids des arêtes qui connectent un sommet de $\overline{v_1}$ avec un sommet de $\overline{v_2}$ ($\overline{v_1}$ et $\overline{v_2}$ étant deux éléments de $\overline{V}$), est l'ensemble $w_{G,\overline{V}}(\overline{v_1},\overline{v_2}) = \{w|(\overline{v_1},\overline{v_2}) \in \overline{V} \times \overline{V}\ et\ \exists v_1 \in \overline{v_1}, v_2 \in \overline{v_2}\ et\ (v_1,v_2,w) \in E(G)\}$. Le poids minimum connectant deux composants $\overline{v_1}$ et $\overline{v_2}$ de $\overline{V}$ est $wMin_{G,\overline{V}}(\overline{v_1},\overline{v_2}) = Min(w_{G,\overline{V}}(\overline{v_1},\overline{v_2}))$ (on définit $Min(\emptyset) = \infty$). Le *$wMin$-mineur de G basé sur $\overline{V}$* est le graphe dont les arêtes sont $\overline{V}$ et les sommets $\{(\overline{v_1},\overline{v_2},w)|(\overline{v_1},\overline{v_2}) \in \overline{V} \times \overline{V}, \overline{v_1} \neq \overline{v_2},\ wMin_{G,\overline{V}}(\overline{v_1},\overline{v_2}) = w\ et\ w \neq \infty\}$ c'est-à-dire tous les sommets de chaque composante de $\overline{V}$ sont contractés et seulement l'arête de poids minimum est préservée dans le mineur.

L'algorithme de Borůvka construit progressivement une hiérarchie des mineurs à partir du graphe initial G. L'idée se base sur la propriété selon laquelle une arête de poids minimum qui connecte deux composants connexes d'un graphe G doit appartenir à l'arbre couvrant de poids minimal, noté MST, de G. Les différentes étapes de l'algorithme sont illustrées ici :

Étape 1 : Pour chaque sommet s de G, choisir une arête de poids minimum incident à s. Pour éviter les cycles, si une arête minimale du sommet est déjà sélectionnée lors de la visite d'un autre sommet, on n'en sélectionne pas d'autre (voir **Figure 5.2(b)**).

Étape 2 : Calculer, par étiquetage des sommets, les composantes connexes du graphe basées sur ces arêtes sélectionnées. (voir **Figure 5.2(c)**).

Étape 3 : Construire un mineur $wMin$ de G en se basant sur des composantes connexes de G qui ont été déterminées dans *Étape 2* (voir **Figure 5.2(d)**). Chacune des composantes connexes correspond à un sommet dans le mineur.

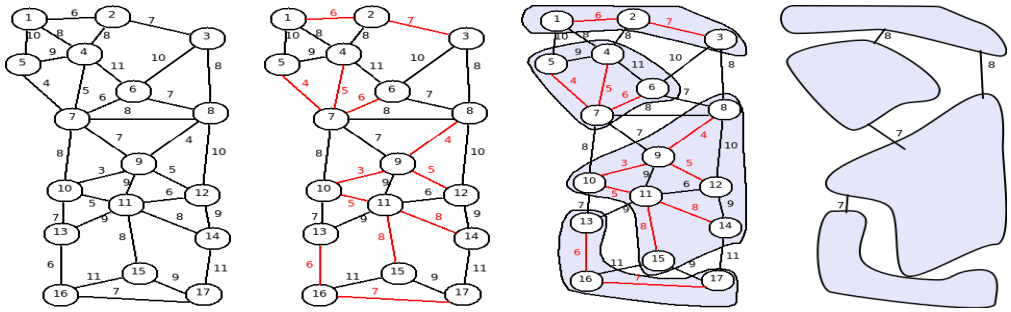
Le $wMin$-mineur, obtenu après cette itération, est l'entrée de l'itération prochaine. L'algorithme terminera lorsque le graphe contracté est réduit à un seul sommet. L'arbre couvrant de poids minimal de G est l'ensemble des arêtes qui ont été choisies dans l'*Étape 1* pour chaque itération. Remarquons que dans l'*Étape 1*, les sommets sont choisis par hasard pendant la recherche des arêtes de poids minimum. La hiérarchie des mineurs peut ainsi être différente en fonction de l'implémentation, même si la complexité reste la même $O(m \times log(n))$ où n, m sont respectivement le nombre de sommets et le nombre d'arêtes du graphe G.

5.2 Calcul de MST adapté au traitement des MNT

5.2.1 Notations et définitions

Voici quelques définitions qui seront utilisées dans la description des algorithmes de ParaFlow.

Une *grille numérique 2D* est une paire (D,h) où D (appelé *domaine* de la grille) est tel que $D \subseteq \mathbb{Z}^2$ et h est une fonction $h : D \mapsto \mathbb{R}$ qui associe à chaque élément de



(a) Graphe initial (b) Étape 1 (c) Étape 2 (d) Étape 3

FIGURE 5.2 – Illustration de l'algorithme de Borůvka

D la valeur contenu dans la grille à cet endroit. Cette valeur est appelée *hauteur* et se note h_v pour un point $v \in D$.

Une telle grille peut être considérée comme une graphe particulier dans lequel les sommets (appelés *points* ou *pixels*) sont les éléments du domaine D. Une grille (D, h) peut se définir par le graphe $G = (V, E)$ où $V = D$ et E est l'ensemble $\{((x_1, y_1), (x_2, y_2), w) |$ (x_1, y_1) et (x_2, y_2) sont voisins et le poids de l'arête w est $max(h_{(x_1,y_1)} h_{(x_2,y_2)})\}$. La connectivité de la grille peut être soit un 4-connectivité, ou une 8-connectivité. Ce graphe associé à la grille (D, h) se note $G(D, h)$.

L'ordre lexicographique est introduit sur des paires d'entiers. Il va nous servir à résoudre de manière simple les zones plates. Cet ordre, noté $\prec$ est définit de la manière suivante : Soient i, j, k et l quatre entiers, $(i, j) \prec (k, l)$ ssi $(j < l)$ ou bien $((j = l)$ et $(i < k))$. Cet ordre $\prec$ est utilisé pour déterminer de manière unique l'arête de poids minimum lorsque plusieurs ont le même poids en terme de hauteur. Notamment, l'ordre $\prec$ élimine le problème de plateaux, les zones formées par au moins de deux pixels ayant la même hauteur.

5.2.2 Délimitation des cuvettes

Cette étape est la première de la construction de la hiérarchie de graphes de l'algorithme de Borůvka. C'est la plus importante en terme de calculs car elle travaille directement sur le MNT. Cette étape se décompose en trois parties. La première va consister à construire un graphe de flux qui va indiquer pour chaque point du MNT la direction dans laquelle l'eau va s'écouler en suivant la pente la plus forte. Ensuite on va délimiter les cuvettes qui seront les composantes connexes de ce graphe de flux, puis afin de préparer l'étape suivante, la connectivité entre les cuvettes sera calculée afin de construire le premier graphe de la hiérarchie.

Le graphe de flux

Pour définir le graphe de flux, nous avons besoin d'introduire la fonction *parent* qui va représenter la direction de la pente la plus forte en chaque point du MNT.

Soit (D, h) une grille 2D, on définit un ordre strict $<$ sur les points de cette grille

de la manière suivante. Pour tous points p et p' de D $p < p'$ ssi soit $h_p < h_{p'}$, soit $h_p = h_{p'}$ et $p \prec p'$.

$\forall p \in D\ parent(p) = p'$ où p' est

- le voisin de p tel que $h_{p'} = Min(\{h_{p''}, p'' \in N_{G(D,h)}(p)\ et\ p < p''\})$ si p a au moins un voisin strictement plus petit que lui,
- p lui-même sinon.

$Parent(D, h)$ représente la relation $\{(p, parent(p)), \forall p \in D\}$.

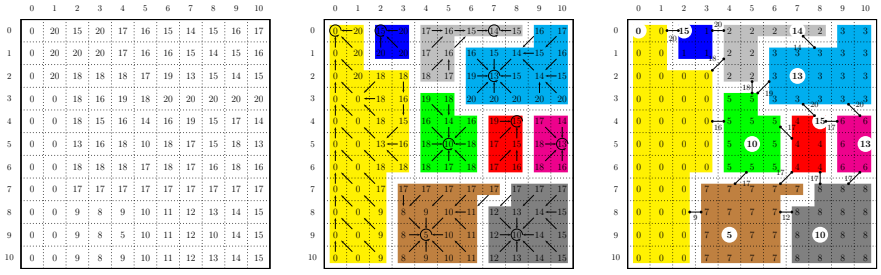
Le *graphe de flux* associé à (D, h), noté $Flux(D, h)$, est le graphe $(D, Parent(D, h))$. La figure 5.3(b) représente le graphe de flux du MNT de la figure 5.3(a).

La construction du graphe de flux est immédiate à partir de sa définition. On peut noter qu'un point n'a besoin que de son voisinage immédiat pour déterminer son parent ce qui sera un atout lors de la parallélisation de cette phase.

Délimitation des cuvettes

Les cuvettes du MNT sont déduites du graphe de flux, puisqu'elles sont en fait les composantes connexes de ce graphe. Chaque composante étant un arbre dont la racine r est appelé **puits primaire**. Les composantes connexes forment les cuvettes (parfois aussi appelés bassins versants) BV du MNT et les puits sont leur exutoire. Ce calcul se fera en attribuant à chaque puits primaire un label et en attribuant ce label à l'ensemble des points appartenant à la même composante connexe. Les calculs de labellisation sont optimisés en utilisant la technique *compression de chemin* [Cormen 1990, Tarjan 1983].

La labellisation du graphe de flux du MNT de la figure 5.3(b) est donnée figure 5.3(c).



(a) MNT (b) Flux et cuvettes (c) GC^0

FIGURE 5.3 – Calcul des cuvettes et du graphe de connectivité GC^0

Connectivité des cuvettes

Cette étape consiste à construire le premier graphe qui sera donné en entrée de l'algorithme de Borůvka. Il va falloir déterminer pour deux cuvettes voisines quel est le point de passage le plus bas entre les deux cuvettes qui servira de poids pour l'arête entre ces deux cuvettes. Ce calcul se fait en une seule passe sur l'ensemble de

la grille en comparant pour chaque point de la grille le label de ses voisins. Si deux voisins p et p' ont des labels différents,

- soit la connectivité entre les deux cuvettes n'existait pas dans le graphe déjà construit et dans ce cas on l'ajoute avec comme poids initial $Max(h_p, h_{p'})$
- soit la connectivité était déjà existante et dans ce cas on met à jour le poids si $Max(h_p, h_{p'})$ est inférieur au poids déjà trouvé.

Notons que seule l'arête allant du point le plus grand vers le point le plus petit p vers p' est considérée en suivant l'ordre sur les points défini précédemment. Le graphe ainsi construit sera appelé $GC^0(D, h)$ ou GC^0 si (D, h) peut être déduit du contexte.

5.2.3 Hiérarchie des bassins versants

Une fois le graphe $GC^0(D, h)$ calculé, on va pouvoir utiliser l'algorithme de Borůvka pour construire le MST qui va représenter le parcours de l'eau de cuvette en cuvette vers la mer et va nous permettre de trouver la ligne de partage des eaux des grands fleuves.

Les différents mineurs du graphe GC^0 seront notés GC^i ou i est le numéro de l'étape au cours de laquelle le graphe GC^i a été construit.

Le calcul de la hiérarchie se réalisera par la fusion progressive des cuvettes, niveau par niveau. Le principe utilisé est le même que lorsqu'on calcule les composantes connexes de $Flux(D, h)$. À un niveau N, le graphe GC^N est définit par (BV^N, E^N) où BV^N est l'ensemble des cuvettes calculées sur le graphe GC^{N-1} et E^N la connectivité de ce graphe. Une cuvette $c \in BV^N$ est représentée par $puits(c)$, le plus petit point qu'elle contient au sens de l'ordre que l'on a définit précédemment sur les points. Pour passer du niveau N au niveau $N+1$ chaque cuvette va "choisir" un déversoir, c'est-à-dire une cuvette dont le puits est plus bas que son propre puits. Contrairement à ce qui se passe au niveau du calcul du graphe de flux, elle ne va pas obligatoirement prendre la cuvette dont le puits est le plus bas car il va falloir prendre en compte l'effort que nécessite le passage d'une cuvette à l'autre. Cet effort est représenté par le poids des arcs du graphe GC^N qui est la hauteur du point de passage le plus bas entre deux cuvettes. La règle qui est utilisée est de sélectionner l'arête de poids le plus faible qui va vers une cuvette dont le puits est plus petit que son propre puits.

Par conséquent, à un niveau N, la fonction *parent* que l'on avait introduit pour le graphe de flux peut s'étendre aux cuvettes de ce niveau de la manière suivante $\forall c \in BV^N$ $parent(c) = c'$ où c' est

- la cuvette voisine de c (avec $(c, c', w') \in E(GC^N)$) telle que $w' = Min(\{w'', (c, c'', w'') \in E(GC^N)$ *et* $puits(c) < puits(c'')\})$ si c a au moins une voisine strictement plus petite qu'elle. Si plusieurs cuvettes voisines ont le poids minimum, celle dont le puits est le plus petit sera choisi.
- c elle-même sinon.

$Parent(GC^N)$ représente la relation $\{(c, parent(c)), \forall c \in BV^N\}$

Le graphe $(BV^N, Parent(GC^N))$, noté GD^N est appelé *graphe de dépendance.*

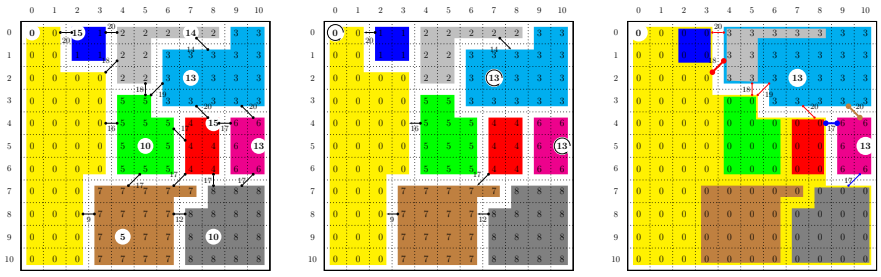
La fonction *parent* représente la fonction de sélection des arêtes du MST. Elle modélise le parcours de l'eau entre les cuvettes.

Les sommets du graphe GC^{N+1} seront les composantes connexes du graphe $(BV^N, Parent(GC^N))$ et l'ensemble des arêtes sera constitué des arcs connectant deux composantes ayant des frontières communes. Si plusieurs arcs connectent les deux composantes seul celui de poids le plus faible sera conservé. La connectivité de ce graphe sera calculé de manière similaire au cas de GC^0, sauf que l'on travaille sur une structure de graphe au lieu du MNT lui-même.

Pour illustrer cette étape nous allons reprendre l'exemple de la figure 5.3 en le prolongeant pour calculer une étape supplémentaire. La figure 5.4 présente ce type de calcul au premier niveau. Le résultat de calcul du premier graphe de connectivité GC^0 est rappelé figure 5.4(a). Sur cette figure, les cuvettes sont représentées par différentes couleurs et leur label est donné dans la grille. Le puits (ou point le plus bas de la cuvette) est donné par le nombre cerclé de blanc. Par exemple le puits de la cuvette 2 est de hauteur 14. Enfin la hauteur du point de passage entre les deux cuvettes est indiquée par l'entier situé à proximité de l'arête représentant la connectivité entre les cuvettes.

Le graphe de dépendances GD^0 est illustré par la figure 5.4(b). On peut remarquer que la cuvette 1 va choisir la cuvette 0 comme déversoir car c'est une voisine dont le puits est plus bas que son propre puits. La cuvette 1 se déverse dans la cuvette 0 plutôt que dans la cuvette 2 car le point de passage vers 0 est plus bas (hauteur 15) que celui vers 2 (hauteur 20). La cuvette 4 a quatre voisines plus basses pour lesquelles la hauteur de passage est de 17, c'est la cuvette 5 qui sera choisie car le puits de cette cuvette est le plus petit (il a une hauteur de 5).

La figure 5.4(c) montre le résultat de la délimitation des cuvettes du graphe de dépendance GD^0 ainsi que du calcul de la connectivité du graphe GC^1 avant l'élimination des arêtes multiples entre deux sommets de ce nouveau graphe.



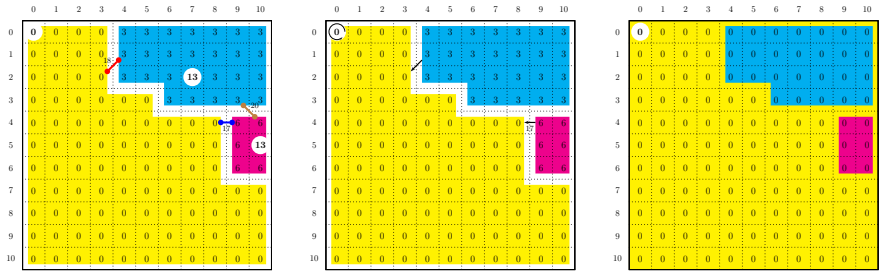
(a) GC^0 (b) GD^0 (c) GC^1_{init}

FIGURE 5.4 – Calcul de la hiérarchie : Premier niveau

Sur la **Figure 5.4(b)**, nous trouvons trois composantes connexes $\mathrm{CC}_1 = \{\, 0 \to 0,\ 1 \to 0,\ 4 \to 7,\ 5 \to 0,\ 7 \to 0,\ 8 \to 7\}$; $\mathrm{CC}_2 = \{2 \to 3,\ 3 \to 3\}$ et $\mathrm{CC}_3 = \{6 \to 6\}$; chacune des composantes connexes CC_i correspond à une cuvette au niveau suivant (voir **Figure 5.4(c)**). Après avoir fusionné les cuvettes au premier niveau, les fron-

tières minimales entre les cuvettes sont aussi calculées en se basant sur la GC_0 et sur le GD_0. Dans **Figure 5.4(c)**, les arêtes en gras sont les frontières minimales entre les cuvettes au niveau 1.

La **figure 5.5** illustre le calcul de la hiérarchie au niveau suivant. Remarquons que la cuvette marquée 0 (zone en jaune), considérée comme une partie de mer, croît rapidement jusqu'à englober l'ensemble du MNT (voir **Figure 5.4(c)** et **Figure 5.5(c)**). Le résultat final, représenté par la **Figure 5.5(c)**, dans laquelle il n' existe qu'un seul bassin versant, ne correspond pas aux besoins dans l'hydrologie, les bassins versants ne sont pas automatiquement délimités comme dans la réalité.



(a) GC^1 (b) GD^1 (c) GC^2

FIGURE 5.5 – Calcul de la hiérarchie : deuxième niveau

C'est pourquoi nous allons introduire une nouvelle règle appelée la règle Ω dans la prochaine section. Cette règle va être appliquée pendant le calcul de la hiérarchie afin d'éviter la mauvaise fusion des bassins versants des fleuves.

5.2.4 Règle Ω

Comme nous l'avons vu dans la section précédente, une application naïve de l'algorithme de Borůvka ne permet pas de délimiter les bassins versants des principaux fleuves contenus dans le MNT car la mer (qui contient le puits le plus bas du MNT) va rapidement englober toutes les cuvettes du MNT. Il va donc falloir modifier les règles de fusions entre les cuvettes pour éviter ce phénomène. Plus généralement, puisqu'on calcule rarement les bassins versants au niveau d'un continent, un MNT donné en entrée de ce calcul contient des informations incomplètes, notamment des morceaux de bassins versants de fleuves dont l'embouchure n'est pas contenue dans le MNT, des morceaux de mer, des zones sans données. La méthode naïve de calcul de la hiérarchie conduit à effectuer des fusions incorrectes. Afin d'obtenir un résultat cohérent avec la réalité, notre méthode se base sur l'idée que les cuvettes ne peuvent se fusioner que si elles appartiennent à une unité hydrologique cohérente, c'est-à-dire qu'elles aboutissent à la mer en passant par la même cuvette voisine de la mer (embouchure).

Un autre problème peut survenir quand une cuvette est au bord du MNT, car aucune information concernant ses voisins à l'extérieur du MNT n'est connue. Sans aucun traitement spécial, ces cuvettes seront fusionnées afin de devenir un seul

bassin versant final du MNT. Cela pose un problème quand l'exutoire réel de ce bassin versant n'est pas dans le domaine global du MNT.

Afin de résoudre ces problèmes, comme dans [Arge 2001a], notre méthode introduit un bassin versant particulier, appelé Ω. On va calculer une forêt de MST dont les racines vont se jeter dans Ω. L'idée sous jacente est de dire que les cuvettes qui vont vers Ω sont les embouchures des fleuves contenus dans le MNT et que l'on va calculer un MST par embouchure afin d'éviter de mélanger les bassins versants de chacun des fleuves. En réalité, le bassin versant Ω est considéré comme la mer. Tous les pixels dont la hauteur est en dessous de zéro mètre (ou d'une autre valeur arbitraire) appartiennent à Ω. Tous ces pixels reçoivent une nouvelle hauteur inférieure à la hauteur de tous les pixels dans le MNT. En mettant en œuvre cette règle, une cuvette (ou une composante connexe) au bord du MNT choisira géralement Ω comme déversoir.

Il faudrait éviter que certaines cuvettes fusionnent trop tôt avec Ω, en particulier celles proches du bord du MNT et qui, sans Ω, auraient été associées à des cuvettes plus à l'intérieur du MNT. En conséquence, il convient d'affiner la méthode en ne permettant pas la fusion des cuvettes avec Ω mais en gardant la trace que certaines cuvettes auraient dû fusionner avec Ω. Dès qu'un sommet devrait fusionner avec Ω, on ne le fait pas, mais on les dits oméga-marqués (noté $\overline{\Omega}$). Une fois une cuvette oméga-marquée, elle reste dans la hiérarchie et on n'essaie plus de la fusionner avec une autre cuvette. Par contre, d'autres cuvettes peuvent sélectionner une cuvette oméga-marquée comme déversoire, dans ce cas au niveau suivant le résultat de la fusion sera lui aussi oméga-marqué.

Cette méthode peut conduire à des configurations où une cuvette c est entourée de cuvettes $\overline{\Omega}$ qui ont toutes une hauteur supérieure à celle de c. Afin de surmonter ces configurations, il suffit d'imposer que cette cuvette isolée c sélectionne la plus petite arête vers un de ses voisins sans tenir compte de la hauteur du voisin.

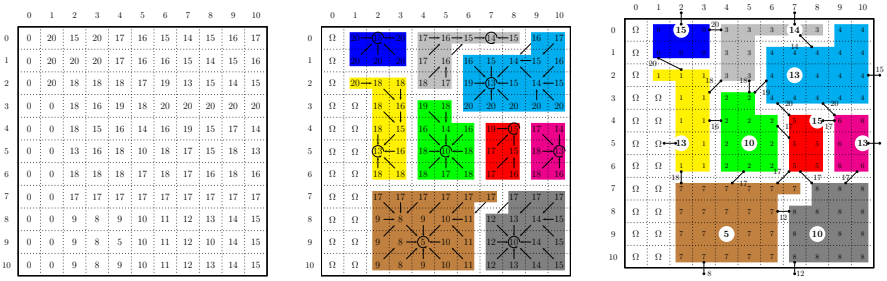
Donc si on veut formaliser la règle Ω, il faut modifier la définition de la fonction *parent* qui sert à calculer le graphe de dépendance. Pour un graphe de connectivité GC^N, la fonction $parent_\Omega$ est définie de la manière suivante : $\forall c \in BV^N$ $parent_\Omega(c) = c'$ où c' est

- c si c est oméga-marquée ou si elle a au moins une voisine non oméga-marquée et que toutes ces voisines c' sont telles que $puits(c) < puits(c')$.
- la cuvette voisine de c (avec $(c, c', w') \in E(GC^N)$) telle que $w' = Min(\{w'', (c, c'', w'') \in E(GC^N)\ et\ puits(c) < puits(c'')\})$ si c a au moins une voisine strictement plus petite qu'elle. Si plusieurs cuvettes voisines ont le poids minimum, celle dont le puits est le plus petit sera choisi.
- la cuvette voisine de c (avec $(c, c', w') \in E(GC^N)$) telle que $w' = Min(\{w'', (c, c'', w'') \in E(GC^N)\})$ sinon.

Le graphe de dépendance est alors construit en utilisant $Parent_\Omega(GC^N)$ qui représente la relation $\{(c, parent_\Omega(c)), \forall c \in BV^N\}$

En reprenant le MNT initial de l'exemple précédent (voir figure 5.4), on va illustrer la règle Ω dans la figure 5.7. Dans cet exemple, les points appartenant à Ω sont d'abord déterminés dans une première phase (ici tous les points de hauteur

0). Le calcul du graphe de flux est limité aux points qui ne sont pas dans Ω. Le résultat de la délimitation des cuvettes est illustré par la figure 5.6(b). Le calcul du graphe de connectivité GC^0 est donné dans la figure 5.6(c). On peut noter que les cuvettes qui sont au bord du MNT (c'est-à-dire $0, 3, 4, 6, 7$ et 8) sont connectés à Ω. La cuvette 1 est aussi connectée à Ω car elle est adjacente à des points qui sont dans Ω. Tous ces sommets sont oméga-marqués.



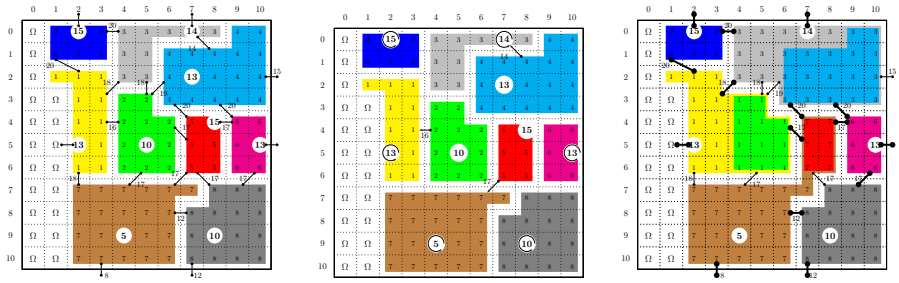
(a) MNT (b) Flux et Puits (c) GC^0

FIGURE 5.6 – Calcul des cuvettes et du graphes des connectivités avec règle Ω

Au premier niveau (voir **Figure 5.7**), chaque sommet s dans GC^0 cherchera son déversoir en se basant sur ses arêtes de poids minimum. **Figure 5.7(a)** nous montre que le sommet 1 du GC^0, la zone en jaune, a son arête de poids minimum vers Ω. C'est-à-dire que la cuvette 1 s'écoule vers la mer en réalité. La cuvette 1 ne sera pas réellement fusionnée à Ω dans la hiérarchie mais sera oméga-marquée. Il en est de même pour les sommets 0, 3, 6, 7, et 8. Par contre la cuvette 4, elle, va choisir la cuvette 3 comme déversoir ce qui est autorisé. Le sommet 5 (zone en rouge) dont la hauteur est de 15 s'écoule vers le sommet 7 (zone en marron) (la hauteur de puits égale à 5). Concernant le sommet 2 dont la hauteur est de 10, il est entouré par de voisins 1, 3, 4, 5, 7 qui sont tous oméga-marqués et de hauteurs supérieures à lui. Dans ce cas l'arête de poids minimum partant du sommet 2 est choisie. On est ici dans le cas où un sommet est entouré de sommets oméga-marqués plus hauts que lui et où on sélectionne l'arête de poids le plus faible sans tenir compte de la hauteur des voisins. Le graphe de dépendance GD^0 calculé avec la règle Ω est donné dans la figure 5.7(b). La fusion entre les cuvettes à ce niveau est réalisée afin de créer les nouvelles cuvettes au niveau suivant (voir **Figure 5.7(c)**).

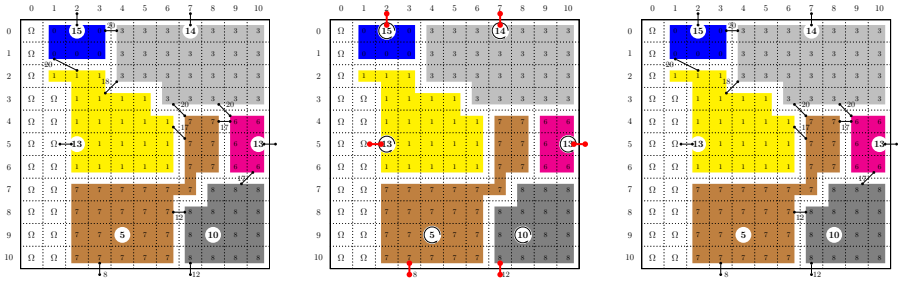
En se basant sur le graphe GD^0 et sur le nouveau graphe initial GC^1, nous calculons les nouvelles arêtes entre les cuvettes dans GC^1. Idem pour le calcul de la hiérarchie au deuxième niveau jusqu'à ce que toutes les cuvettes s'écoulent directement vers Ω.

On peut remarquer que le résultat final contient autant de cuvettes qu'il y avait de cuvettes oméga-marquées qui se déversent vers Ω à la première étape. Cela a permis d'isoler les cuvettes qui sont au bord du MNT et qui se déversent vers



(a) GC^0 (b) GD^0 (c) GC^1 initial

FIGURE 5.7 – Hiérarchie avec règle Ω : Première étape



(a) GC^1 (b) GD^1 (c) GD^2 init

FIGURE 5.8 – Hiérarchie avec règle Ω : Deuxième étape

l'extérieur du MNT (cuvettes 0, 3, 6, 7, et 8) d'une part et de délimiter le bassin versant du fleuve dont l'embouchure se trouve dans la cuvette 0 de graphe GC^0, d'autre part. Évidemment, ce MNT est tellement petit qu'on a l'impression que la règle Ω va segmenter le MNT en petits bassins versants. Dans la réalité ce phénomène va se produire uniquement au bord du MNT ce qui permettra d'isoler les cuvettes pour lesquelles nous n'avons pas assez d'information pour savoir à quel bassin versant elles appartiennent. Par contre cette règle va permettre de bien délimiter les bassins versants des grands fleuves comme nous le verrons dans le chapitre 7. Par ailleurs, on remarque que la règle Ω change pas la complexité de l'algorithme, ni ne modifie son potentiel de parallélisation.

5.3 Calcul des flots d'accumulation

5.3.1 Objectifs

Le calcul des flots d'accumulation permet de déterminer en chaque point du MNT, le nombre de points qui s'y déversent. Comme on l'a vu précédemment ce type d'information est très important par exemple dans l'étude des écoulements de polluants dans les cours d'eau mais aussi pour essayer de déterminer automatiquement les cours d'eau au sein du MNT. Dans ParaFlow, ce calcul est le dernier de la chaine et va utiliser les résultats obtenus lors des étapes précédentes. Plus par-

ticulièrement, on va utiliser le graphe des flux calculé à l'étape initiale et le MST calculé au cours de l'étape 2. Il va produire une grille 2D de la même taille que le MNT initial dont chaque point contiendra le flot d'accumulation du point du MNT correspondant.

Le calcul s'effectue en deux étapes. La première consiste à calculer les flots d'accumulation locaux aux cuvettes calculées lors de l'étape initiale. Ensuite le MST sera utilisé pour faire transiter l'eau d'une cuvette à l'autre jusqu'à la mer. Cette étape est appelée calcul des flots d'accumulation globaux.

5.3.2 Calcul des flots d'accumulation locaux

Pour faire ce calcul, l'entrée est la grille qui représente le graphe de flux $Flux(D, h)$ calculé pour le MNT (D, h). Le résultat sera une grille de domaine D appelée FA (pour Flots d'Accumulation). Toutes les cases de cette grille seront initialisées à une valeur ∞. Le problème pour cette phase est d'arriver à faire le cumul sans avoir à mettre à jour plusieurs fois les mêmes points lors du calcul car sur de très gros MNT les temps de calcul pourraient devenir trop importants. L'idée de l'algorithme que nous avons utilisé est la suivante : on parcourt l'ensemble des points de la grille FA. Pour chaque point p de FA si $FA(p) = \infty$ on va déterminer le flot d'accumulation du point p. Pour cela on va créer une pile de points non résolus (c'est à dire valant ∞) contenant initialement uniquement le point p. Cette pile va servir à connaitre la liste des points dont il faut déterminer le flot d'accumulation pour pouvoir déterminer celui de p. Donc pour chaque point p' non résolu qui est nécessaire au calcul de p, on va regarder parmi ses voisins dont l'image par la fonction *parent* est p' ,ceux qui ne sont pas résolus, si il y en a on les mets dans la pile, si il n'y en a pas on calcule le flot d'accumulation du point p' (et on dépile ce point).

L'inconvénient de ce type d'algorithme est la possible explosion de la taille de la pile (dans le plus mauvais des cas elle pourrait atteindre la taille du MNT). En fait, sur les MNT réels que nous avons traités la taille moyenne des cuvettes de l'étape initiale est de l'ordre de 50 points ce qui fait que l'algorithme est optimal en terme de parcours du MNT est ne nécessite pas de pile trop grande.

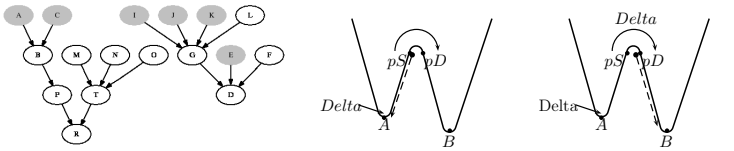
5.3.3 Calcul des flots d'accumulation globaux

Dans cette phase, on va utiliser la grille FA calculée précédemment ainsi que le MST calculé par ParaFlow appelé HT dans la suite (pour arbre hiérarchique). Dans **Figure 5.9(a)**, la cuvette A est une feuille dans HT, l'eau s'écoule alors à partir de la cuvette A vers son père B. Des flots d'accumulations du puits A sont propagés vers le puits B. Le flot d'accumulations de B est la somme de ses flots d'accumulation locaux (le nombre de pixels appartenant à la cuvette du puits B) et la somme des flots d'accumulations de tous les fils du B dans HT. La propagation se fait ensuite de B à son père P jusqu'à ce que nous atteignions le puits R (où le père de R correspond à Ω).

L'arbre hiérarchique HT impose un ordre afin de propager des flots d'accumu-

lations entre les puits. Nous considérons par exemple le HT dans **Figure 5.9(a)**. Supposons que les deux puits A et C aient fini de calculer leurs flots d'accumulations et qu'ils se propagent vers B, la propagation de B vers son père P sera ensuite réalisable. Par contre, dans la deuxième composante, les puits I, J et K sont marqués comme finis, mais L n'est pas encore fini pour son calcul. Cela signifie que nous n'avons pas assez d'informations pour continuer cette propagation et idem pour son père D.

Dans notre implémentation, cette propagation entre le puits A et son père B sera effectuée par une mise à jour dans la matrice des flots d'accumulation. Cette mise à jour consiste à modifier le flot d'accumulation des points $\forall a \in \pi_{[A,pS]}$ et $\forall b \in \pi_{[pD,B]}$ dans le graphe de flux où pS et pD sont des points de passage entre la cuvette de A et celle de B. Dans notre modélisation d'écoulement sur le graphe des flux, l'eau s'écoule selon le chemin de pS à A (noté $\pi_{[pS,A]}$) dans le sens inverse de la direction des flux calculé dans le graphe $Flux(D, h)$ (voir figure 5.9(b)), elle s'en va ensuite de pS à pD et elle s'écoule finalement sur le chemin de pD vers B, noté $\pi_{[pD,B]}$ (voir figure 5.9(c)). Les pixels $a \in \pi_{[pS,A]}$ porteront tous des flots d'accumulations de A, et les pixels $b \in \pi_{[pD,B]}$ accumulent leurs flots d'accumulations ainsi que ceux de A.



(a) Forêt de MST (b) ToA on $\pi_{[pS,A]}$ (c) ToB_LOC on $\pi_{[pD,B]}$

Une fois les flots d'accumulations globaux calculés, la détermination des cours d'eau se fait par un simple seuillage sur la matrice FA obtenu. C'est à dire que tous les points ayant un flot d'accumulation supérieur à un certain seuil seront considérés comme appartenant à un cours d'eau.

Chapitre 6

L'implantation parallèle de ParaFlow

Sommaire

6.1 Calcul de l'étape initiale 74
6.1.1 Division des données 74
6.1.2 Délimitation parallèle des cuvettes 74
6.1.3 Calcul parallèle du graphe de connectivité 79
6.2 Calcul parallèle hiérarchique des bassins versants 82
6.2.1 Parallélisation de l'algorithme de Borůvka 83
6.2.2 Implémentation dans ParaFlow 84
6.3 Calcul parallèle des flots d'accumulations 88
6.3.1 Écoulement de l'eau à l'intérieur des cuvettes 88
6.3.2 Modélisation des cours d'eau entre les cuvettes 90

Dans ce chapitre, nous décrirons les détails de l'implémentation en parallèle des différents algorithmes décrits dans le chapitre précédent. Les différentes sections de ce chapitre vont reprendre chacunes des trois étapes de ParaFlow.

Nous présenterons dans le chapitre suivant les résultats expérimentaux que nous avons été obtenus en utilisant ParaFlow pour l'analyse de gros MNT. Ces résultats nous montrent aussi que ParaFlow a des avantages non seulement sur la puissance de calcul et mais aussi sur la cohérence avec le réalité.

6.1 Calcul de l'étape initiale

Cette étape est la plus importante et la plus coûteuse de la délimitation des bassins versants des grands fleuves. Puisqu'elle manipule le MNT initial qui peut contenir des milliards de points, chaque phase de cette étape est calculée en parallèle, en limitant les communications et en ne faisant aucun regroupement de données. Le MNT initial est distribué sur les processeurs participants au calcul avec une zone de recouvrement d'un pixel ce qui permet de limiter les communications. L'algorithme 10 indique la manière dont cette étape a été implémentée. Les sections suivantes vont donner les détails permettant de comprendre cet algorithme.

6.1.1 Division des données

Soit un MNT représenté par la gille 2D (D, h) et P l'ensemble des processeurs (ou nœuds) participant au calcul. On note $|P|$ le nombre des processeurs qui participent au calcul.

Le domaine global D de la grille $G(D, h)$ est partitionné en autant de sous-domaines disjoints D_i qu'il y a de processeurs. Ces sous-domaines seront de forme rectangulaire. À chaque sous-domaine D_i est associé ce que l'on va appeler un *zone d'extension* notée ${D_i}^+$. Cette zone d'extension contient tous les voisins des pixels qui appartiennent à D_i et qui ne sont pas éléments de D_i. Elle est définie de la manière suivante : ${D_i}^+ = \{p \in D | p \notin D_i \ et \ \exists q \in D_i \ t.q. \ p \in N_G(q)\}$. On peut noter que la taille de la zone d'extension pour un sous-domaine carré de coté n est de $4 \times (\sqrt{n} + 1)$.

6.1.2 Délimitation parallèle des cuvettes

Etant donné que le MNT a été distribué sur les processeurs, il se peut qu'une cuvette du MNT global soit répartie sur deux ou plusieurs processeurs. Dans ce cas, il est impossible de calculer des composantes connexes réelles sans accès aux données globales. Ce type d'accès prend beaucoup de temps et, afin d'éviter cela, l'idée de notre méthode consiste à d'abord calculer les cuvettes locales, puis après une phase de communication impliquant une petite quantité d'information, de finaliser le calcul pour obtenir le résultat global.

Cette étape va donc se décomposer en quatre phases. La première traite la mer et les zone sans données. La seconde va consister à déterminer la direction des flux. Elle sera suivi d'une phase d'étiquetage (ou délimitation) local des cuvettes sur chaque processeurs. Les phases 2 et 3 sont purement locales. Ensuite, après des échanges d'information, les cuvettes partagées sur plusieurs processeurs seront relabélisées afin d'obtenir un résultat global cohérent.

6.1.2.1 Traitement de la mer et de l'absence de données

Notre méthode débute en identifiant les zones marquées comme Ω dans les MNT. Remarquons que les zones d'absence de données sont aussi traitées comme la mer.

Tous les pixels dont la hauteur est inférieure à la hauteur de la mer (par défaut égale à zéro) ainsi que les pixels marqués en absence de données se verront attribuer une valeur prédéfinie (par défaut 0).

Ensuite un traitement simplifié mais similaire à celui décrit dans les sections suivantes sera effectué pour délimiter les zones de hauteur 0 et d'en déterminer leur taille. Nous ne décrirons pas ce traitement pour éviter les redondances avec ce qui va suivre mais, in fine, les zones de hauteur 0 seront classées en deux catégories suivant leur taille :

- Les zones dont la taille est supérieure à un certain seuil fixé à l'avance (1000 points par défaut) seront considérées comme étant la mer et seront intégrés dans Ω (voir section 5.2.4). Ces zones seront appelées *MER*
- Les zones dont la taille est inférieure au même seuil sont considérées comme des zones sans données et seront complètement ignorées dans le reste du calcul. Ces zones seront appelées *TROU*.

Dans les sections suivantes, nous ne parlerons plus de ces zones car elles ne modifient pas fondamentalement le principe de la méthode mais leur traitement risque de compliquer inutilement les explications.

6.1.2.2 Direction des flux

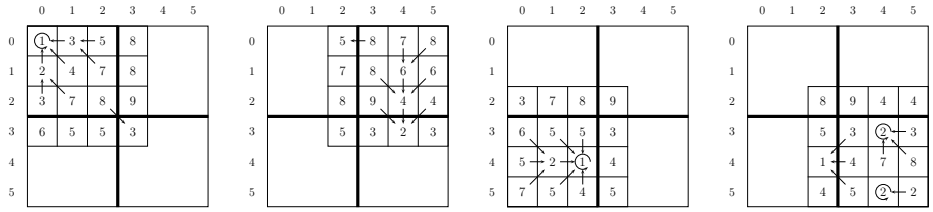
La phase suivante consiste à calculer la direction des flux représentée par le graphe $Flux(D, h)$. Ce type de calcul est très simple parce que la direction d'écoulements d'un pixel $p \in D_i$ est déterminée par ses voisins et, comme chaque processeur *Proc. i* possède son domaineD_i ainsi que la zone d'extension D_i^+. Un parcours linéaire de D_i est ainsi suffisant pour calculer le graphe $Flux(D, h)|_{D_i}$. Par ailleurs, aucune communication n'est nécessaire pour cette étape. Le résultat sera stocké dans une grille de domaine D^i où les valeurs encoderont la direction du flux. Cette grille est notée *parent* dans les algorithmes qui vont suivre.

On peut noter que le graphe $Flux(D, h)$ est l'union des $Flux(D, h)|_{D_i}$ calculés sur chaque processeur.

Figure 6.1 nous illustre les graphes des directions d'écoulements locaux aux quatre processeurs, les flèches entre des sommets représentent la fonction de *parent* à un pixel. Dans cet exemple, les deux pixels $(5, 4)$ et $(5, 5)$ (voir **Figure 6.1(g)**) forment un plateau. Grâce à l'ordre lexicographique, nous pourrons calculer le parent de deux pixels $parent((5, 4)) = parent((5, 5)) = (5, 4)$.

6.1.2.3 Étiquetage local des cuvettes

Dans un premier temps, on va étiqueter les puits (c'est-à-dire le point le plus bas) de chacune des cuvettes grâce à l'algorithme 4. Lors de cette phase on va pouvoir distinguer les puits primaires, c'est à dire ceux qui sont de vrais puits du MNT global (repérés par le fait que $parent(p) = p$ pour ces points) et les *puits secondaires* qui sont des points qui ont leur exutoire dans la zone d'extension. Ces puits secondaires obtiennent une étiquette provisoire (distribuée par la fonction $LabSec()$) qui sera



(d) Proc. 0 (e) Proc. 1 (f) Proc. 2 (g) Proc. 3

FIGURE 6.1 – Calcul parallèle des directions d'écoulements

remis en cause lors de la phase de relabellisation. Notons que chaque processeur distribue des étiquettes disjointes de celles de tous les autres processeurs.

Figure 6.1(d) nous montre qu'il existe un puits primaire $(0, 0)$ et un puits secondaire $(2, 2)$. Chacun a reçu une étiquette unique, respectivement 0 et 4 (voir **Figure 6.2(a)**).

Algorithme 4: Étiquetage parallèle des puits

Input : Set D_i of nodes on processor i and *parent* 2D-grid
Output : $listePuits_{primaire}$, $listePuits_{secondaire}$, *root* and *label*

```
1  foreach p ∈ D_i do
2      q_min = parent[p] ;
3      root[p] = q_min;
4      label[p] = ⊥;
5      if (q_min == p) then
6          q_min ⇒ listePuits_primaire ;
7          label[q_min] = LabPrim();
8      end
9      else if (q_min ∉ D_i) then
10         p ⇒ listePuits_secondaire ;
11         label[p] = LabSec();
12     end
13 end
```

Après avoir étiqueté les puits, nous allons propager le label de chaque puits à l'ensemble de la cuvette qui lui correspond. Pour cela on va utiliser la technique de *Compression de chemin* afin d'alléger la quantité d'accès aux données pendant les calculs locaux. Cette approche s'inspire de l'algorithme de Union-Find [Tarjan 1983, Cormen 1990]. Cette phase est décrite par les algorithmes 5 et 6.

Jusqu'à présent, tous les calculs peuvent localement être effectués au processeur *Proc. i* sans communication avec d'autres processeurs *Proc. j*. Ceci est optimal pour la scalabilité et l'accélération.

Algorithme 5: Recherche de la racine pour $p \in D_i$

Input : p, FD_i
Output : *root* grid updated for p

```
1 parent = FD_i[p] ;
2 while (parent ≠ p) and (parent ∈ D_i) do
3 |   p = parent;
4 |   parent = FD_i[p];
5 end
6 return p ;
```

Algorithme 6: Étiquetage du pixel p en utilisant "Compression de chemin"

Input : FD_i, $Puits_i$, p
Output : $Puits_i$ est MAJ

```
1 racine = findroot^{FD_i}(p) dans Algorithme 5;
2 aux = p;
3 while ((aux ≠ racine) and (Puits_i[aux] = ⊥)) do
4 |   aux_parent = FD_i[aux];
5 |   FD_i[aux] = racine;
6 |   Puits_i[aux] = Puits_i[racine];
7 |   if (aux_parent ∈ D_i) then aux=aux_parent;
8 |   else aux=racine;
9 end
```

6.1.2.4 Étiquetage global des cuvettes

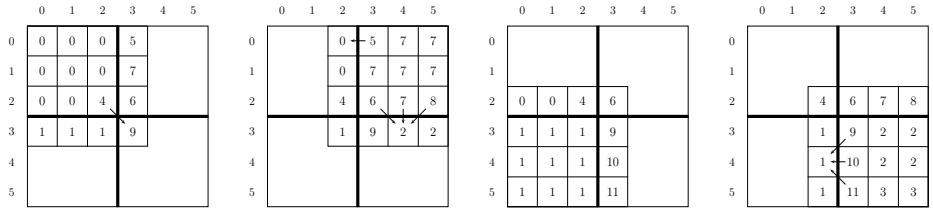
L'étiquetage global des cuvettes va consister à relabelliser les cuvettes dont le puits est un puits secondaire afin de leur attribuer l'étiquette de leurs véritables puits dans le MNT global. Cela va se dérouler en trois temps. Tout d'abord, les processeurs vont s'échanger les étiquettes attribuées aux points de la zone d'extension. Ensuite, chaque processeur va construire un graphe, appelé *graphe de dépendance local*, qui indiquera vers quels puits se déversent leurs puits secondaires. Enfin, après s'être échangé ces informations de dépendances entre puits, chaque processeur pourra déterminer l'étiquette finale de chacun de ses puits secondaires.

Échange des étiquettes de la zone d'extension

Chaque processeur va envoyer à ses voisins les étiquettes des points de son domaine qui correspondent à la zone d'extension de ses voisins. En retour, il va attendre les labels attribués par ses voisins aux points de sa propre zone d'extension.

Graphe de dépendance local

Chaque processeur va construire son graphe de dépendance LDG_i contenant les informations de dépendances de ses puits secondaires (c-à-d le label du point de la



(a) Proc. 0 (b) Proc. 1 (c) Proc. 2 (d) Proc. 3

FIGURE 6.2 – Calcul parallèle local des bassins versants aux *Proc. i*

zone d'extension vers lequel se déverse le puits secondaire).

Le graphe de dépendances LDG_i est simplement calculé en considérant les arrêtes $\{(sw, pw)|sw \in listePuits^i_{secondaires} \text{ et } pw = label[parent[sw]]\}$. La construction du graphe de dépendance est décrit dans l'**algorithme 7**.

La **figure 6.2** nous illustre les quatre graphes de dépendance locaux LGD_i correspondant au *Proc. i* $LDG_0 = \{4 \to 9\}$, $LDG_1 = \{5 \to 0, 6 \to 2, 7 \to 2, 8 \to 2\}$, $LDG_2 = \emptyset$, $LDG_3 = \{9 \to 1, 10 \to 1, 11 \to 1\}$.

Algorithme 7: Calcul du graphe de dépendances local LDG_i au *Proc. i*

Input : Results from previous local computations
Output : LDG_i local dependency graph for processor i
1 Upload of *parent*, *root* and *label* distant function values for node $p \in {D_i}^+$
2 **forall the** *($p \in listSecondaryWells$)* **do**
3 | $LDG_i[p] = label[parent[p]]$
4 **end**

Finalisation de la délimitation des cuvettes

Les graphes de dépendance locaux sont ensuite diffusés à tous les autres processeurs. Chaque processeur calcule le graphe de dépendance global GDG, noté $GDG = \bigcup_{0}^{|P|} LDG_i$, (on rappelle que $|P|$ est est le nombre de processeurs). Le GDG est un graphe dont les composantes connexes sont des arbres enracinés par un puits primaire.

Dans l'exemple de la **Figure 6.3(a)**, le graphe de dépendance global GDG après la diffusion des LGD_i est $GDG = \{4 \to 9, 6 \to 2, 7 \to 2, 8 \to 2, 9 \to 1, 10 \to 1, 11 \to 1\}$.

La résolution du graphe de dépendance global consiste à chercher vers quel puits primaire vont se déverser les puits secondaires. La technique de *Compression de chemin* est appliquée une deuxième fois afin de faire un lien vers un puits primaire unique pour toutes les étiquettes appartenant au même arbre dans le GDG. Ce

processus est décrit par l'**Algorithme 8** Rappelons que le processeur *Proc. i* ne résoudra qu'une partie du graphe *GDG* correspondant au LGD_i.

Algorithme 8: Recherche de la racine *rw* d'un puits secondaire *sw* dans GDG

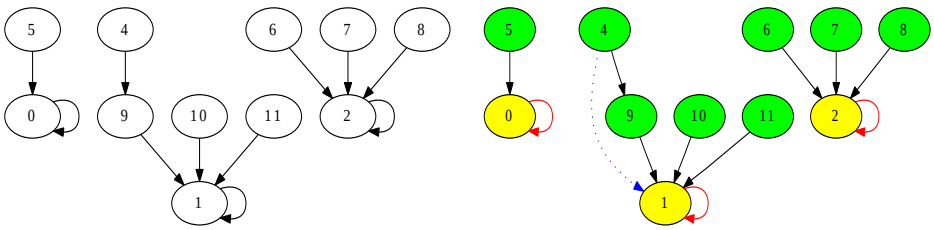
Input : GDG, un puits secondaire *sw*
Output : racine *rw* de *sw* dans GDG

```
1 parent = GDG[sw] ;
2 while (parent ≠ sw) do
3 |   sw = parent;
4 |   parent = GDG[sw];
5 end
6 return sw ;
```

Dans l'exemple de la **Figure 6.3(a)**, nous obtenons le graphe resolu suivant (voir **Figure 6.3(b)**). $GDG_{res} = \{4 \to 1, 6 \to 2, 7 \to 2, 8 \to 2, 9 \to 1, 10 \to 1, 11 \to 1\}$.

La dernière étape est purement parallèle. Elle consiste dans un parcours linéaire de tous les pixels associés aux puits secondaires. L'étiquette de ces pixels est remplacée par celle de puits primaires correspondant au *GDG* résolu.

Après cette phase, les processeurs vont diffuser la partie résolue de leur *GDG* afin de préparer l'étape suivante : le calcul du graphe de connectivité.



(a) GDG (avant de résoudre) (b) GDG (après avoir résolu)

FIGURE 6.3 – Graphe de dépendances GDG global

L'implémentation complète de la phase de délimitation des cuvettes est décrite dans **Algorithme 10**.

6.1.3 Calcul parallèle du graphe de connectivité

Une fois les cuvettes délimitées (et étiquetées), il va falloir construire le premier graphe qui sera donné en entrée de l'algorithme de Borůvka, c'est-à-dire le graphe de connectivité nommé GC^0 dans la section 5.2.2.

Les connectivités locales au processeur *Proc. i* entre des cuvettes sont déterminées par un simple parcours linéaire dans D_i (voir **Figure 6.4**). Lorsque deux pixels

Algorithme 9: Propagation de l'étiquette finale vers des puits secondaires

Input : secondary well *sw*, GDG
Output : *GDG* est MAJ entre *sw* et sa racine *rw* sur GDG

```
1 rw = findroot^GDG(sw) dans Algorithme 8 ;
2 parent = GDG[sw];
3 while (parent ≠ rw) do
4 |   GDG[sw] = rw ;
5 |   sw = parent ;
6 |   parent = GDG[sw];
7 end
```

Algorithme 10: Calcul parallèle des bassins versants

Input : The Grid mapped on processors
Output : *parent*, *root* and *label* 2D-grids updated

```
1  begin
2  |   foreach processor i do
3  |   |   Compute grid parent ∀p ∈ D_i and well label using (Algorithm 4) ;
4  |   |   foreach p ∈ D_i do
5  |   |   |   Algorithm 6 ;          // Labeling using Path Compression
6  |   |   end
7  |   |   Synchronization barrier;
8  |   |   LDG_i computation ;
9  |   |   Synchronization barrier;
10 |   |   Global exchange of LDG local dependency graphs;
11 |   |   GDG= ⋃_0^{|P|-1} LDG_i;
12 |   |   foreach sw ∈ GDG do
13 |   |   |   Algorithm 9 ;     // Propagating final label to secondary
   |   |   |   wells
14 |   |   end
15 |   |   foreach p ∈ D_i do
16 |   |   |   if IsSecond(label[p]) then
17 |   |   |   |   label[p] = GDG[label[p]]
18 |   |   |   end
19 |   |   end
20 |   end
21 end
```

adjacents n'appartiennent pas à la même cuvette, une nouvelle arête entre ces deux cuvettes est ajoutée dans ${GC^0}_i$ si elle n'était pas encore présente dedans. Le poids de l'arête correspond à la hauteur maximale entre ces deux pixels. Par contre, elle est

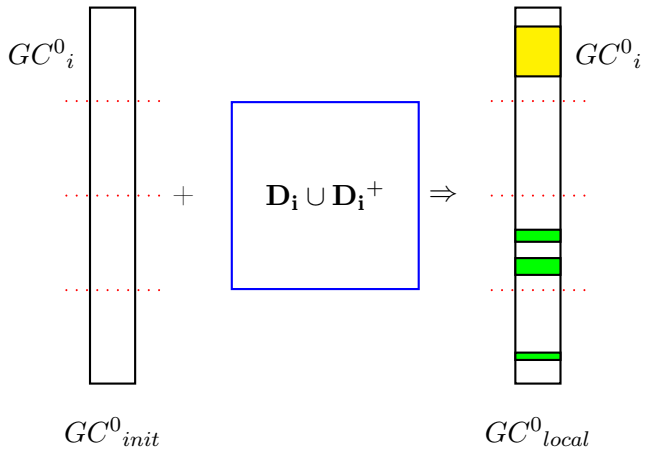


FIGURE 6.4 – Calcul de $GC^0{}_i$ dans un sous-domaine D_i

mise à jour en choisissant la connectivité dont le poids est minimal. Afin de prendre en compte les zones d'absence de données (présentées dans **Section 6.1.2.1**), une cuvette CB_p au bord du MNT a toujours une connectivité avec Ω. De plus, pendant les traitements du MNT $\forall p \in \{NODATA, MER\}$, les pixels p sont bien classifiés en Ω ou en $TROUS$ en fonction du seuil prédéfini. Ce sont des cuvettes particulières. Dans notre méthode, nous ne prenons pas en compte les connectivités entre $TROUS$ et d'autres cuvettes.

Algorithme 11: Finalisation du $GC^0{}_i$ au *Proc. i*

```
1 begin
2     for (∀GC⁰ᵢ ∈ GC⁰ | i ∈ [0, nbProcs)) do
3         GC⁰ᵢ au proc. i = ∑ GC⁰ᵢ aux proc. j (i ≠ j );
4     end
5 end
```

Après la phase d'échange entre tous les processeurs *Proc. i*, de nouvelles arêtes sont détectées et ajoutées à chaque *Proc. i*. Certains poids des connectivités sont aussi mis à jour (voir **Figure 6.5**). L'implémentation parallèle de cette phase est décrite dans **Algorithme 11**. **Algorithme 12** montrera notre implémentation parallèle afin de calculer le graphe de connectivités.

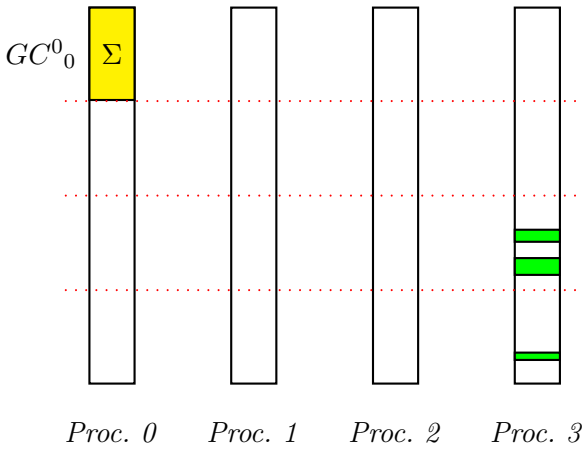


FIGURE 6.5 – Finalisation $GC^0{}_0$ correspondant au *Proc. 0*

Algorithme 12: Computing Connectivity of $G_1(D)$

```
1 foreach p_i in P do
2 |   Compute connectivity of G_1(Flow(D)|_{D_i}) ;
3 |   foreach j ∈ [1, |P|] do
4 |   |   Send to p_j the edge (v, v', w) ∈ G_1(Flow(D)|_{D_i}) such that either
  |   |   v ∈ C_j or v' ∈ C_j;
5 |   |   Get from each processor edges ended by elements of C_i;
6 |   end
7 |   Finalize connectivity of elements of C_i;
8 end
```

6.2 Calcul parallèle hiérarchique des bassins versants

Comme nous l'avons vu dans le chapitre précédent, la deuxième étape de ParaFlow consiste à calculer une forêt d'arbres couvrants de poids minimum en utilisant l'algorithme de Borůvka. Dans ParaFlow, nous proposons de calculer les bassins versants des rivières à l'aide d'un arbre couvrant de poids minimum du graphe de connectivité calculé dans **Section 6.1.3**. L'algorithme de Borůvka est particulièrement intéressant, non seulement de par son efficacité mais aussi parce que, par ses fusions successives de sommets, il calcule une hiérarchie de bassins versants (des petites cuvettes aux grands bassins des fleuves). Evidemment, nous avons repris les principes de la règle Ω (voir **Section 5.2.4**) afin d'éviter le problème de fusions prématurées de bassins versants disjoints dans la réalité(voir **Section 6.1.2.1**).

Nous allons d'abord expliquer les principes de la parallélisation de cet algorithme puis donner les détails de l'implémentation dans ParaFlow.

6.2.1 Parallélisation de l'algorithme de Borůvka

L'algorithme de Borůvka est un bon candidat pour la parallélisation. Nous donnons ici les principes de la méthode utilisée dans [Chung 1996].

Comme pour l'étape initiale de ParaFlow, le graphe est distribués sur les processeurs participant au calcul. Chaque processeur *Proc i* est en charge d'un ensemble de sommets D_i appelé domaine. Ce domaine est complété par une zone d'extension D_i^+ qui se définit comme dans la section 6.1.1. La figure 6.6(a) illustre la définition d'un domaine (en bleu sur la figure) et de son extension (en rose sur la figure).

Ensuite, l'algorithme se déroule de la manière suivante :

Étape 1 : Pour chaque sommet s de G, choisir une arête de poids minimum incident à s. Cette étape peut être réalisée localement si chaque processeur *Proc. i* possède tous les bords associés à ses sommets locaux. Cette étape est illustrée par la figure 6.6(b) où les arêtes sélectionnées sont dessinées en rouge.

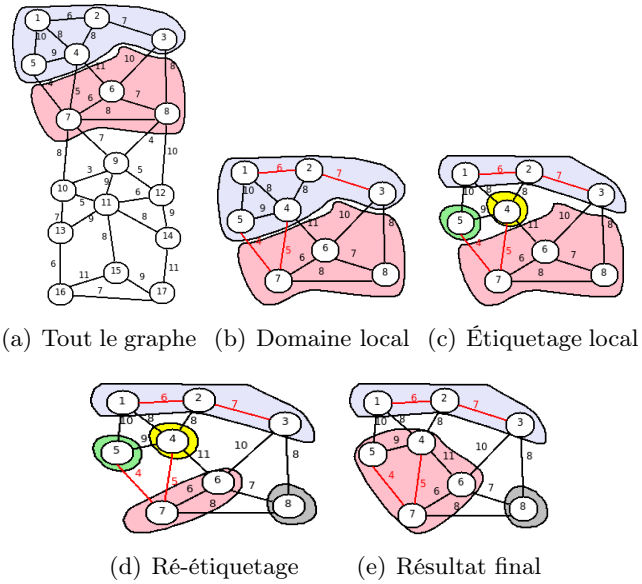
Étape 2 : Calculer les composantes connexes CC du graphe G en ne gardant que les arêtes sélectionnées afin de produire son $wMin$-mineur basée sur CC. Au cours de cette étape, chaque composante correspondant à un sommet dans le $wMin$-mineur, n'est affectée qu'à un seul processeur. Des communications entre les processeurs sont nécessaires pour cette étape. En fait, elle se déroule de la même manière que la délimitation des cuvettes de l'étape initiale, c'est-à-dire **(a)** délimitation des composantes connexes locales, illustrées figure 6.6(c) avec séparation entre puits primaires (en bleu sur la figure) et puits secondaire (en vert et jaune sur la figure) **(b)** communications des étiquettes de la zone d'extension (figure 6.6(d)) **(c)** ré-étiquetage des puits secondaires (figure 6.6(d)).

Étape 3.1 : Chaque processeur *Proc. i* calcule localement un graphe contracté en se basant sur des composantes connexes CC qui sont déterminées dans l'*Étape 2* à partir de son sous-graphe. Cette étape est totalement locale.

Étape 3.2 : Chaque processeur ramasse les arêtes incidentes aux composantes qui lui ont été assignées.

Afin de montrer les calculs parallèles des cuvettes en se basant sur l'algorithme de Borůvka, nous prenons un MNT, (voir **Figure 6.6(a)**), qui présente tout un graphe initial. Considérons un sous-domaine dans un processeur (voir **Figure 6.6(b)**). Ce sous domaine est une zone en bleu et sa zone étendue est en rose. Les arêtes sélectionnées sont marquées en rouge. Les sommets du domaine appartiennent à deux différentes composantes connexes de tout le graphe (voir **Figure 5.2(c)**). Le résultat de l'étiquetage local est introduit dans **Figure 6.6(c)**. Le processeur détecte que le sommet 1 est un puits primaire, tandis que 4 et 5 sont ceux secondaires, car ils s'écoulent vers 7 dans la zone d'extension.

Après l'échange des étiquettes des sommets de la zone d'extension (voir **Figure 6.6(d)**), chaque processeur *Proc. i* calcule son graphe de dépendances local et ensuite, il diffuse ce graphe local aux autres processeurs. Le graphe de dépen-



(a) Tout le graphe (b) Domaine local (c) Étiquetage local

(d) Ré-étiquetage (e) Résultat final

FIGURE 6.6 – Algorithme de Borůvka dans le calcul parallèle des cuvettes

dance global permet de connaître le puits primaire de chaque puits secondaire. **Figure 6.6(e)** nous illustre le résultat final après le ré-étiquetage des puits secondaires. Pour la prochaine itération du l'algorithme de Borůvka, le processeur sera en charge uniquement du sommet représentant la contraction des sommets 1, 2 et 3, car il possède tous ces sommets. Par contre la composante connexe à laquelle appartiennent les sommets 4 et 5 sera pris en charge par le voisin de ce processeur qui possède les sommets 6 et 7 car c'est celui-ci qui possède le puits de la composante connexe.

Dans [Chung 1996], il n'est clairement indiqué comment les cycles sont évités dans l'*Étape 1*. Dans cet article l'*Étape 2* est une boucle avec des synchronisations et des communications. Pour chaque sommet, un processeur effectue des communications pour trouver le sommet racine de la composante à laquel il appartient. Dans l'*Étape 3* toutes les arêtes sont mises à jour via des communications pour connaître les nouveau noms des sommets. Ensuite, les arêtes sont redistribuées, elles sont envoyées vers le processeur qui contient leur racine. Ce processeur fusionne les arêtes pour obtenir une partie du mineur global. Cette méthode nécessite beaucoup de communications car elles ont lieu avant la fusion des arêtes. Dans notre implémentation nous allons améliorer tous ces points.

6.2.2 Implémentation dans ParaFlow

Chaque itération de l'algorithme de Borůvka correspond à un niveau de la hiérarchie de graphes de mineur. Le calcul de la hiérarchie commence à partir du graphe de connectivités $GC^0(D,h)$ qui a été calculé dans l'étape précédente (voir **Section 6.1.3**). Chaque niveau N de la hiérarchie se calcule à partir des cuvettes du niveau $N-1$ et d'un graphe de connectivités GC^{N-1} dans lequel les puits qui ont des frontières communes sont connectés entre eux. À partir de ce graphe de connectivité,

on calcule un graphe de dépendance qui permet de connaître le trajet de l'eau entre les cuvettes. Grâce à un système de règles de déversement adéquat, nous arrivons à retirer un maximum d'informations du MNT sans le déformer. A la dernière étape de la hiérarchie nous obtenons les bassins versants réels du MNT. Tout ce processus correspond à calculer une forêt d'arbres couvrants de poids minimal à partir d'un graphe de connectivités entre des cuvettes déterminées.

Pour l'implémentation parallèle, notre algorithme parallèle de Borůvka subdivise le graphe $GC^0(D)$ en blocs de sommets et chacun est distribué à chaque processeur *Proc. i*. Cette distribution vise à minimiser le nombre d'arêtes externes reliant deux sommets appartenant à deux différents domaines. Afin d'optimiser les calculs, les sommets des arêtes externes sont ajoutés aux graphes des connectivités partiels GC_i^0 appartenant aux processeurs *Proc. i*, c'est la zone d'extension. Chaque *Étape i* en-dessous correspond à l'étape i^{me}de l'algorithme parallèle de Borůvka (voir **Section 6.2.1**).

Étape 1

Notre implémentation dans *Étape 1* consiste à sélectionner pour chaque sommet v une arête de poids minimal (v, v', w) en prenant les règles décrite section 5.2.3. Cette étape est réalisée au *Proc. i* en utilisant un parcours de tous les sommets dans son partiel du GC_0 et sans avoir de communication avec d'autres processeurs. Dans cette étape, si on a une cuvette CB qui possède une arête incidente à Ω, au lieu de fusionner la cuvette avec Ω, elle sera marquée comme $\overline{\Omega}$ et elle sera puits (c-à-d que $parent(CB) = CB$). Ce sommet sera conservé pendant le calcul de graphe contracté pour le niveau prochain. Avec cette règle simple, la construction de l'arbre minimal couvrant est dirigée de manière telle que sa racine est un sommet particulier Ω, ses premiers enfants sont des cuvettes dont les points de déversement sont Ω.

Comme nous l'avons déjà vu dans le chapitre précédent, cette méthode peut conduire aux configurations où une cuvette CB est entouré par de cuvettes $\overline{\Omega}$ dont la hauteur est supérieure à celle de CB. Afin de surmonter ces configurations, il suffit d'imposer que cette cuvette isolée CB sélectionne son déversoire parmi ses voisines $\overline{\Omega}$ sans tenir compte de la hauteur de celles-ci. **Algorithme 13** décrit comment le déversoir d'une cuvette, correspondant à un sommet dans un graphe de connectivité, est calculé à chaque niveau de la hiérarchie à partir de son arête de poids minimal.

Afin de modéliser la relation entre un bassin versant et son déversoir à chaque niveau, un graphe de dépendance local ${LDG^N}_i$ est aussi introduit à chacun des processeurs *Proc. i*. Tous ces graphes sont ensuite diffusés à tous les processeurs afin de calculer un graphe de dépendances global GDG^N. Afin d'obtenir le graphe de dépendances global résolu ${GDG^N}_{res}$, chaque processeur *Proc .i* résoudra une partie du GDG^N correspondant à son graphe local ${LDG^N}_i$ en utilisant la technique de Compression de chemin (voir [Tarjan 1983, Cormen 1990]). Cette partie sera ensuite diffusée une deuxième fois. Une autre solution peut viser à globalement résoudre le GDG^N à chacun des processeurs sans communication.

Algorithme 13: Calcul un exutoire d'un puits P au niveau N de la hiérarchie

Input :
- Un graphe de connectivités GC_i^N au Proc. i
- une arête de poids minium de P $\overrightarrow{(P,Q)}_{|(pS,pD,height)}$

Output :
- exutoire de P : exutoire(P)
- arêtes de poids minimum dans MST

```
1  begin
2      if (P is Ω̄) then exutoire(P) = P ;
3      else
4          con_min = (P,Q)⃗_|(pS,pD,height) ;
5          if (P > Q)_height then exutoire(P) = Q ;
6          else
7              if (Q is Ω̄) then exutoire(P) = Q ;
8              else
9                  if (P = Q)_height then exutoire(P) = Min{P,Q} | idPuits ;
10                 else exutoire(P) = P ;
11             end
12         end
13     end
14     if (exutoire(P) ≠ P) then paire (pS,pD) ⇒ MST ;
15     return exutoire(P) ;
16 end
```

Étape 2

L'étape 2 consiste à fusionner des cuvettes pour un niveau de la hiérarchie en calculant les nouvelles étiquettes pour les sommets d'un graphe de connectivité. Elle se base sur la détection des composantes connexes de ce graphe, en fonction des arêtes sélectionnées. Cette fusion modélisera ainsi l'écoulement entre des bassins versants niveau par niveau de la hiérarchie. Ceci est fait de la même manière que dans l'étape initiale, la seule différence est que l'on travail sur un graphe représenté par des listes d'adjacences au lieu d'une grille.

Étape 3

Cette étape a été implémentée comme la description dans la **Section 6.2.1**. Au niveau N dans la hiérarchie, chaque processeur diffuse toutes les connectivités des sommets du GC^{N+1} qui connectent vers un autre sommet extérieur de son sous-domaine. De cette façon, chaque processeur *Proc. i* est capable de connaitre toute la connectivité d'un sommet. Notez que seules les plus petites arêtes entre ces sommets sont gardées.

Algorithme 14: Calcul de la hiérarchie à un niveau N

Input : ${GC^N}_i$ un graphe de connectivités au processeur *Proc. i*
Output :
- ${GC^{N+1}}_i$ le mineur de ${GC^N}_i$ au processeur *Proc. i*
- les arêtes de poids minimum MST à chaque niveau de la hiérarchie.

```
1  begin
2      // Proc. i calcule l'exutoire (∀w ∈ GC^N_i)
3      for (∀w ∈ GC^N_i) do
4          calculer de exutoire_w de w (Algorithme 13) ;
5          LGD^N_i[w] = exutoire_w ;
6      end
7      Synchronization;
8      // nouveau sommet: ∀v ∈ LDG^N_i, ssi exutoire_v = v
9      Proc. i détectera de nouveaux sommets dans le mineur GC^{N+1}_i
10     Synchronization;
11     // Diffision de LGD^N_i entre des processeurs Proc. i
12     GDG^N_glo = Σ_{k=0}^{|P|-1} LGD^N_k ;
13     for (∀v ∈ GC^N_i) do
14         // chercher la racine root d'une cuvette v ∈ GDG^N_glo
15         root = findroot(v) sur le GDG^N_glo ;
16         // fusionner ∀v ∈ chemin_[v,racine) dans GDG^N_glo à root
17         pathcompression(v, root) sur GDG^h_glo ;
18     end
19     Synchronisation;
20     // diffuser GDG^N_i en deuxième fois entre des Proc. i
21     GDG^N_glo = Σ_{k=0}^{|P|-1} GDG^N_k ;
22     Synchronisation;
23     Proc. i calcule de nouvelles arêtes pour le mineur GC^{N+1}_i ;
24     Synchronisation;
25     // En échangeant des arêtes extérieurs et MAJ
26     Proc. i finalisera GC_i^{N+1} afin d'obtenir GCE_i^{N+1} ;
27 end
```

Algorithme 14 présente l'implémentation parallèle pour un niveau de la hiérarchie. Le calcul s'arrête si le mineur GC^{N+1} est égal au graphe de connectivités GC^N. Notez que la complexité d'itérations est la même que celle de l'étape initiale, sauf que la taille du graphe diminue fortement à chaque itération comme nous le verrons dans le chapitre suivant.

6.3 Calcul parallèle des flots d'accumulations

Cette section est consacrée pour la discussion du calcul des flots d'accumulation à partir des résultats précédents afin d'automatiquement extraire le réseau hydrographique à partir d'un MNT. L'idée consiste à déterminer d'abord les flots d'accumulations au niveau des cuvettes. Les flots d'accumulation des cuvettes sont ensuite propagés en prenant en compte le graphe de flux initial FD ainsi que le MST calculé, appelé HT dans la suite. Le HT a été obtenu dans la phase précédente. Il est enraciné par une cuvette particulière Ω. Cette propagation est commencée à partir des cuvettes qui sont marquées comme les feuilles sur HT et enfin s'arrête lors qu'elle atteint une cuvette de type $\overline{\Omega}$.

6.3.1 Écoulement de l'eau à l'intérieur des cuvettes

Dans cette section, nous présentons notre algorithme parallèle afin de déterminer les flots d'accumulation de tous les pixels p à partir des directions d'écoulement. C'est le nombre de pixels q, appartenant à la même cuvette que p, qui se déversent vers p. C'est ce que nous avons appelé les flots d'accumulation locaux. Une grille 2D FA, appelée la matrice de flots d'accumulation, sera le résultat de ce calcul. Plus formellement, étant donné un MNT de domaine D et son graphe de directions d'écoulement initial FD, le flot d'accumulation d'un pixel p (qui n'est ni dans une zone sans données, ni la mer) est le nombre de pixels p' tel qu'il existe un chemin de p' à p dans FD.

Initialement, les flots d'accumulation des pixels $p \in D_i$ ont été initialisés à la valeur ∞ ($FA[p] = \infty$).

Chaque processeur *Proc. i* possède son graphe des directions d'écoulement local noté FD_i. Rappelons que $\bigcup_{i=0}^{|P|-1} FD_i$ est exactement le graphe global FD dans tout le MNT. Dans le FD, pour chaque pixel $p \in D$, $fils(p)$ donne l'ensemble de ses voisins p' tels que p' s'écoule directement vers p et la fonction $feuille(p)$ est vraie si seulement si $fils(p) = \emptyset$.

Le FD_i est aussi considéré comme une forêt d'arbres enracinés par des puits et chaque arbre correspond à une cuvette. L'écoulement des pixels, dans une cuvette, peut être modélisé en se basant sur le FD_i. Notre implémentation parallèle du calcul des flots d'accumulation des cuvettes est décrite dans **Algorithme 15**. Rappelons qu'une cuvette correspondant à un puits peut être partagée sur plusieurs processeurs à cause de la distribution FD sur tous les processeurs. Les données locales au bord B_i d'un processeur *Proc. i* doivent être échangées avec les autres processeurs voisins *Proc. j*. Ces phases d'échanges font apparaître des inefficacités en terme de barrières globales entre les processeurs en fonction de la forme ou de la taille d'un bassin versant et du nombre de processeurs.

Afin d'éviter ces barrières, notre méthode distingue bien le calcul pour les pixels

Algorithme 15: Calcul local des flots d'accumulations locaux $\forall p \in D_i$

```
1  procedure computeFAL(D_i)
2  begin
3      // Computing FAL at border B_i of D_i
4      repeat
5          for ( ∀b ∈ B_i ) do
6              if (FA[b] = ∞) then
7                  computeFAL(b) in Algorithm 16;
8              end
9          end
10         Global synchronization;
11         Global Exchange of FA[B_i] for all processors ;
12     until (FA[p] ≠ ∞, ∀p ∈ B_i);
13
14     // computing FAL for all other pixels q in D_i
15     for (∀q ∈ D_i s.t. FA_i[q] = ∞) do
16         computeFAL(q) in Algorithm 16;
17     end
18 end
```

$p \in B_i$ des autres. L'accumulation de tous les pixels $p \in B_i$ dans le sous-domaine D_i est d'abord calculée pour concentrer les communications nécessaires uniquement dans cette phase. Après cela, chaque processeur réalise localement le calcul pour les autres pixels $q \in D_i$(implémenté dans **Algorithme 16**).

L'accumulation d'un pixel $p \in D_i$ est déterminée si et seulement si tous les fils $c \in FD_i$ de p, noté $fils(p)$, ont été déterminés. L'accumulation d'une feuille f dans le FD_i est égale à zéro. Naturellement, ce calcul pour tous les fils c est récursivement réalisé lorsque les fils de c ont été aussi déterminés. Notre implémentation élimine cette récursion à chaque itération en utilisant une pile des pixels *Stack* pour chacun des processeurs, afin de localement calculer des flots d'accumulation pour un pixel p donné.

Dans **Algorithme 16**, le pixel $p \in D_i$ est inséré dans la pile *Stack*. Ensuite une boucle assure que tant que l'accumulation de p n'est pas encore déterminée et peut être calculée (c'est-à-dire qu'elle ne dépend pas des voisins extérieures au sous-domaine du processeur), on effectuera les opérations suivantes : dépiler un pixel p' de la pile *Stack* et calculer la somme des flots d'accumulation de ses fils ($fils(p')$) si les $fils(p')$ ont tous été déterminés, sinon, p' est réinséré dans la pile *Stack*. Tous les fils c de p' dont les flots d'accumulation sont indéterminés appartiennent à un des trois cas suivants :

Cas 1 : Le pixel c est une feuille dans FD_i, son flot d'accumulation est alors déterminé à zéro et toutes les feuilles ne seront pas insérées dans la pile *Stack*.

Algorithme 16: Calcul local des flots d'accumulations pour un pixel $p \in D_i$

```
1  Stack = ∅;
2  p → Stack;
3  canCompute = true;
4
5  while ((Stack ≠ ∅) and (canCompute)) do
6  |  p' ← Stack;
7  |  if (∀ c ∈ fils(p'), FA_i[c] ≠ ∞) then
8  |  |  FA_i[p'] = 0;
9  |  |  for (∀ c ∈ fils(p')) do
10 |  |  |  FA_i[p'] = FA_i[p'] + FA_i[c] + 1;
11 |  |  end
12 |  end
13 |  else
14 |  |  p' → Stack;
15 |  |  for ((∀c ∈ fils(p') s.t. FA_i[c]=∞) and (canCompute)) do
16 |  |  |  if (feuille(c)) then FA_i[c] = 0;
17 |  |  |  else
18 |  |  |  |  if (c ∈ D_i) then c → Stack;
19 |  |  |  |  else canCompute = false;
20 |  |  |  end
21 |  |  end
22 |  end
23 end
24
25 if (not canCompute) then Stack = ∅;
```

Cas 2 : Le pixel c appartient au D_i, c est ensuite inséré dans la pile *Stack*.

Cas 3 : Le pixel c n'appartient pas au D_i, le calcul des flots d'accumulation pour p ne peut pas être continué. C'est le cas où une cuvette est partagée entre certains processeurs. Ce calcul de p ne sera pris en compte qu'après l'échange des flots d'accumulation des pixels $b \in B_i$.

6.3.2 Modélisation des cours d'eau entre les cuvettes

Dans cette section, nous décrirons comment obtenir un résultat global couvrant tout le MNT à partir du calcul au niveau des cuvettes (décrite dans **Section 6.3.1**). Le problème principal consiste à déterminer des cours d'eau principaux dans tout le MNT et à conduire ainsi ces cours afin de progressivement atteindre la mer. Dans **Section 6.2**, nous avons proposé de construire un arbre couvrant de poids minimum enraciné par un bassin versant particulier Ω. Cet arbre est appelé arbre hiérarchique HT dans lequel l'écoulement entre les cuvettes correspondant aux puits a été bien

précisé. À partir de cet arbre HT, nous pourrons déterminer un chemin d'écoulement d'une cuvette correspondant au puits vers le bassin versant Ω (correspondant à la mer).C'est donc le HT qui servira à calculer des flots d'accumulations globaux.

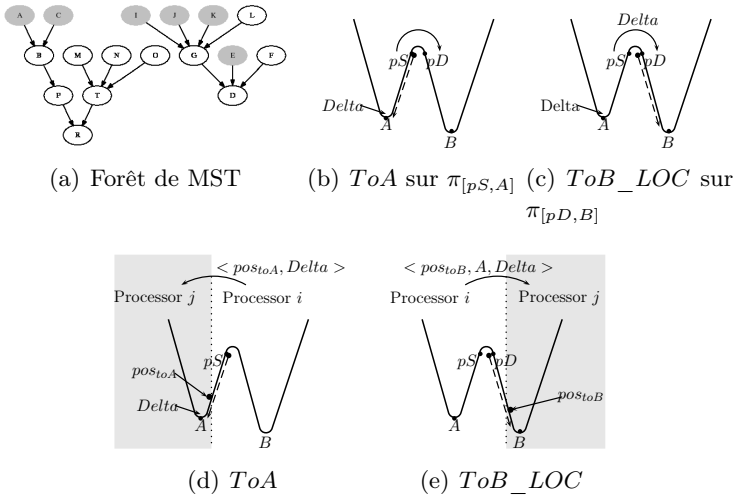
6.3.2.1 Écoulement dans l'arbre hiérarchique

Nous en venons maintenant à décrire l'écoulement entre des bassins versants dans l'arbre hiérarchique HT. Dans **Figure 6.7(a)**, le puits A est une feuille dans HT, l'eau s'écoule alors à partir du puits A vers son père B. Les flots d'accumulations du puits A sont propagés vers le puits B. Les flots d'accumulations du puits B sont la somme de ses flots d'accumulations locaux (le nombre de pixels appartenant à la cuvette du puits B) et la somme des flots d'accumulations de tous les fils du B dans HT. La propagation se fait ensuite de B à son père P jusqu'à ce que nous atteignions le puits R (où le père de R correspond à Ω ou bien le puits R est marqué comme $\overline{\Omega}$).

6.3.2.2 Mise à jour des flots d'accumulations des pixels

Nous allons décrire maintenant comment nous utilisons des flots d'accumulations au niveau des cuvettes afin de mettre à jours des flots d'accumulations au niveau des pixels comme décrit dans la section 5.3.3.

On rappelle que l'idée est de sortir l'eau accumulée au fond d'une cuvette, pour la déverser dans la cuvette parente (voir figure 6.7).



(a) Forêt de MST (b) ToA sur $\pi_{[pS,A]}$ (c) ToB_LOC sur $\pi_{[pD,B]}$

(d) ToA (e) ToB_LOC

FIGURE 6.7 – Propagation FA à partir d'une cuvette A to son parent B

Dans la propagation ToA (voir **Figure 6.7(b)**) décrite dans **Algorithme 17**, les flots d'accumulation des pixels a dans le chemin descendant $\pi_{[pS,A]}$ sont remplacés par une quantité $Delta$. La quantité $Delta$ est représentative des flots d'accumulation du puits A déterminés dans la phase de calcul local (voir **Section 6.3.1**) et dans le calcul sur l'arbre hiérarchique HT.

Algorithme 17: Propagation ToA d'une quantité $Delta$ pour $pos_{toA} \in \pi_{[pS,A)}$ au *Proc. i*

```
1  procedure ToA(PairA < pos_toA, Delta >)
2  begin
3      if (pos_toA ∉ D_i) then
4          pair_A = < pos_toA, Delta >;
5          pair_A ⇒ ListSendA_i;
6      end
7      else
8          mFA_i[pos_toA] = Delta;
9          parent_toA = Parent_i[pos_toA];
10         while (pos_toA ≠ parent_toA) and (parent_toA ∈ D_i) do
11             FA_i[parent_toA] = Delta;
12             pos_toA = parent_toA;
13             parent_toA = Parent_i[pos_toA];
14         end
15         if (parent_toA ∉ D_i) then
16             pair_A = < parent_toA, Delta >;
17             pair_A ⇒ ListSendA_i;
18         end
19     end
20 end
```

Dans notre implémentation parallèle, le domaine global D est subdivisé en sous-domaines D_i et chacun est distribué à un processeur *Proc. i*. Alors, ce chemin descendant peut ne pas entièrement appartenir au sous-domaine D_i. Dans ce cas-là, cette propagation est brisée lorsqu'elle atteint un pixel voisin $pos_{toA} \notin D_i$ (voir **Figure 6.7(d)**). Afin de surmonter ce problème, pour chacun des processeurs *Proc. i*, nous utilisons une structure de données $ListSendA_i$ afin de continuer cette propagation ToA à partir de pos_{toA} dans le sous-domaine D_j auquel pos_{toA} appartient. Au processeur *Proc. i*, une paire $paire_A < pos_{toA}, Delta >$ est immédiatement inséré à la liste $ListSendA_i$. La liste $ListSendA_i$ est ensuite échangée avec d'autres processeurs (voir **Figure 6.7(d)**).

Afin d'optimiser des données échangées entre les processeurs, chacun *Proc. i* envoie seulement au processeur *Proc. j* tous les paires $paire_A < pos_{toA}, Delta > \in ListSendA_i$ tels que pos_{toA} appartient au sous-domaine D_j. Et après cela, la propagation ToA sur le chemin descendant $\pi_{[pS,A]}$ peut être continuée avec une quantité $Delta$ à partir de pos_{toA} dans la prochaine itération.

Cette propagation ToA peut être répétée dans certains cycles (calcul local/échange global) jusqu'à ce que la liste $ListSendA_i$ de tous les processeurs *Proc. i* soit entièrement vide.

Nous allons maintenant décrire la propagation ToB_LOC du puits A au père

Algorithme 18: Propagation ToB_LOC pour tous les pixels $pos_{toB} \in \pi_{[pD,B)}$

```
1  procedure ToB_LOC(pair_B < pos_toB, A, Delta >)
2  begin
3      FA_i[pos_toB] = FA_i[pos_toB] + Delta;
4      parent_toB = Parent[pos_toB];
5      while ((parent_toB ≠ pos_toB) and (parent_toB ∈ D_i)) do
6          FA_i[parent_toB] = FA_i[parent_toB] + Delta;
7          pos_toB = parent_toB;
8          parent_toB = Parent_i[pos_toB];
9      end
10     if ((parent_toB ∉ D_i)) then
11         pair_B = < parent_toB, A, Delta >;
12         pair_B ⇒ ListSendB_i;
13         return true;
14     end
15     else
16         PassedSinks_LOC[A] = finished;
17         return false;
18     end
19 end
```

B. Cette propagation est réalisée via un point de passage $pD \in CB_B$ (voir **Figure 6.7(c)**), une quantité $Delta$ est ajoutée aux flots d'accumulation de tous les pixels appartenant au chemin descendant $\pi_{[pD,B]}$. Elle est décrite dans **Algorithme 18**. Nous commençons d'abord la propagation de ToB_LOC au point de passage $pD \in CB_B$. Ce traitement sera réalisé en suivant l'ordre des pixels sur le chemin $\pi_{[pD,B]}$. La phase finale dépend de l'état que nous obtenons.

État 1 : Dans la propagation ToB_LOC, $< pstoB$, A, $Delta >$ a atteint B. C'est-à-dire que la propagation de A vers B est finie. A est alors marqué comme fini pour la propagation à son père B. C'est la condition pour pouvoir continuer cette propagation vers la racine de A dans HT.

État 2 : La propagation ToB_LOC a atteint le pixel $pos_{toB} \in \pi_{[pD,B]}$ et $pos_{toB} \neq B$. pos_{toB} n'appartient pas au sous-domaine D_i du processeur *Proc. i* ($pos_{toB} \notin D_i$). C'est-à-dire que la propagation de A vers B ne peut pas être continuée. Des échanges de données au bord B_i ont besoin d'intervenir. Comme ci-dessus, nous avons utilisé une structure de données $ListSendB_i$ à chaque processeurs *Proc. i* afin de résoudre le problème où un chemin descendant $\pi_{[pD,B]}$ peut être partagé avec d'autres processeurs (voir **Figure 6.7(e)**). Une paire $pair_B < pos_{toB}, A, Delta >$ est insérée dans $ListSendB_i$. Après avoir échangé la liste $ListSendB_i$, cette propagation peut être continuée au processeur *Proc. j* à partir du pos_{toB}.

Algorithme 19: Propagation *ToB_GLO* d'une quantité *Delta* à partir de $pos \in CB_B$ vers $sink_{root}$ R de B sur HT

```
1  procedure ToB_GLO(pair_B < pos, A, Delta >)
2  begin
3      if ( ToB_LOC(pair_B)) then
4          P = getParent(B, pS_new, pD_new);
5          propagable_LOC = true;
6          ToRoot = PropagableToRoot(B);
7          while ((P ≠ root) and (propagable_LOC) and (ToRoot)) do
8              newDelta = FA_i[B];
9              ToA(pS_new, newDelta) in Algorithm 17;
10             pair_B = < pD_new, B, newDelta >;
11             if (pD_new ∉ D_i) then
12                 pair_B ⇒ ListSendB_i;
13                 propagable_LOC = false;
14             end
15             else
16                 if ( ToB_LOC(pair_B)) then
17                     propagable_LOC = false;
18                 end
19                 else
20                     B = P;
21                     ToRoot = PropagableToRoot(B);
22                     P = getParent(B, pS_new,pD_new);
23                     propagable_LOC = true;
24                 end
25             end
26         end
27     end
28 end
```

Algorithme 20: Calcul parallèle des flots d'accumulations globaux pour les pixels dans D_i

```
1  procedure computeFAG(D_i)
2  begin
3      for (∀ sink A ∈ HT_i) | (A is leaf) do
4          B = getParent(A, pS, pD);
5          Delta = FA_i[A];
6          ToA(< pS, Delta >) in Algorithm 17 ;
7          pair_B = < pD, A, Delta >;
8          if (pD ∉ D_i) then
9              pair_B ⇒ ListSendB_i;
10         end
11         else ToB_GLO(pair_B) in Algorithm 19;
12     end
13     allFinished = false;
14     while (not allFinished) do
15         Global synchronization;
16         ListRecvA_i = {pair_A < pS', Delta >∈ ListeSendA_j | pS' ∈ D_i};
17         ListRecvB_i = {pair_B < pD', X, Delta >∈ ListeSendB_j | pD' ∈ D_i};
18         ListSendA_i = ∅ ;
19         ListSendB_i = ∅ ;
20         for (∀ pair_A ∈ ListRecvA_i) do
21             ToA(pair_A) in Algorithm 17;
22         end
23         ListRecvA_i = ∅;
24         for (∀ pair_B ∈ ListRecvB_i) do
25             ToB_GLO(pair_B) in Algorithm 19;
26         end
27         ListRecvB_i = ∅;
28         allFinished_i = (ListSendA_i=∅) and (ListSendB_i=∅);
29         Global Synchronization;
30         allFinished = ∑_1^numProcs allFinished_k;
31     end
32 end
```

La propagation globale ToB_GLO des flots d'accumulations à partir d'un pixel $pos \in CB_B$ vers sa racine $root$ du puits B est décrite dans **Algorithme 19**. La fonction $PropagableToRoot(B)$ est utilisée afin de vérifier que la propagation peut être continuée pour le père de B dans HT. Le puits B peut se propager vers son père si seulement si tous les fils de B ont été marqués comme terminés. L'algorithme parallèle s'arrête lorsque les deux listes $ListSendA_i$ et $ListSendB_i$ de tous les processeurs *Proc. i* sont vides. Le résultat FA_i à chaque processeur *Proc. i* contient

les flots d'accumulations globaux des bassins versants des fleuves. L'implémentation parallèle, pour la mise à jour des flots d'accumulations des pixels, est décrite dans **Algorithme 20**.

Chapitre 7

ParaFlow : Résultats expérimentaux

Dans ce chapitre, nous allons présenter les résultats expérimentaux dans l'analyse de la Ligne de Partage des Eaux en utilisant ParaFlow afin de montrer son efficacité et sa scalabilité pour la manipulation de gros MNT. Nous avons réalisé les tests sur différents MNT dont certains nous ont été fournis par la société Géo-Hyd, à Orléans en France et d'autres ont été récupérés sur site de SRTM (http://www2.jpl.nasa.gov/srtm/).

Les expérimentations ont été réalisées sur la grappe d'ordinateurs du LIFO s'appelant MIREV, illustrée dans **Figure 7.1**. Elle est constituée de huit nœuds reliés par le réseaux Ethernet très haut débit (Ethernet gigabit). Chaque nœud est biprocesseur de type AMD Opteron Quad-Core 2,3 GHz avec 16 Go de mémoire vive. Ces nœuds utilisent le système Ubuntu 8.04.4 Hardy Heron. GNU C++ 4.2.4 et Open MPI version 1.4.1 sont aussi utilisés afin de tester ParaFlow.

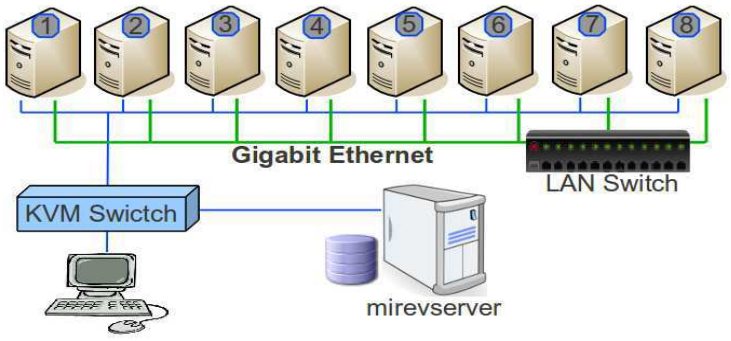


Figure 7.1 – Plate-forme MIREV au LIFO

Soit $|P|$ le nombre de processeurs utilisés. Le temps d'exécutions $T(|P|)$, pour l'analyse dans ParaFlow, est la différence entre des temps où le premier processeur commence l'algorithme et le dernier processeur se termine. Le temps d'exécution mesuré ne prend pas en compte les temps de chargement, de distribution et de sauvegarde de données. Pour chaque MNT on a effectué cinq fois le calcul et les résultats présentés sont la moyenne des cinq exécutions.

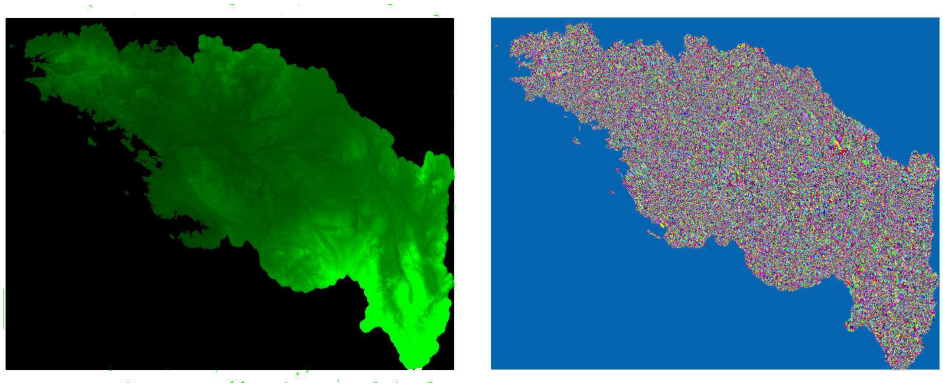
L'accélération relative de l'algorithme parallèle est alors mesurée par le facteur

$SP(|P|) = \frac{T_{ref}}{T_{|P|}}$, où T_{ref} est le temps d'exécution de référence pour le jeu d'essai. Pour les petits MNT cela correspond au temps d'exécution sur un seul processeur. Par contre sur les plus gros MNT, à cause de la limitation de la taille de mémoire principale d'un nœud, notre algorithme parallèle ne peut pas s'exécuter sur un seul nœud, dans ce cas le temps de référence sera le temps d'exécution sur 2 ou 4 nœuds suivant la taille du MNT.

7.1 Calcul parallèle des cuvettes

Le calcul parallèle des bassins versants au niveau des cuvettes dans ParaFlow est testé sur de différents MNT, tels que mnt100_sb50 avec la taille 3.980 × 5.701, MNT_LB avec la taille 10.086 × 14.786, Niger_2 avec la taille 36.002 × 54.002).

Figure 7.2(b) nous montre le résultat de ParaFlow dans la délimitation des cuvettes à partir du MNT_LB dont la taille est de 10.086 × 14.786. Il correspond à la Loire en France et contient 537.040 cuvettes ; chacune des cuvettes correspond à une zone colorée. La grande zone en bleu, correspond à la mer, appartient au bassin particulier appelé Ω. Comme nous l'avons déjà vu dans les chapitres précédents, à cause des bruits dans les données initiales, les résultats du calcul des cuvettes, donne une segmentation en une multitude très petites régions du MNT ce qui ne correspondent pas à la réalité géologique du terrain.



(a) MNT (b) Cuvettes à calculées

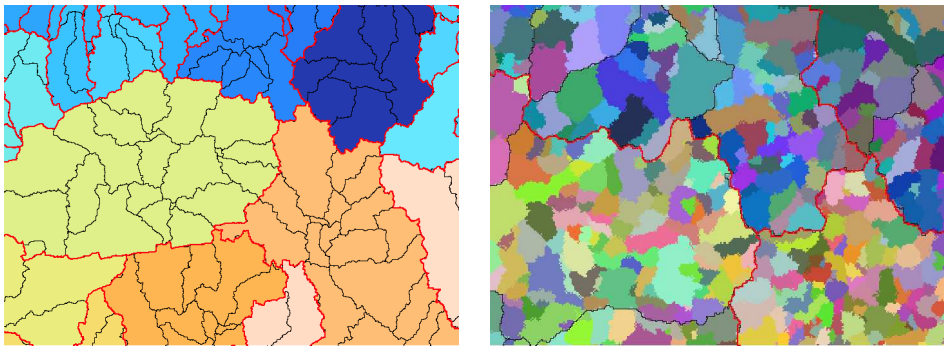
FIGURE 7.2 – La Loire en France avec des cuvettes calculées par ParaFlow

7.2 Calcul parallèle de la hiérarchie

Dans cette section, nous présentons les résultats expérimentaux en utilisant ParaFlow pour le calcul de la hiérarchie des bassins versants. Nous avons analysé

différents jeux de données. Le premier correspond à une partie de la France avec différentes résolutions, telles que MNT75m, MNT50m, mnt100_sb50m, MNT_LB, MNT_LB4. Le SRTM_Europe représente le continent européen. Niger_1, Niger_2, est une partie de l'Afrique dans laquelle s'écoule le fleuve Niger. SRTM_Asie représente une partie important de l'Asie.

La vérification de la précision de nos résultats sur ParaFlow n'est pas facile car on trouve principalement des cartographies classiques dans la littérature. De plus il est très difficile d'exécuter des logiciels de SIG tels que GRASS sur nos plus gros MNT. Nous ne pouvons comparer que notre résultat avec ceux de GRASS que lorsque la taille du MNT est petite. Dans la modélisation hydrologique, GRASS a classiquement proposé deux fonctions, *r.watershed* et *r.terraflow*, pour le calcul des bassins versants. Leurs résultats peuvent être considérés comme la subdivision d'un terrain en bassins versants. Ces bassins versants ne correspondent pas exactement à nos résultats puisque ces fonctions de GRASS ne sont pas implémentées pour l'analyse globale de la hiérarchie. Cette vérification a été effectuée en comparant les trois lignes de partage des eaux que produisent les trois méthodes. **Figure 7.3** donne une comparaison sur une partie de la Loire. Les lignes rouges sont obtenues par notre méthode ParaFlow et les noires (voir **Figure 7.3(a)** et **Figure 7.3(b)**) sont produites par *r.watershed*. Les zones colorées dans **Figure 7.3(b)** sont produites par *r.terraflow*. Dans cet exemple, les lignes de partage ont été comparées à 3 pixels près. La similarité est de plus de 90% entre ParaFlow et *r.watershed*, elle atteint un taux 98,6% entre ParaFlow et *r.terraflow*.



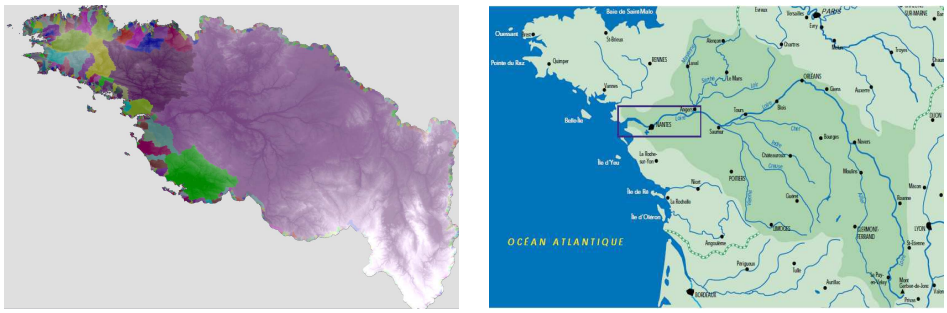
(a) ParaFlow et r.watershed (b) ParaFlow et r.terraflow

FIGURE 7.3 – Comparaison entre ParaFlow et GRASS sur une partie de la Loire

Nous avons comparé nos résultats en utilisant ParaFlow avec des cartographies correspondant à chaque MNT. Pour la visualisation des résultats, nous avons utilisé un autre outil, qui a été développé au LIFO, afin de permettre les visualisations en multiples couches, telles que le MNT, des bassins versants, les flots d'accumulations. Nos résultats sous format ascii sont aussi compatibles avec d'autres outils, tels que QGIS, GRASS pour la visualisation.

Figure 7.4 donne une comparaison sur le bassin versant de la Loire. Ces bassins

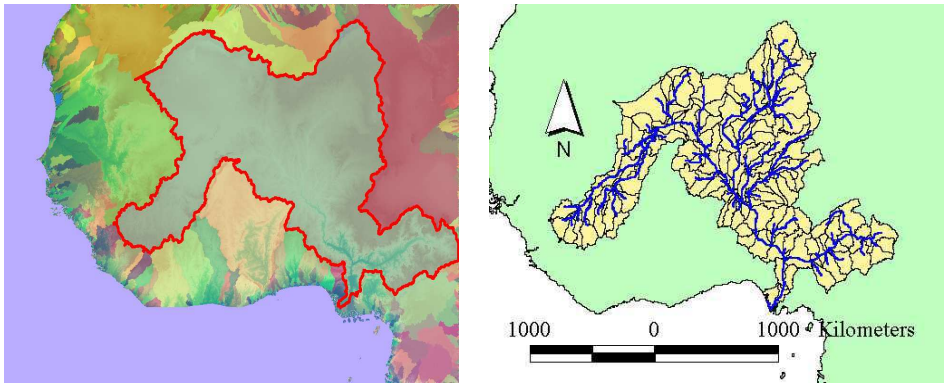
versants correspondent au MNT_LB dont la taille est de 10.086 × 14.786. Notre résultat (voir **Figure 7.4(a)**) est cohérent avec la réalité (voir **Figure 7.4(b)**). La grande zone en rose correspond au bassin versant à la Loire en France. Nous avons réalisé neuf niveaux de la hiérarchie à partir du graphe de connectivité initial qui contient environ 537.040 cuvettes (voir **Figure 7.2(b)**).



(a) Résultat de notre méthode (ParaFlow) (b) Bassin versant en réalité http://www.eau-loire-bretagne.fr

FIGURE 7.4 – Bassin versant de la Loire en France

Figure 7.5 nous montre aussi un autre résultat sur le bassin versant du Niger (Niger_1 dont la taille est de 30.001 × 42.001). La grande zone en vert, qui est entourée en rouge, est le bassin versant de Niger. On peut voir que notre résultat (voir **Figure 7.5(a)**) est très cohérent avec la réalité (ressource sur wikipedia **Figure 7.5(b)**). À partir d'un graphe de connectivités avec 48.129.608 cuvettes, le calcul de la hiérarchie pour Niger a été réalisé avec 24 niveaux de la hiérarchie.

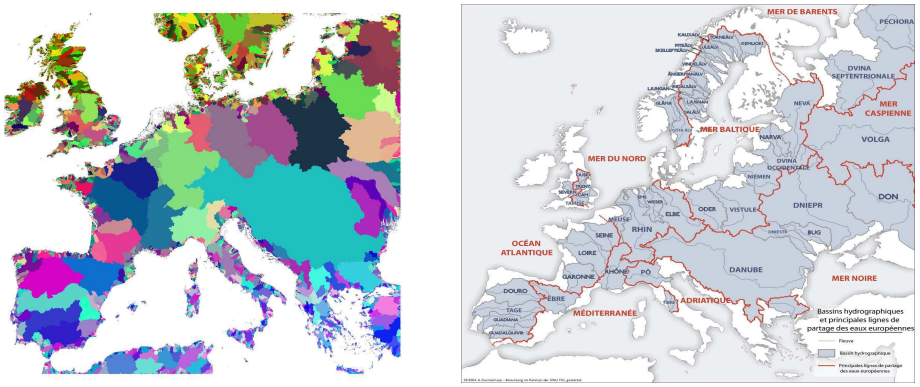


(a) Résultat de ParaFlow (b) Niger dans la réalité (ressource à wiki)

FIGURE 7.5 – Bassin Versant du Niger

La figure 7.6 concerne les bassins versants en Europe. La délimitation automatique des bassins versants, à partir du MNT SRTM_Europe (taille 30.001 × 48.001),

après avoir calculé des cuvettes, nous avons construit son graphe de connectivité dont le nombre de sommets est de 19.694.358. Les calculs de la hiérarchie sont ensuite réalisés jusqu'à 18 niveaux. Nous avons pu comparer les bassins versants, des grands fleuves en France (la Seine, la Loire, la Garonne, le Rhône, (voir Figure 7.6), avec ceux automatiquement détectés par ParaFlow. Nous avons pu constater que nos résultats (voir Figure 7.6(a)) sont très cohérents avec la réalité (voir Figure 7.6(b)).



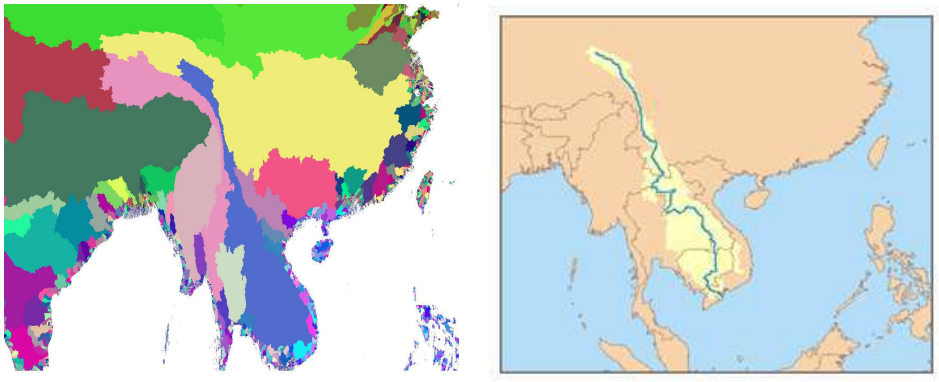
(a) Bassins versants en Europe sur ParaFlow (b) Bassins versants réels (*Source WikiPedia CC*)

FIGURE 7.6 – Les bassins versants en Europe (SRTM Europe 90m)

La **figure 7.7** présente les bassins versants en Asie dont la taille du MNT est de 60.001 × 90.001. Dans cette figure, nous mettrons l'accent sur le bassin versant du Mékong, la grande zone en violet. Le calcul de la hiérarchie est réalisé sur 25 niveaux à partir d'un graphe de connectivités dont le nombre de sommets est de 124.688.171 où chaque sommet correspond à une cuvette. Le temps d'analyse (en prenant en compte uniquement le temps d'exécution) en utilisant 8 machines dans la grappe est environ 9055 secondes (voir Table 7.3) c'est-à-dire 2 heures 30 minutes. Ce temps semble raisonnable pour manipuler un très gros MNT dont la taille est de 60.001 × 90.001.

La **table 7.1** montre que le nombre de sommets des graphes qui sont calculés par l'algorithme de Borůvka dans la hiérarchie, diminuent exponentiellement. La dernière ligne nous donne la taille du graphe final de la hiérarchie. Elle nous montre aussi qu'après 4 à 7 étapes du calcul de la hiérarchie, la taille du graphe de connectivité est du même ordre que celle du graphe final. Les dernières étapes fusionnent de moins en moins les sommets à cause de la règle Ω qui empêche la fusion des sommets afin d'éviter des résultats erronés.

Concernant le volume de communications entre les processus, lors de l'étape initiale de ParaFlow, on peut constater que la distribution par bloc les réduit de manière drastique. Par exemple, lorsque SRTM_Asie dont la taille est de 60.001



(a) Résultat de notre méthode ParaFlow (b) Bassin versant de Mékong en réalité (*Source WikiPedia CC*)

FIGURE 7.7 – Bassins versants en Asie : La détaille du Mékong

× 90.001, est distribué sur 64 processeurs, le domaine de chaque processeur est d'environ 84 millions de pixels, la zone d'extension est d'environ 37.000 pixels ce qui est le volume de données que chaque processeur échange avec ses voisins. La deuxième communication consiste à diffuser le graphe de dépendance local dont la taille est inférieure à la zone d'extension. Pour le MNT SRTM_Asie, la taille du graphe de dépendance global ne excède pas la valeur $taille = (60.001 + 90.001) \times 7 = 1.050.014$. En moyenne, chaque processeur va diffuser son graphe de dépendance local dont la taille est égale à 1.050.014 / 64 (environ 16.400 sommets). Pendant le calcul des connectivités du premier graphe de la hiérarchie, la communication entre les processus aura aussi lieu sur une quantité de données comparable avec des données initiales. Remarquons que cette phase n'est pas une diffusion, un seul échange est effectuée en parallèle sur les processus.

Nous trouvons que le nombre de sommets du graphe au premier niveau est entre 1.5% et 4% de la taille initiale du MNT, la communication diminue ainsi dans le même ordre. Idem pour les autres étapes dans la hiérarchie. De plus, dans l'étape initiale, la cartographie des sommets d'un graphe sur les processeurs est directement liée à la localisation spatiale 2D ce qui renforce la localité des données.

Le temps d'exécution en fonction du nombre de processeurs est donné dans **Table 7.2** pour de petits MNT et dans **Table 7.3** pour de gros MNT (tels que Niger_1, Niger_2, SRTM_Europe et SRTM_Asie). Ces temps ne prennent en compte ni les chargements et ni les sauvegardes des données. Mais, elles contiennent tous les temps d'identification les zones appartenant à Ω, de calcul des cuvettes et de la hiérarchie.

Remarquons, à cause de la limitation de la taille de la mémoire principale d'un nœud, que notre algorithme parallèle ne peut pas s'exécuter sur une seule machine sur de gros MNT (tel que Niger_2 avec la taille 36.002 × 54.002, ou SRTM_Asie 60.001 × 90.001). Dans ces cas-là, nous avons choisi T_{ref} comme le temps d'exécutions de notre algorithme parallèle sur 4 machines (pour Niger_1, Niger_2 et

level	**SRTM_Asie**	**SRTM Europe**	**Niger_2**	**Niger_1**
0	5.400.150.001	1.944.180.004	1.260.072.001	1.440.078.001
1	124.688.171	48.145.249	48.975.687	19.604,358
2	26.152.246	11.613.414	12.034.031	4.341.387
3	7.986.087	3.781.878	3.948.759	1.578.189
4	3.084.592	1.436.682	1.508.167	882.791
5	1.565.476	634.618	665.150	689.636
6	1.041.208	332.798	334.471	630.937
7	847.910	212.107	214.688	611.810
*	717.780	115.556	115.560	601.336

TABLE 7.1 – Nombre de sommets à chaque niveau de la hiérarchie

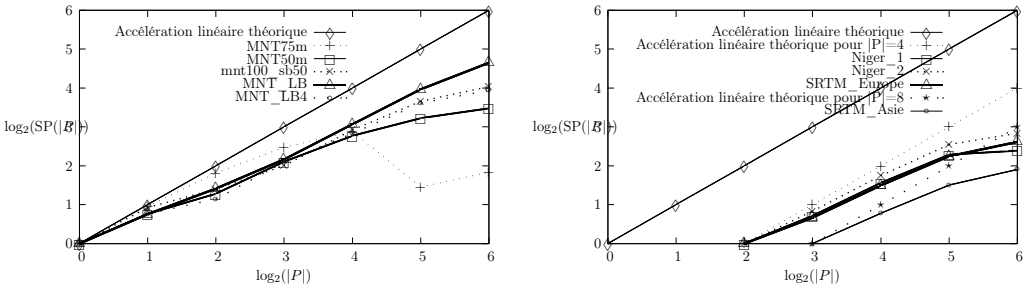
$\|P\|$	**MNT75m**	**MNT50m**	**mnt100_sb50m**	**MNT_LB**	**MNT_LB4**
	784 x 1.058	2.000 x 2.000	3.980 x 5.701	10.086 x 14.786	7.216 x 10.106
1	4,714	25.064	148.078	1025.768	472.632
2	2.426	14.749	77.516	605.845	277.919
4	1.336	10.284	55,702	383.874	213.112
8	0.844	5.827	36,706	228.195	110.138
16	0.642	3.658	19,485	122.110	66.065
32	1.714	2.646	11,676	65.630	38.693
64	1.320	2.241	8,990	40.847	30.715

TABLE 7.2 – Temps d'exécution (en secondes) pour de petits MNT

$\|P\|$	**Niger_1**	**Niger_2**	**SRTM_Europe**	**SRTM_Asie**
	30.001 x 42.001	36.002 x 54.002	30.001 x 48.001	60.001 x 90.001
4	3.775,796	6.225,885	3.483,853	
8	2.292,847	3.470,633	2.197,380	9.054,824
16	1.277,718	1.851,728	1.230,305	5.270,098
32	771,357	1.064,216	735,557	3.188,205
64	718,563	871,658	568,332	2.413,184

TABLE 7.3 – Temps d'exécution (en secondes) pour de gros MNT

SRTM_Europe), ou sur 8 machines (pour SRTM_Asie). Ces courbes d'accélération linéaires commencent à la coordonnée (2,0) et à la coordonnée (3,0) respectivement pour quatre machines et huit machines. Elles sont aussi parallèles avec la courbe d'accélération linaire classique (voir **Figure 7.8(b)**).



(a) Petits MNTs (b) Gros MNTs

FIGURE 7.8 – Accélération relative en fonction du nombre de processeurs ($|P|$)

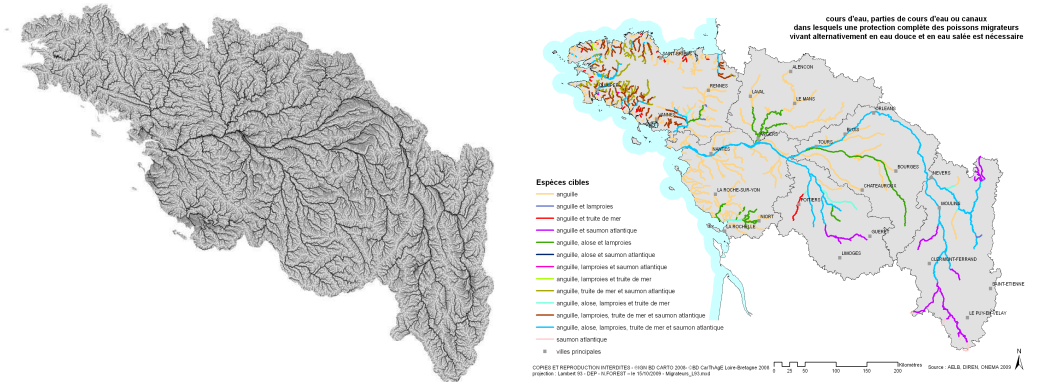
Regardons **Figure 7.8(a)** pour de petits MNT, la courbe d'accélération relative est très proche de la celle linéaire au début des courbes. Cela nous signifie que l'accélération augmente linéairement. Lorsque le nombre de processeurs $|P|$ augmente, l'accélération relative en fonction de $|P|$ va se réduire, par exemple pour MNT75m à partir de 32 processeurs. Cela est dû au fait que les communications entre des processeurs deviennent plus coûteuses que des calculs locaux.

Pour de gros MNT (tels que 1.9 G-points avec Niger_2 ; ou 5.4 G-points avec SRTM_Asie), l'accélération relative augmente linéairement (voir **Figure 7.8(b)**). Cela nous illustre une bonne scalabilité de notre méthode (ParaFlow) afin de calculer des bassins versants de fleuves au niveau des continents en appliquant une implémentation parallèle de l'algorithme de Borůvka.

7.3 Calcul parallèle des flots d'accumulations

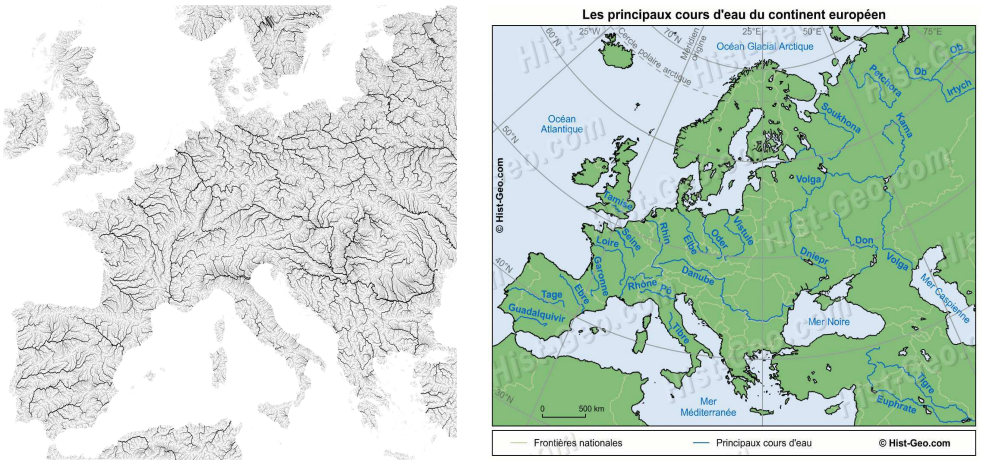
Dans cette section, nous présentons nos résultats sur le calcul des flots d'accumulations avec ParaFlow en les analysant non seulement sur la précision et mais aussi sur la puissance de calcul de notre méthode ParaFlow.

Figure 7.9(a) et **Figure 7.10(a)** nous montre respectivement notre résultat qui porte sur les cours d'eau principaux pour la Loire en France (MNT_LB) et pour le continent européen (SRTM_Europe). Les cours d'eau sont aussi extraits en se basant sur les flots d'accumulations. En comparant entre notre résultat et la réalité non seulement pour la Loire (entre **Figure 7.9(a)** et **Figure 7.9(b)**), mais aussi pour les cours d'eaux en Europe (entre **Figure 7.10(a)** et **Figure 7.10(b)**) nous avons trouvé que ParaFlow présente de bons exutoires pour les cours d'eau.



(a) Dans notre méthode (ParaFlow) (b) Source http://www.eau-loire-bretagne.fr

FIGURE 7.9 – Cours d'eau principaux en Loire-Bretagne (France)

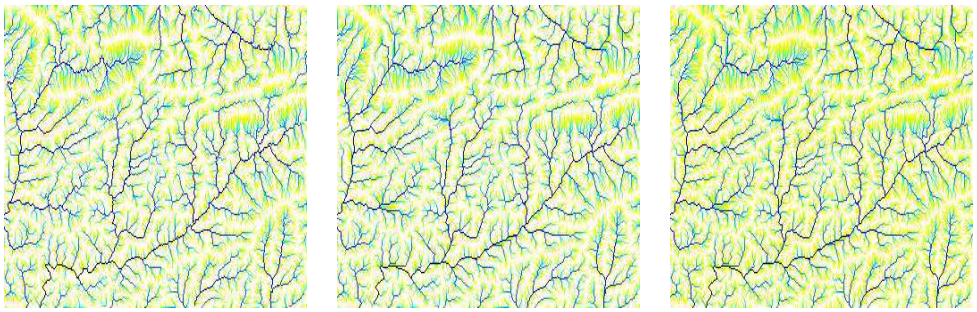


(a) Flots d'accumulations en Europe (b) Source http://www.hist-geo.com

FIGURE 7.10 – Cours d'eau principaux du continent européen

Nous avons aussi comparé la qualité du résultat de notre méthode (ParaFlow) avec d'autres systèmes (plateformes libre TerraFlow et commerciale ArcGIS). Les trois méthodes nous donnent toutes presque le même réseau hydrographique (voir **Figure 7.11**). Nous avons comparé au niveau de pixel, les trois matrices de flots d'accumulations qui sont produites par les trois méthodes. Nous avons constaté qu'il y a moins de 3% de différences entre ParaFlow et les deux autres méthodes.

Le temps d'exécution, afin d'obtenir les flots d'accumulations pour chaque MNT est donné dans **Table 7.4**. Tous ces temps ne prennent en compte ni le chargement ni la sauvegarde de données. Le calcul est d'abord commencé, à partir d'un MNT, pour la délimitation de toutes les cuvettes et pour la construction d'un graphe de connectivités. Les calculs de la hiérarchie, afin de modéliser l'écoulement des eaux



(a) ParaFlow (b) ArcGIS (c) TerraFlow

FIGURE 7.11 – Flots d'accumulations d'un MNT sur trois plateformes

entre les cuvettes, sont ensuite réalisées jusqu'à ce que chaque cuvette se trouve un exutoire global. En se basant sur les directions d'écoulement, les débits de tous les pixels p au niveau local sont déterminées par le nombre de pixels q dans une même cuvettes que p qui se déversent vers p. À partir d'un arbre couvrant de poids minimum, dont la racine est la cuvette particulière Ω, déterminé dans les calculs hiérarchiques, nous continuons à modéliser les directions d'écoulements des cuvettes en hiérarchiquement propageant les débits entre les cuvettes dans cet arbre à travers des points de passages à leur frontière.

	MNT50m	**MNT_LB4**	**MNT_LB**	**Niger_2**
1	176,041	724,033	1.085,136	
2	90,991	418,475	635,858	
4	63,498	303,956	386,441	10.046,061
8	38,949	159,737	245,779	4.972,299
16	21,517	96,668	135,382	3.003,389
32	13,987	57,582	76,083	1.964,106
64	24,461	38,316	62,512	1.584,681

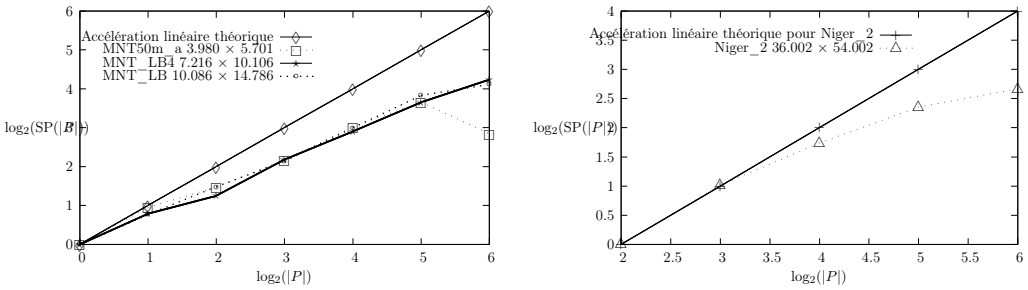
TABLE 7.4 – Temps d'exécution de calcul parallèle des flots d'accumulations (en secondes)

Remarquons qu'à cause de la limitation de mémoire d'une machine, ParaFlow ne peut pas alors analyser le Niger_2 (taille 36.002×54.002) sur une seul machine. Dans ce cas-là, ParaFlow analysera le Niger_2 en utilisant 4 processeurs sur 4 nœuds différents. La courbe d'accélération linéaire sera alors commencée à la coordonnée (2,0) et sera parallèle avec celle linéaire théorique classique (voir **Figure 7.12(b)**).

Regardons sur **Figure 7.12(a)**, lors de l'analyse de MNT50m, de MNT_LB4 et de MNT_LB, ses accélérations relatives sont proches de l'accélération linéaire.

Pour de gros MNT (par exemple Niger_2), l'accélération relative augmente linéairement (voir **Figure 7.12(b)**). Cela signifie que notre plate-forme ParaFlow

sera scalable pour de petits MNT et aussi de gros MNT.



(a) Petits MNT (b) Gros MNT

FIGURE 7.12 – Accélération relative en fonction du nombre de processeurs $|P|$

La **table 7.3** nous montre des comparaisons le temps d'analyse pour MNT_LB4 (7.216×10.106) entre ParaFlow et TerraFlow/GRASS (Geographic Resources Analysis Support System). Le temps d'analyse pour ParaFlow dans cette table prend en compte les chargements et les sauvegardes de données pour être équitable avec TerraFlow.

$\|P\|$	**ParaFlow**	**TerraFlow**
1	837,753	2.047
2	518,859	
4	396,683	
8	245,370	
16	181,810	
32	173,209	

TABLE 7.5 – Comparaison de temps d'exécution (en secondes) du MNT_LB4 entre TerraFlow et ParaFlow

Sur MNT_LB4, notre plate-forme ParaFlow est plus rapide que TerraFlow sur un seul processeur et elle est 10 fois plus rapide que Terraflow sur 8 processeurs.

L'implémentation de TerraFlow est séquentielle, l'analyse de MNT_LB4 avec TerraFlow est réalisée sur un seul processeur. En conséquence, la durée d'analyse en fonction du nombre de processeurs se présente sur le graphique sous la forme d'une ligne horizontale (voir **Figure 7.13**). Par contre ParaFlow analyse le MNT_LB4 en parallèle et la courbe de son temps décroît en fonction du nombre de processeurs (jusqu'à 64 processeurs) (voir **Figure 7.13**). Si toutefois la taille des données devient plus grande que la mémoire principale il faut utiliser plus de machines dans la grappe. TerraFlow qui adopte une démarche out-of-core n'a pas ce défaut.

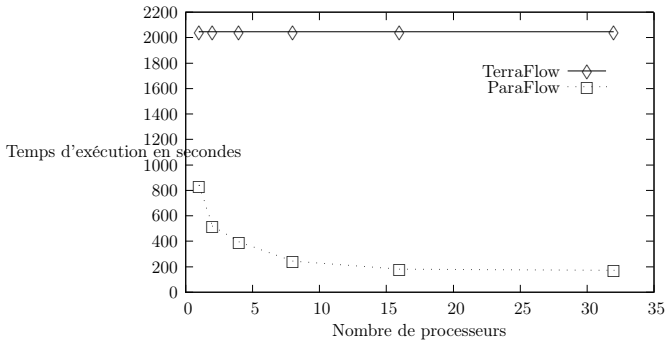


FIGURE 7.13 – Comparaison du temps d'exécution pour MNT_LB4 entre TerraFlow et ParaFlow

Quatrième partie

PARAFLOW-OOC : ANALYSE PARALLÈLE DE LPE AVEC OUT OF CORE

Chapitre 8

Introduction de ParaFlow-OOC

La méthode d'analyse des MNT volumineux que nous avons proposée repose sur l'hypothèse que nous disposons d'assez de noeuds sur la machine parallèle pour pouvoir faire tenir en mémoire toutes nos structures de données. Si ce n'est pas le cas nous postulons que l'utilisateur peut faire appel à une machine plus puissante. Evidement ce n'est pas toujours possible en pratique et il est parfois plus simple d'accepter un calcul plus lent si l'on peut l'effectuer sur une machine plus modeste mais disponible. Il est intéressant dans ce cas d'offrir la possibilité de sauvegarder temporairement les données sur une mémoire externe sans s'en remettre directement au mécanismes de pagination ce qui dégraderait considérablement les performances. Le transfert de données, entre le disque dur et la mémoire principale, est souvent le goulot d'étranglement dans l'efficacité des algorithmes lors du traitement des données très volumineuses (voir [Danner 2007]). Nous avons étendu notre methode afin qu'elle puisse exploiter au maximum la mémoire distribuée d'une grappe de PC tout permettant d'utiliser la mémoire externe, par exemple le disque dur,si cela est nécessaire. Cette extension que nous aloons présenter est optimisée pour ses accès aux données.

Les algorithmes séquentiels de ce type et leurs structures de données ont été étudiés et développés pour plusieurs problèmes fondamentaux, tels que le problème de tri (voir [Aggarwal 1988]), des problèmes utilisant de gros graphes, les problèmes géométriques et les problèmes de modélisation et d'analyse de gros terrains. Ils se basent sur le modèle d'entrée/sortie (E/S) à deux niveaux (The two-level I/O-model, en anglais) (voir **Figure 8.1**) proposé par Aggarwal et Vitter (voir [Aggarwal 1988]). Dans ce modèle, la machine se compose d'une mémoire principale, appelée mémoire interne, de taille M et d'un disque, considéré en tant que mémoire externe, de taille supposée infinie. Un bloc de B éléments consécutifs peut être transféré entre la mémoire interne et la mémoire externe en une seule opération d'E/S. Les calculs ne peuvent avoir lieu sur des éléments dans la mémoire externe et la complexité d'un algorithme est mesuré en termes du nombre d'E/S qu'il effectue.

Dans le cas des machines parallèles, chaque machine peut posséder P processeurs (où $P \geq 1$) et D mémoires externes ($D \geq 1$) indépendantes parallèles. Un modèle d'E/S spécifique au parallélisme a été proposé : le modèle des disques parallèles (Parallel Disk Model en anglais). Dans ce cadre il existe une hiérarchie dans l'organisation de la mémoire et deux formes de modélisation ont été développées, l'une à à deux niveaux (Two level I/O an anglais, voir **Figure 8.2**) et l'autre à multiples niveaux hiérarchiques de la mémoire (Hierarchical Multilevel Memories en anglais,

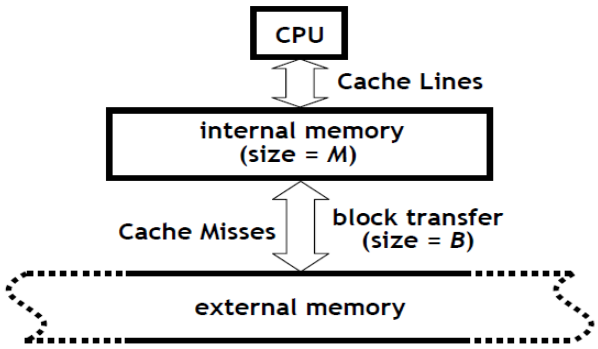
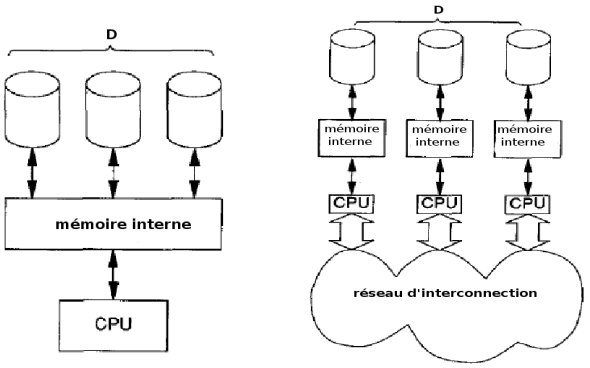


FIGURE 8.1 – Modèle de E/S en deux niveaux

voir **Figure 8.3**).



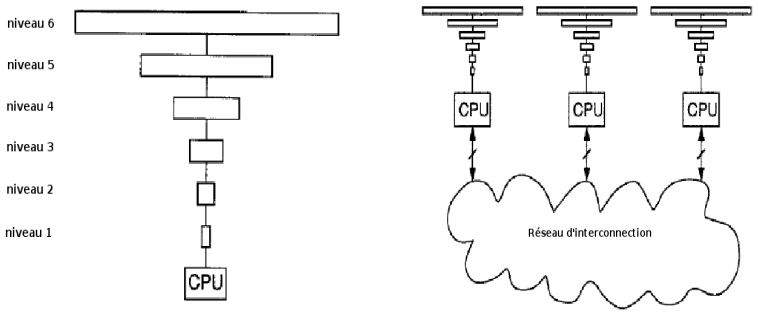
(a) un processeur (b) P processeurs

FIGURE 8.2 – Modèles de disques parallèles : deux niveaux

La performance des E/S des algorithmes et des structures de données, qui se basent sur le modèle des disques parallèles en deux niveaux, peut être exprimée par les opérations fondamentales (voir [Vitter 1992, Vitter 1999]) ci-dessous :

1. scan(N) : réalisation d'un passage sur l'ensemble des N éléments d'un fichier. La complexité de scan(N) est $O(\frac{N}{D\times B})$ opérations d'E/S.
2. sort(N) : tri de N éléments, les N éléments sont consécutivement sauvegardés dans la mémoire externe, La mémoire interne est vide au départ. L'opération sort(N) est réalisée avec la complexité $O(\frac{N}{D\times B} \times log_{\frac{M}{B}}(\frac{N}{B}))$ opérations d'E/S,
3. L'opération de recherche en ligne de N items est $O(log_{D\times B}(N))$ opérations d'E/S.

Plusieurs algorithmes séquentiels, qui manipulent la mémoire externe pour le problème de la gestion des gros graphe, ont été considérés dans des contributions (voir [Abello 1998, Agarwal 1998, Arge 1995a, Arge 1995b, Buchsbaum 2000, Chiang 1995,



(a) un processeur (b) P processeurs

FIGURE 8.3 – Modèles de disques parallèles : multiples niveaux hiérarchiques

Feuerstein 1993, Kumar 1996, Maheshwari 1999, Munagala 1999, Nodine 1993, Ullman 1990, Arge 2001b, Arge 2004, Vitter 2001, Meyer 2001b, Her 2005]). Dans les graphes directes G = (V, E), où V est un ensemble de sommets et E est un ensemble d'arêtes, les algorithmes pour le problème de parcours en largeur (Breadth First Search en anglais, ou BFS) et pour le parcours en profondeur (Depth First Search en anglais, ou DFS), utilisent $O((V + scan(E))) \times log\frac{V}{B} + sort(E)$ opérations d'E/S (voir [Buchsbaum 2000]). Le problème d'arbre couvrant de poids minimum et de composantes connexes peut être résolu dans $O(sort(E))$ opérations de E/S (voir [Chiang 1995, Abello 1998]).

Il existe aussi des implémentations de ces algorithmes (voir [Arge 2003b, Arge 2001c, Agarwal 2006b, Arge 2006]) et structures de données (voir [Arge 2002]) pour les gros volumes de données dans la bibliothèque TPIE (`http://www.cs.duke.edu/TPIE/`). Cette bibliothèque est utilisée afin d'analyser des terrains en se basant sur la technique "out-of-core" mais ils ne sont implémentés que sur un ordinateur séquentiel possédant un ou plusieurs disques durs. C'est sur cette base que les plateformes TerraFlow (`http://www.cs.duke.edu/geo*/terraflow/`) et TerraStream (`http://www.madalgo.au.dk/Trac-TerraSTREAM`), ont été conçues dans le but d'analyser les gros terrains. Ces deux plateformes sont décrites dans **Section 2.2.3**

Comme nous avons illustré dans **Partie III**, ParaFlow consiste en un pipeline de calculs parallèles dans l'analyse de la Ligne de Partage des Eaux pour de gros MNT en se basant sur la puissance de traitements de la grappe de machines (cluster en anglais). Dans un cas où nous ne pouvons pas obtenir une grande grappe de machines suffisamment grande, ou bien où nous n'avons que des machines bureautiques, nous pouvons aussi utiliser l'extension de ParaFlow, appelée ParaFlow-OOC, dans l'analyse de la LPE. Les travaux dans la suite dans cette thèse se sont ainsi portés sur le calcul parallèle des bassins versants au niveau cuvettes (voir [Hiep-Thuan 2009]) et ainsi au niveau des fleuves (voir [Hiep-Thuan 2010]) et sur le calcul des flots d'ac-

cumulations (voir [Hiep-Thuan 2011]) en utilisant la mémoire externe pour de gros volumes de données. Cette partie consiste à étendre les résultats obtenus précédemment en utilisant des techniques dites “out-of-core” lorsque le problème à traiter n’entre pas en mémoire centrale des machines de la grappe.

Chapitre 9

Détails de ParaFlow-OOC

Sommaire

9.1 Calcul parallèle des directions d'écoulement 115
9.2 Calcul parallèle des cuvettes 116
9.3 Calcul parallèle du graphe de connectivité 117
9.4 Calcul parallèle hiérarchique des bassins versants 119
9.5 Tests 119

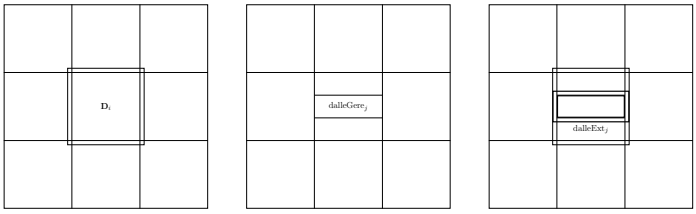
Notre méthode de parallelisation subdivise un domaine global D d'un MNT en sous-domaines D_i et chacun D_i est gérés par un processeur *Proc. i*. Les données locales sont étendues par la zone d'extension $D^+{}_i$ qui est gérés par un autre processeur. À cause de la limitation en taille du nombre de noeuds de la grappe et des mémoires principales de chacun des noeuds, ces zones de données à calculer peuvent encore être trop volumineux pour tenir en mémoire principale d'un nœud comme on l'a vu dans le chapitre 7. L'idée notre méthode méthode out-of-core est de continuer à subdiviser le sous-domaine D_i en dalles telles que chaque processeur est capable à les charger une seule à la fois en mémoire principale afin d'effectuer les calculs. Chaque dalle contient un nombre d'éléments consécutifs dans son sous domaine D_i (voir figure 9.1).

Bien évidemment, cette distribution pose des problèmes similaires à ceux que l'on rencontre lorsque l'on parallélise. Par exemple un bassin versant global peut se partager sur un certain nombre de processeurs mais aussi sur un certain nombre de dalles. Dans ce cas, il est impossible de calculer des composantes connexes réelles sans accès aux données globales (accès au disque dur et la communication entre des processeurs). Nous avons vu que nous avons résolu le problème des accès distants sur les autres processeurs en calculant seulement un arbre partiel et en remettant le problème à une phase ultérieur de communication. C'est la même démarche que nous adoptons pour le calcul out-of-core, nous calculons un arbre partiel sur les données d'une dalle puis nous résolvons séparément le problème pour les valeurs dépendantes des autres dalles.

9.1 Calcul parallèle des directions d'écoulement

Cette phase est très similaire à la version incore de Paraflow. Pendant le calcul de directions d'écoulement, lorsque la mémoire principale d'un nœud n'est pas suffisante notre méthode subdivise D_i (voir figure 9.1 en plusieurs dalles $liste_{dalles}$ de telle

manière que chaque processeur *Proc. i* peut manipuler ces données sur une dalle chargée en mémoire désignée ici par $dalleGere_j \in liste_{dalles}$ (voir figure 9.1(b). Les données sont d'abord sauvegardées sur un disque dur local. Pendant ce traitement, chaque dalle est chargée, traitée, le résultat est sauvegardé sur le disque avant de passer à la dalle suivante.



(a) Sous-domaine D_i (b) $dalleGere_j$ (c) $dalleExt_j$

FIGURE 9.1 – Découpage des domaines en dalles

Dans le cadre de ce calcul il n'y a pas de difficulté particulière, il faut juste veiller à charger chaque dalle en l'accompagnant d'une zone d'extension comme dans le cas de la version incore de notre algorithme. Chaque dalle $dalleGere_j \in liste_{dalles}$ doit aussi être étendue par une zone d'extension $dalleExt_j$ en fonction de la zone à gérer D_i et de la zone étendue $D^+{}_i$ d'un processeur *Proc. i* (voir figure 9.1(c)). Un graphe de directions d'écoulement FD_i^j est calculé pour tous les pixels $p \in dalleGere_i$ en utilisant les données de la $zonedalleGereExt_j = dalleGere_j \cup dalleExt_j$.

Pendant ce calcul chaque processeur *Proc. i* peut aussi détecter dans le sous-domaines D_i les *puits primaires* et les *puits secondaires* dont la définition est décrite dans la **Section 6.1.2.2** Ces puits primaires et secondaires sont respectivement sauvegardés dans la liste $listePuits_{primaire}$ et la liste $listePuits_{secondaire}$.

Cette phase est très simple à implémenter, aucune communication n'est nécessaire entre les processeurs. Le calcul est réalisé dalle par dalle, prises dans la liste de dalles $liste_{dalles}$. Cela correspond à une simple séquentialisation sur chaque noeuds de phases parallèles décrites dans la version incore.

9.2 Calcul parallèle des cuvettes

Le calcul dans la phase précédente nous donne un graphe de directions d'écoulements FD. Comme dans la version incore il est possible localement d'étiqueter les puits primaires et secondaires. Vient ensuite la phase de propagation des labels de chaque puits à tous les pixels de la cuvette concernée. Lorsque cette propagation ne conduit pas à explorer des pixels hors de $zonedalleGereExt_j$ il n'y a pas de problème. Le calcul est le même que pour la version incore. Si l'algorithme de compression de chemin abouti à des pixels dans une dalle qui n'est pas celle en mémoire alors le problème est géré exactement comme dans le cas où l'on arrive au bord de la zone dont chaque processeur a la responsabilité. On crée des puits secondaires internes et un nouveau graphe, $GdgFD_i$, est généré où sont notées ces

dépendances. Une fois que toutes les dalles ont été traitées localement, ce graphe dépendance est résolu en utilisant à nouveau l'algorithme de compression de chemins toujours à l'image de ce qui est fait dans la version incore pour ce qui concerne l'étiquetage global des cuvettes. En pratique nous ne créons pas de labels spécifiques en interne pour ce traitement, $GdgFD_i$ stocke des liens entre des pixels. Une structure de données Map_i<PosPuits,IdPuits> permet d'associer à chaque pixel racine son identifiant. Cette structure de données permet de connaître l'identifiant d'un pixel racine même si la dalle qui le contient n'est pas chargée en mémoire ce qui permet d'éviter de nombreux chargement/déchargements. De même les pixels qui sont aux bords de D_i,sont maintenus en mémoire dans une structure de données $Map_{B_{D_i}} =$ {p, idPuits}. Ceci évite de recharger toutes les dalles une à une pour accéder à ces pixels pour la suite du processus qui se déroule à présent exactement comme dans la version incore de ParaFlow, illustrée dans **Section 6.1.2.4**.

Finalement, l'étiquette des pixels correspondant aux puits secondaires est remplacée par celle de leur racine. Cette phase est totalement parallèle et réalisée par un parcours linéaire pour tous les pixels apres chagement séquentiel de chaque $dalleGere_j \in liste_{dalles}$.

9.3 Calcul parallèle du graphe de connectivité

L'objectif de la construction du graphe de connectivité a été présenté dans la **Section 6.1.3**. Ce graphe de connectivité GC_0 va reconstituer la connectivité entre les cuvettes pour l'ensemble du MNT. Chaque cuvette (étant un bassin versant au premier niveau) représente un sommet et est identifiée par un numéro unique.

Dans la version incore de Paraflow, les connectivités locales au processeur *Proc. i* entre des cuvettes sont simplement déduites lors d'un parcours linéaire de tous les pixels de D_i. Lorsque deux pixels adjacent n'appartiennent pas à la même cuvette, une nouvelle arête entre ces deux cuvettes est ajoutée dans GC_i^0 si elle n'était pas encore présente. Le poids de l'arête correspond à la hauteur maximale entre ces deux pixels. Si elle était déjà présente, l'arête est mise à jour si le nouveau poids est inférieur à celui déjà trouvé.

Comme dans la phase précédente, nous proposons de charger en mémoire dalle par dalle les données de D_i. Nous ne manipulons à un instant donné que les données de $dalleGere_j$. Mais cela se révèle insuffisant si GC_i^0 est très important comme cela est souvent le cas pour les très gros MNT au début du calcul. Nous complétons donc notre méthode en sauvegardant des portions de GC_i^0 au cours du calcul. La difficulté réside dans le fait qu'il n'est pas possible de savoir si une donnée de GC_i^0 ne risque pas d'être mise à jour alors qu'elle est déchargée sur le disque. Dans ce cas nous choisissons de stocker cette nouvelle information et de résoudre le conflit ultérieurement dans une seconde phase de traitement.

Voici la démarche plus précisément. L'idée consiste à localement détecter, pour chaque $dalleGere_j$, toutes les arêtes de poids minimal entre les cuvettes. Nous choi-

sissons de distinguer trois types d'arêtes :

1. Le premier type contient toutes les arêtes de poids minimal qui aboutissent aux sommets s tels que le puits s corresponde à un pixel appartenant à $dalleGere_j$. Ces arêtes sont insérées dans un ensemble noté $gcGere_i^0[j]$.
2. Le deuxième est l'ensemble de toutes les arêtes de poids minimal qui aboutissent aux sommets s tels que le puits s corresponde à un pixel n'appartenant pas à $dalleGere_j$. Si ce pixel appartient à $dalleGere_k$ ($k \neq j$) alors cet arête est insérée dans $gcGere_i^0[k]$.
3. Le dernier est l'ensemble de toutes les arêtes de poids minimal qui aboutissent aux sommets s tels que le puits s corresponde à un pixel n'appartenant pas à D_i. Une telle arête est insérée dans $gcExt_i^0[r]$.

L'ensemble $gcGere_i^0[j]$ qui correspond au premier type d'arêtes est généralement très majoritaire. En effet les sous domaines des dalles contiennent une multitudes de cuvettes souvent entièrement contenues dans une seule dalle. Cela signifie que $gcGere_i^0[j]$ contient majoritairement des arrêtes dont il ne sera pas nécessaire de modifier la valeur de hauteur à l'examen des valeurs du reste du domaine D_i. Il est donc judicieux de privilégier cet ensemble pour un stockage sur disque et ainsi libérer de la mémoire. L'ensemble $gcGere_i^0[j]$ est sauvegardé sur disque à chaque changement de dalle. Les deux autres ensembles, bien moins volumineux sont conservés en mémoire. Ils sont mis à jours dès que de nouvelles données apparaissent lors du chargement des $dalleGere_{j'}$. Notons que certaines de ces données sont mises sur disque lors du traitment des dalles $dalleGere_{j'}$ qui stockent sur disque les $gcGere_i^0[j']$. Notons aussi que lors du traitement de $dalleGere_{j'}$, nous gardons en mémoire dans $gcGere_i^0[j]$ d'éventuelles arêtes de type 2 qui aboutissent à des puits qui correspondent à des pixels appartenant à $dalleGere_j$.

Ensuite exactement comme dans la version incore, nous pratiquons une phase d'échange des arrêtes de $gcExt_i^0$ entre tous les processeurs. Chaque processeur va donc obtenir un ensemble de nouvelles arêtes, noté $gcRecv_i$, calculées par les autres processeurs.

Après avoir chargé toutes les dalles une première fois, et avoir réalisé la phase d'échanges, le traitement consiste à recharger une par une toutes les sauvegardes sur disque de $gcGere_i^0[j]$ et à les mettre à jour avec les données contenues dans $gcGere_i^0[j]$ et dans $gcRecv_i$. Une seule version des arrêtes est conservée avec une hauteur minimale. Nous obtenons le graphe de connectivité local sur les disques de la machine parallèle.

Notons que la technique "out-of-core" prend aussi en compte la règle Ω (voir **Section 5.2.4**).

9.4 Calcul parallèle hiérarchique des bassins versants

Dans **Section 6.2**, nous avons présenté une implémentation parallèle de l'algorithme Borůvka afin de calculer un arbre couvrant de poids minimum et nous avons aussi montré comment mettre en œuvre cette méthode dans le calcul de la hiérarchie des bassins versants. Ce calcul commence à partir d'un graphe de connectivité GC^0 au premier niveau. L'idée consiste à progressivement fusionner les sommets niveau par niveau.

La parallèlisation de notre méthode, est réalisée en subdivisant en blocs de sommets le graphe GC^0. Chaque bloc GC_i^0 est distribué à un processeur *Proc. i*. Cette distribution vise à minimiser le nombre d'arêtes externes reliant deux sommets appartenant à deux domaines différents. Dans le cas de données massives, le graphe GC_i^0, peut ne pas tenir en mémoire, et donc on peut le redécouper en sous-graphe $GC_i^0[j]$ de telle sorte qu'un processeur *Proc. i* puisse charger chacun des sous-graphes $GC_i^0[j]$ en mémoire centrale. Chacun des sous-graphes est chargé et, progressivement , on construit une nouvelle structure de donnée, notée $GCWithMC_i$, qui est bien plus légère que GC_i^0 et qui est un graphe de connectivité dans lequel on ne conserve pour chaque sommet que les arrêtes dont le poids est minimal. Ce graphe $GCWithMC_i$ doit pouvoir tenir dans la mémoire d'un processeur. Le reste du processus est réalisé sur $GCWithMC_i$ il est identique à la version incore. Notons toutefois que le calcul du graphe de connectivité au rang supérieur nécessite d'utiliser les sous graphes $GC_i^0[j]$ pour déterminer les nouvelles arêtes entre les nouveaux sommets. Ceci peut être réalisé en chargeant séparément dans le temps les différents $GC_i^0[j]$.

9.5 Tests

Les premiers tests de Paraflow-OOC ont été réalisés sur la machine Dell (2Go de RAM, et processeur bi-core 2,2GHz, Ubuntu 10.0.4, OpenMPI version 1.4.1). Le temps de calcul de la partie calcul des cuvettes pour un seul processus sur le MNT de nigeretendu (30.001×42.001) est environ de 311 minutes. Ce temps inclu le chargement et la sauvegarde des données. ParaFlow-OOC donne strictement les mêmes résultats pixel par pixel, pour le calcul des cuvettes, que ParaFlow.

La version ParaFlow-OOC actuelle est encore un prototype qui demande à être amélioré notamment sur les premières étape de l'algorithme de Borůvka. Cependant les premiers résultats sont concluants et valident la méthodologie employée. Cela nous permet d'envisager une version performante de ParaFlow-OOC qui aura des performances comparable à celles de TerraFlow sur une machine de bureau et qui aura en plus la capacité d'utiliser les ressources d'une machine parallèle lorsqu'elle est disponible.

Cinquième partie

CONCLUSION ET PERSPECTIVES

Chapitre 10

Conclusion

Dans le cadre du projet eXtenGIS (https://traclifo.univ-orleans.fr/extengis/), l'objectif principal de la thèse porte sur la parallélisation de traitements sur des gros volumes de données géo-référencées notamment dans le cadre de la délimitation des bassins versants de grands fleuves et le calcul des flots d'accumulation. Elle est basée sur l'analyse des données locales afin d'éviter des communications ou synchronisations inutiles. Tous les calculs sont totalement parallèles et aucune structure de données complexes n'est construite. Toutes les structures de données sont réparties sur les nœuds de la grappe d'ordinateurs.

Après une étude bibliographique approfondie sur le problème de la délimitation des bassins versants et ainsi que sur l'extraction automatique des réseaux hydrographiques dans les modèles numériques de terrain (ou MNT en abrégé), les travaux de thèses ont été consacrés au développement d'une technique de calcul parallèle d'une forêt d'arbres couvrants de poids minimum représentant le parcours de l'eau de chaque point du MNT vers la mer. Cette technique débute par une délimitation des cuvettes (ensemble de points allant vers le même minimum local) contenues dans le MNT. Ensuite une hiérarchie de déversement des cuvettes les unes dans les autres est construite jusqu'à obtenir les bassins versants des fleuves. Cette construction est dirigée pour que l'eau prenne le chemin le moins couteux pour aller vers la mer et pour que les bassins versants de deux fleuves différents ne soient pas fusionnés. Par ailleurs la technique utilisée permet aussi de tenir compte de l'absence d'information au bord du MNT en se basant sur la règle Ω (voir **Section 5.2.4**). Notre méthode combine les techniques utilisées dans de différents domaines de recherches telles que l'hydro-géologie, le traitement d'images et la théorie des graphes afin d'obtenir un algorithme qui donne à la fois des résultats plus précis en terme de géo-morphologie et est efficace et scalable. De plus, cette méthode n'a pas besoin non plus de traitements particuliers, tels que *Stream Burning*, lorsque le MNT contient les bruits naturels, tels que les zones d'absence d'information ou les dépressions. Les traitements doivent modifier le MNT afin d'éliminer ces bruits. Ils sont très coûteux pour de gros MNT, en plus, le processus du traitement doit être itératif.

Cette méthode a été implémentée en C++ avec la librairie de communication MPI et des tests ont été effectués sur des MNT allant jusqu'à 10 milliards de points. Les résultats obtenus en terme performance montrent l'extensibilité de la méthode (c'est-à-dire que plus on a d'ordinateurs à disposition plus le calcul est rapide). De plus sur le plan géologique, les résultats sont tout à fait cohérents par rapport aux délimitations des bassins versants (manuelles) qui ont été effectuées notamment en

Europe.

L'idée de notre méthode, dans la modélisation et l'implémentation de ce type de calcul parallèle des cuvettes pour de grands MNT, qui soit efficace et qui passe à l'échelle. Les calculs d'étiquetage, pour tous les pixels dans un MNT, sont optimisés en utilisant la technique *compression de chemin*. Les expérimentations montrent que notre technique a une accélération quasi-linéaire en fonction du nombre de processeurs. Ce travail (voir [Hiep-Thuan 2009]) est publié dans les actes de la conférence internationale, "*The 2nd High Performance Computing and Applications 2009*", à Shanghai en Chine.

La suite du travail a consisté à étudier et à implémenter une hiérarchie de bassins versants permettant la fusion progressive des cuvettes afin d'aboutir à la délimitation automatique des bassins versants réels des cours d'eau. Cette méthode a été implémentée et testée sur de grands jeux de données avec des résultats très convaincants, tant par la qualité de la délimitation des bassins que par le passage à l'échelle du calcul. Ce travail (voir [Hiep-Thuan 2010]) a été publié à la conférence internationale "*The International Conference on High Computing & Simulation*" HPCS 2010 à Caen en France et a été sélection parmi les cinq meilleurs articles de la conférence. Une synthèse des ces travaux a été aussi soumise à un journal international, "*Concurrency and Computation : Practice and Experience*", CCPE.

Concernant la partie flots d'accumulation, c'est-à-dire déterminer combien de points du MNT se déversent en chaque point, Encore une fois ce calcul devient très coûteux sur des MNT de très grande taille et demande donc une implémentation parallèle efficace qui puisse passer à l'échelle. La méthode sur laquelle nous travaillons actuellement se base sur la hiérarchie de bassins présentée ci-dessus. La technique développée et implémentée dans la thèse se montre aussi extensible et cohérente vis-à-vis des résultats obtenus par des logiciels de traitement de MNT reconnus (ArcGIS ou GRASS). Ce travail (voir [Hiep-Thuan 2011]) a été publié à la conférence internationale, "*The International Conference on Computational Science ICCS-2011 à Singapour*".

Ses développements seront bientôt disponibles et distribués sous forme de modules GRASS. L'ensemble de mes travaux ont été implémentés dans un logiciel appelé ParaFlow, qui sera prochainement disponible sur le site du projet eXtenGIS (`https://traclifo.univ-orleans.fr/extengis/`).

Enfin la dernière partie du travail de cette thèse consiste à étendre les résultats obtenus précédent en utilisant des techniques dites "out-of-core " lorsque le problème à traiter n'entre pas en mémoire centrale des ordinateurs de la grappe de PC. Cette extension de ParaFlow, appelée ParaFlow-OOC, a été aussi développée d'une manière de calcul parallèle en combinant avec la technique "out-of-core". ParraFlow-OOC n'a pas seulement la puissance de calcul parallèle de la grappes des machines, mais aussi elle vise à la fois aux cas où nous n'avons pas de conditions afin

d'obtenir une assez grande grappe de machines, ou bien, où nous n'avons que les machines bureautiques, nous pouvons aussi utiliser ParaFlow-OOC dans l'analyse de la LPE pour de gros MNT. Les premiers résultats obtenus en utilisant ParaFlow-OOC sont tout à fait encourageants dans le calcul des cuvettes pour de gros MNT. ParaFlow-OOC a aussi donné les mêmes résultats de pixel par pixel, pour le calcul des cuvettes, que ParaFlow. Ce travail est en cours de finalisation donne lieu à une soumission dans une conférence internationale. Pour le calcul de la hiérarchie avec la technique "out-of-core", ce travail est en cours de finalisation de l'implémentation, aussi bien sûr l'expérimentation.

Chapitre 11

Perspectives

Dans le cadre du projet eXtenGIS, nous avons implémenté une plateforme de calcul répartie pendant l'analyse de la LPE pour de gros MNT. Cette plateforme consiste en un pipeline de calculs parallèles (calcul des cuvettes, calcul de la hiérarchie des bassins versants et calcul des flots d'accumulations). La plateforme a été développée non seulement par la puissance de calcul parallèle sur la grappe des machines (ParaFlow), mais aussi par l'efficacité dans l'analyse de gros MNT en optimisant entre le coût des matériels et le temps d'analyse (ParaFlow-OOC).

Pendant les années de la thèse, nous avons entièrement implémenté et expérimentaté les trois phases d'analyse de la LPE en manipulant des données sur la mémoire principale. Le calcul parallèle des cuvettes et de la hiérarchie des bassins versants avec la technique "out-of-core" sont en cours d'implémentation. Les expérimentations sont en cours de la réalisation. ParaFlow-OOC manque encore une fonctionnalité de calcul parallèle des flots d'accumulations avec la technique "out-of-core". Dans l'avenir, nous continuons ainsi à concevoir et à implémenter ParaFlow-OOC, l'extension de ParaFlow, pour ce type de calcul avec la technique "out-of-core".

La suite de travail sera aussi de proposer une autre solution, dans l'analyse de la LPE, en combinant les deux modèles de programmation parallèle, la mémoire partagée et la mémoire distribuée, afin de non seulement accélérer la puissance de calcul, mais aussi de réduire l'échange des données entre les processeurs dans les mêmes machines.

Afin de mieux comprendre les cours d'eau, au niveau de chaque petite cuvette, nous proposerons aussi une autre partie dans ParaFlow afin de faire tous les calculs précédents en se basant sur un modèle de multiples directions d'écoulements (en abrégé MFD) (voir **Section 2.2.2**).

Enfin, les différents travaux effectués pendant les années de la thèse ont permis de mettre au jour un certain nombre de problématiques liées à l'utilisation MNT volumineux sur une grappe d'ordinateurs. Ces problématiques concernent la répartition et l'accès rapide aux données du MNT. La résolution de ce problème va passer par l'utilisation de systèmes fichiers parallèles, comme PVFS "*Parallel Virtual File System*" et le développement d'algorithmes spécifiques permettant d'optimiser leur utilisation.

Bibliographie

[Abello 1998] James M. Abello, Adam L. Buchsbaum et Jeffery R. Westbrook. *A Functional Approach to External Graph Algorithms.* In Proceedings of the 6th Annual European Symposium on Algorithms, ESA '98, pages 332–343, London, UK, 1998. Springer-Verlag.

[Agarwal 1998] Pankaj K. Agarwal, Lars Arge, T. M. Murali, Kasturi R. Varadarajan et Jeffrey Scott Vitter. *I/O-efficient algorithms for contour-line extraction and planar graph blocking.* In Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms, SODA '98, pages 117–126, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[Agarwal 2005] Pankaj K. Agarwal, Lars Arge et Ke Yi. *I/O-Efficient Construction of Constrained Delaunay Triangulations.* In ESA, pages 355–366, 2005.

[Agarwal 2006a] Pankaj K. Agarwal, Lars Arge et Andrew Danner. *From point cloud to grid DEM : A scalable approach.* In In Proc. 12th International Symposium on Spatial Data Handling, pages 771–788. Springer-Verlag, 2006.

[Agarwal 2006b] Pankaj K. Agarwal, Lars Arge et Ke Yi. *I/O-efficient batched union-find and its applications to terrain analysis.* In Proceedings of the twenty-second annual symposium on Computational geometry, SCG '06, pages 167–176, New York, NY, USA, 2006. ACM.

[Aggarwal 1988] Alok Aggarwal et Jeffrey Scott Vitter. *The input/output complexity of sorting and related problems.* Commun. ACM, vol. 31, pages 1116–1127, September 1988.

[Alnuweiri 1992] H.M. Alnuweiri et V.K Prasanna. *Parallel architectures and algorithms for images component labeling.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 10, pages 1014–1034, October 1992.

[Arge 1995a] Lars Arge. *The Buffer Tree : A New Technique for Optimal I/O-Algorithms (Extended Abstract).* In Proceedings of the 4th International Workshop on Algorithms and Data Structures, WADS '95, pages 334–345, London, UK, 1995. Springer-Verlag.

[Arge 1995b] Lars Arge. *The I/O - Complexity of Ordered Binary - Decision Diagram Manipulation.* In Proceedings of the 6th International Symposium on Algorithms and Computation, ISAAC '95, pages 82–91, London, UK, 1995. Springer-Verlag.

[Arge 2001a] Lars Arge, J. S. Chase, P. N. Halpin, L. Toma, Jeffrey Scott Vitter, D. Urban et R. Wickremesinghe. *Flow computation on massive grid terrains.* GeoInformatica, vol. 7, page 2003, 2001.

[Arge 2001b] Lars Arge, Ulrich Meyer, L. Toma et Norbert Zeh. *On External-Memory Planar Depth First Search.* pages 471–482, 2001.

[Arge 2001c] Lars Arge, L. Toma et Jeffrey Scott Vitter. *I/O-Efficient Algorithms for Problems on Grid-Based Terrains.* J. Exp. Algorithmics, vol. 6, page 1, 2001.

[Arge 2002] Lars Arge, Octavian Procopiuc et Jeffrey Scott Vitter. *Implementing I/O-efficient Data Structures Using TPIE.* In Proceedings of the 10th Annual European Symposium on Algorithms, ESA '02, pages 88–100, London, UK, UK, 2002. Springer-Verlag.

[Arge 2003a] Lars Arge, J. S. Chase, P. N. Halpin, L. Toma, Jeffrey Scott VitterD. Urban et R. Wickremesinghe. *Efficient flow computation on massive grid terrain datasets.* Geoinformatica, vol. 7, no. 4, pages 283–313, 2003.

[Arge 2003b] Lars Arge, Jeffrey S. Chase, Patrick Halpin, Laura Toma, Jeffrey S. Vitter, Dean Urban et Rajiv Wickremesinghe. *Efficient Flow Computation on Massive Grid Terrain Datasets.* Geoinformatica, vol. 7, no. 4, pages 283–313, 2003.

[Arge 2004] Lars Arge, Gerth Stølting Brodal et L. Toma. *On external-memory MST, SSSP and multi-way planar graph separation.* J. Algorithms, vol. 53, pages 186–206, November 2004.

[Arge 2006] Lars Arge, Andrew Danner, Herman Haverkort et Norbert Zeh. *I/O-Efficient Hierarchical Watershed Decomposition of Grid Terrain Models.* In In Proc. 12th International Symposium on Spatial Data Handling, pages 825–844. Springer-Verlag, 2006.

[Band 1986] L. E. Band. *Topographic partition of watersheds with digital elevation models.* Water Resources Research, vol. 22, no. 1, pages 15–24, 1986.

[Betser 2005] J. Betser, S. Delest et R. Boné. *Unbiased Watershed Hierarchical 3D Segmentation.* In J.J. Villanueva, editeur, Visualization, Imaging, and Image Processing (VIIP 2005), Benidorm, Spain, 2005.

[Beucher 1979] Serge Beucher et C. Lantuéjoul. *Use of watersheds in contour detection.* In International Workshop on Image Processing : Real-time Edge and Motion Detection/Estimation, Rennes, France., September 1979.

[Beucher 1990] Serge Beucher. *Segmentation d'images et morphologie mathématique.* PhD thesis, Ecole de Mines, Paris, June 1990.

[Beucher 1993] Serge Beucher et F. Meyer. *The Morphological Approach to Segmentation : The Watershed Transformation.* In E. R. Dougherty, editeur, Mathematical Morphology in Image Processing, volume 34 of *Optical Engineering*, chapitre 12, pages 433–481. Marcel Dekker, New York, 1993.

[Beucher 1994] Serge Beucher. *Watershed, hierarchical segmentation and waterfall algorithm.* In J. Serra et P. Soille, editeurs, Mathematical Morphology And Its Applications To Image Processing, volume 2, 1994.

[Beucher 2004] Serge Beucher. *Algorithmes sans biais de ligne de partage des eaux.* Rapport technique, Centre de Morphologie Mathématique de l'Ecole des Mines de Paris, 2004.

[Beucher 2005] Serge Beucher et Beatriz Marcotegui. *Fast Implementation of Waterfall Based on Graphs.* In C. Ronse, L. Najman et E. Decencière, editeurs, Mathematical Morphology : 40 Years On, volume 30 of *Computational Imaging and Vision*, pages 177–186. Springer-Verlag, Dordrecht, 2005.

[Bieniek 1996] A. Bieniek, M. Nöelle, G. Schreiber, H. Burkhardt et H. Marschner. *A parallel watershed algorithm.* Rapport technique TR-402-96-006, 1996.

[Bieniek 1997] A. Bieniek, H. Burkhardt, H. Marschner et M. Nöelle. *A parallel watershed algorithm.* In Proceedings of 10th Scandinavian Conference on Image Analysis, pages 237–244, Lappeenranta, Finland, June 1997.

[Bieniek 1998] A. Bieniek et A. N. Moga. *A Connected Component Approach to The Watershed Segmentation.* In ISMM '98 : Proceedings of the fourth international symposium on Mathematical morphology and its applications to image and signal processing, pages 215–222, Norwell, MA, USA, 1998. Kluwer Academic Publishers.

[Bieniek 2000] A. Bieniek et A. N. Moga. *An efficient watershed algorithm based on connected components.* Pattern Recognition, vol. 33, no. 6, pages 907–916, 2000.

[Bock 2005] J. D Bock, Patrick De Smet et W. Philips. *A Fast Sequential Rainfalling Watershed Segmentation Algorithm.* In Advanced Concepts for Intelligent Vision Systems, volume 3708 of *LNCS - Lecture Notes in Computer Science*, pages 476–482. Springer Berlin / Heidelberg, Germany, 2005.

[Buchsbaum 2000] Adam L. Buchsbaum, Michael Goldwasser, Suresh Venkatasubramanian et Jeffery R. Westbrook. *On external memory graph traversal.* In Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms, SODA '00, pages 859–860, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[Chew 1989] L. Paul Chew. *Constrained Delaunay triangulations.* Algorithmica, vol. 4, pages 97–108, 1989.

[Chiang 1995] Yi-Jen Chiang, Michael T. Goodrich, Edward F. Grove, Roberto Tamassia, Darren Erik Vengroff et Jeffrey Scott Vitter. *External-memory graph algorithms.* In Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms, SODA '95, pages 139–149, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[Chorowicz 1992] Jean Chorowicz, Charles Ichoku, Serges Riazanoff, Youn Jong Kim et Bernard Cervelle. *A Combined Algorithm for Automated Drainage Network Extraction.* Water Resources Research, vol. 28, pages 1293–1302, May 1992.

[Chung 1996] S. Chung et A. Condon. *Parallel implementation of Boruvka's minimum spanning tree algorithm.* In IPPS '96 : Proceedings of the 10th International Parallel Processing Symposium, pages 302–308, Washington, DC, USA, 1996. IEEE Computer Society.

[Cormen 1990] T. H. Cormen, C. E. Leiserson et R. L. Rivest. Introduction to algorithms. MIT Press, Cambridge, MA, 1990.

[Cousty 2008] Jean Cousty, Michel Couprie, Laurent Najman et Gilles Bertrand. *Weighted fusion graphs : Merging properties and watersheds.* Discrete Appl. Math., vol. 156, pages 3011–3027, August 2008.

[Danner 2007] Andrew Danner, Thomas Mølhave, Ke Yi, Pankaj K. Agarwal, Lars Arge et Helena Mitasova. *TerraStream : from elevation data to watershed hierarchies.* In Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems, GIS '07, pages 28 :1–28 :8, New York, NY, USA, 2007. ACM.

[Digabel 1978] H. Digabel et C. Lantuéjoul. *Itérative algorithms.* In J.-L. Chermant, editeur, Actes du Second Symposium Européens d'Analyse Quantitative des Microstructures en Sciences des Matériaux, Biologie et Médecine, volume 2, pages 85–99, Stuttgart, 4-7 Octobre 1978. Rierder Verlag.

[Edelsbrunner 2000] H Edelsbrunner, D. Letscher et A. Zomorodian. *Topological persistence and simplification.* In Proceedings of the 41st Annual Symposium on Foundations of Computer Science, pages 454–463, Washington, DC, USA, 2000. IEEE Computer Society.

[Edelsbrunner 2001] Herbert Edelsbrunner, John Harer et Afra Zomorodian. *Hierarchical morse complexes for piecewise linear 2-manifolds.* In Proceedings of the seventeenth annual symposium on Computational geometry, SCG '01, pages 70–79, New York, NY, USA, 2001. ACM.

[Embrechts 1993] H. Embrechts, D. Roose et P. Wambacq. *Component labelling on a MIMD multiprocessor.* CVGIP : Image Understanding, vol. 57, no. 2, pages 155–165, March 1993.

[Fairfield 1991] J. Fairfield et P. Leymarie. *Drainage networks from grid digital elevation models.* Water Resources Research, vol. 27, no. 5, pages 709–717, 1991.

[Feuerstein 1993] Esteban Feuerstein et Alberto Marchetti-Spaccamela. *Memory Paging for Connectivity and Path Problems in Graphs.* In Proceedings of the 4th International Symposium on Algorithms and Computation, ISAAC '93, pages 416–425, London, UK, 1993. Springer-Verlag.

[Fisher 1996] P. F. Fisher. *EXTENDING THE APPLICABILITY OF VIEWSHEDS IN LANDSCAPE PLANNING.* Photogrammetric Engineering and Remote Sensing (PE&RS), vol. 62, no. 11, pages 1297–1302, 1996.

[Floriani 1994] Leila De Floriani, Paola Marzano et Enrico Puppo. *Line-of-sight communication on terrain models.* International Journal of Geographical Information Systems, vol. 8, no. 4, pages 329–342, 1994.

[Flynn 1972] Michael J. Flynn. *Some computer organizations and their effectiveness.* IEEE Trans. Comput., vol. 21, pages 948–960, September 1972.

[Freeman 1991] T.G. Freeman. *Calculating catchment area with divergent flow based on a regular grid.* Comput. Geosci., vol. 17, no. 3, pages 413–422, 1991.

[Garbrecht 1997] J. Garbrecht et L. W. Martz. *The assignment of drainage direction over flat surfaces in raster digital elevation models.* Journal of Hydrology, vol. 193, pages 204–213, 1997.

[Grama 2003] Ananth Grama, Anshul Gupta, George Karypis et Vipin Kumar. Introduction to parallel computing (second edition). Addison Wesley, 2003.

[Grimaldi 2007] Salvatore Grimaldi, Fernando Nardi, Francesco Di Benedetto, Erkan Istanbulluoglu et Rafael L. Bras. *A physically-based method for removing pits in digital elevation models.* Advances in Water Resources, vol. 30, no. 10, pages 2151 – 2158, 2007. Recent Developments in Hydrologic Analysis.

[Grimaud 1992] Michel Grimaud. *New measure of contrast : the dynamics.* volume 1769, pages 292–305. SPIE, 1992.

[Hahn 2003] H. K. Hahn et H.-O. Peitgen. *IWT-Interactive watershed transform : A hierarchical method for efficient interactive and automated Segmentation of multidimensional gray-scale images.* In SPIE–The International Society for Optical Engineering., volume 5032, pages 643–653, 2003.

[Haralick 1985] Robert M. Haralick et Linda G. Shapiro. *Image segmentation techniques.* Computer Vision, Graphics, and Image Processing, vol. 29, no. 1, pages 100 – 132, 1985.

[Her 2005] Jun-Ho Her et R. S. Ramakrishna. *External-memory depth-first search algorithm for solid grid graphs.* Inf. Process. Lett., vol. 93, pages 177–183, February 2005.

[Hiep-Thuan 2009] Do Hiep-Thuan, Sébastien Limet et Emmanuel Melin. *Parallel computing of catchment basins in large digital elevation model.* In The second International Conference on High Performance Computing and Applications (HPCA), volume 5938 of *LNCS - Lecture Notes in Computer Science*, pages 133–138, Berlin Heidelberg, August 2009. Springer-Verlag.

[Hiep-Thuan 2010] Do Hiep-Thuan, Sébastien Limet et Emmanuel Melin. *Parallel computing of cachment basins of rivers in large digital elevation models (Selected as one of 5 best papers).* The International Conference on High Performance Computing & Simulation (HPCS), June–July 2010.

[Hiep-Thuan 2011] Do Hiep-Thuan, Sébastien Limet et Emmanuel Melin. *Parallel Computing Flow Accumulation in Large Digital Elevation Models.* Procedia Computer Science, vol. 4, pages 2277–2286, 2011.

[Holmgren 1994] P. Holmgren. *Multiple flow direction algorithms for runoff modelling in grid based elevation models : An empirical evaluation.* Hydrological Processes, vol. 8, pages 327–334, 1994.

[Hutchinson 1989] M. F. Hutchinson. *A new procedure for gridding elevation and stream line data with automatic removal of spurious pits.* Journal of Hydrology, vol. 106 (3-4), pages 211–232, 1989.

[Isaac 1956] E. J. Isaac et R. C. Singleton. *Sorting by address calculation.* J. ACM, vol. 3, no. 3, pages 169–174, 1956.

[Jenson 1988] S. Jenson et J. Domingue. *Extracting topographic structure from digital elevation data for geographic information system analysis.* Photogrammetric Engineering and Remote Sensing, vol. 54(11), pages 1593–1600, 1988.

[Johansson 1994] T. Johansson. *Image analysis algorithms on general purpose parallel architectures.* PhD thesis, Centre for Image Analysis, University of Uppsala, Uppsala, 1994.

[Jones 2002] R. Jones. *Algorithms for using a DEM for mapping catchment areas of stream sediment samples.* Comput. Geosci., vol. 28, no. 9, pages 1051–1060, 2002.

[Kumar 1996] Vijay Kumar et Eric J. Schwabe. *Improved Algorithms and Data Structures for Solving Graph Problems in External Memory.* In Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing (SPDP '96), SPDP '96, pages 169–, Washington, DC, USA, 1996. IEEE Computer Society.

[Lantuéjoul 1978] C. Lantuéjoul. *La squelettisation et son application aux mesures topologiques des mosaiques polycristallines.* PhD thesis, Ecole de Mines, Paris, 1978.

[Lea 1992] N. L. Lea. An aspect driven kinematic routing algorithm, pages 147–175. Chapman & Hall, New York, 1992.

[Lee 1991] J. Lee. *Analyses of visibility sites on topographic surfaces.* Analyses of visibility sites on topographic surfaces, vol. 5, pages 413–429, 1991.

[Leighton 1992] F. T. Leighton. Introduction to parallel algorithms and architectures. Morgan Kaufman, San Mateo, California, 1992.

[Lindsay 2005] J. B. Lindsay. *The Terrain Analysis System : A tool for hydro-geomorphic applications.* Hydrological Processes, vol. 19, no. 5, pages 1123–1130, 2005.

[Maheshwari 1999] Anil Maheshwari et Norbert Zeh. *External Memory Algorithms for Outerplanar Graphs.* In Proceedings of the 10th International Symposium on Algorithms and Computation, ISAAC '99, pages 307–316, London, UK, 1999. Springer-Verlag.

[Maidment 2002] D. R. and Maidment. Arc hydro : Gis for water resources. Environmental Systems Research Inst., 2002.

[Mark 1984] D. M. Mark. *Automated detection of drainage networks from digital elevation models.* Cartographica, vol. 21, no. 2–3, pages 168–178, 1984.

[Mark 1988] D. M. Mark. *Network modes in geomorphology.* In anderson, editeur, Modelling in Geomorphological Systems, chapitre 4, pages 73–97. John Wiley, New York, 1988.

[Marks 1984] D. Marks, J. Dozier et J. Frew. *Automated basin delineation from digital elevation data.* Geo. Processing, vol. 2, pages 299–311, 1984.

[Martz 1988] L. W. Martz et E. De Jong. *CATCH : a FORTRAN program for measuring catchment area from digital elevation models.* Comput. Geosci., vol. 14, no. 5, pages 627–640, 1988.

[Martz 1992] L. W. Martz et J. Garbrecht. *Numerical definition of drainage network and subcatchment areas from digital elevation models.* Comput. Geosci., vol. 18, no. 6, pages 747–761, 1992.

[Martz 1998] Lawrence W. Martz et Jurgen Garbrecht. *The treatment of flat areas and depressions in automated drainage analysis of raster digital elevation models.* Hydrological Processes, vol. 12, no. 6, pages 843–855, 1998.

[Meijster 1995] A. Meijster et J. B. T. M. Roerdink. *A proposal for the implementation of a parallel watershed algorithm.* In CAIP '95 : Proceedings of the 6th International Conference on Computer Analysis of Images and Patterns, pages 790–795, London, UK, 1995. Springer-Verlag.

[Meijster 1996] A. Meijster et J. B. T. M. Roerdink. *Computation of Watersheds Based on Parallel Graph Algorithms.* In P. Maragos, R. W. Shafer et M. A. Butt, editeurs, Mathematical Morphology and its Applications to Image and Signal Processing, pages 305–312. Kluwer Acad, Dordrecht, 1996.

[Meijster 1998] A. Meijster et J. B. T. M. Roerdink. *A Disjoint Set Algorithm for The Watershed Transform.* In S. Theodoridis, I. Pitas A. Stouraitis et N. Kalouptsidis, editeurs, Proceedings IX European Signal Processing Conference (EUSIPCO'98), pages 1665–1668, Rhodes, Greece, 08- 11 September 1998.

[Meyer 1993] F. Meyer. *Integrals, Gradients and Watershed Lines.* In J. Serra et P. Salembier, editeurs, Proc. Workshop on Mathematical Morphology and its Applications to Signal Processing, pages 70–75. Barcelona, Spain, 1993.

[Meyer 1994] F. Meyer. *Topographic distance and watershed lines.* Signal Process, vol. 38, no. 1, pages 113–125, 1994.

[Meyer 1999] F. Meyer. *Graph Based Morphological Segmentation.* In IAPR-TC-15 Workshop on Graph Based Representation, pages 51–61, Vienna, Austria, may 1999.

[Meyer 2001a] F. Meyer. *An Overview of Morphological Segmentation.* IJPRAI, vol. 15, no. 7, pages 1089–1118, 2001.

[Meyer 2001b] Ulrich Meyer. *External memory BFS on undirected graphs with bounded degree.* In Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms, SODA '01, pages 87–88, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[Mitasova 2005] H. Mitasova, L. Mitas et R. Harmon. *Simultaneous Spline Approximation and Topographic Analysis for Lidar Elevation Data in Open-Source GIS.* IEEE Geoscience and Remote Sensing Letters, vol. 2, pages 375–379, 2005.

[Moga 1994] A. N. Moga, B. P. Dobrin, M. Gabbouj et T. Viero. *Implementation of a Distributed Watershed Algorithm.* In J. Serra et P. Soille, editeurs, ISMM'94 Mathematical Morphology and It's Applications to Image Processing, chapitre 12, pages 281–288. Kluwer Academic, 1994.

[Moga 1995a] A. N. Moga, B. Cramariuc et M. Gabbouj. *An efficient watershed segmentation algorithm suitable for parallel implementation.* In Proceedings IEEE International Conference in Image Processing, volume 2, pages 101–104, Washington D.C, October 1995. IEEE Computer Society Press.

[Moga 1995b] A. N. Moga, B. Cramariuc et M. Gabbouj. *A parallel watershed algorithm based on rainfalling simulation.* In Proceedings 12th European Conference on Circuit Theory and Design, volume 1, pages 339–342, Istanbul, Turkey, August 1995.

[Moga 1995c] A. N. Moga, B. Cramariuc et M. Gabbouj. *A parallel watershed algorithm based on the shortest paths computation.* In P. Fritzson et L. Finmo, editeurs, Parallel Programming and Applications, pages 137–140, Amsterdam, The Netherlands, May 1995. IOS Press Ohmsha.

[Moga 1995d] A. N. Moga, T. Viero, B. P. Dobrin, M. Gabbouj, M. Nöelle, G. Schreiber et H. Burkhardt. *Parallel watershed algorithm based on sequential scanning.* In Proceedings IEEE Workshop on Nonlinear Signal and Image Processing, volume 2, pages 991–994, Halkidiki, June 1995.

[Moga 1997a] A. N. Moga. *Parallel watershed algorithms for image segmentation.* PhD thesis, Tampere University Technology, February 1997.

[Moga 1997b] A. N. Moga et M. Gabbouj. *Parallel image component labeling with watershed transformation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 5, pages 441–450, May 1997.

[Moga 1998a] A. N. Moga, H. Burkhardt et A. Bieniek. *PISA - Parallel image segmentation algorithms.* Rapport technique 2/98, Albert-Ludwigs-Universität, Freiburg, Institut für Informatik, 1998.

[Moga 1998b] A. N. Moga, B. Cramariuc et M. Gabbouj. *Parallel watershed transformation algorithms for image segmentation.* Parallel Computing, vol. 24, no. 14, pages 1981–2001, 1998.

[Moga 1998c] A. N. Moga et M. Gabbouj. *Parallel marker-based image segmentation with watershed transformation.* J. Parallel Distrib. Comput., vol. 51, no. 1, pages 27–45, 1998.

[Moore 1991] I. D. Moore, R. B. Grayson et A. R. Ladson. *Digital terrain modelling : A review of hydrological, geomorphological, and biological applications.* Hydrological Processes, vol. 5, pages 3–30, 1991.

[Moore 1996] I. D. Moore. *Hydrologic Modelling and GIS.* In M. F. Goodchild, L. T. Steyaert, B. O. Parks, C. Johnston et D. Maidmen, editeurs, GIS and Environmental Modelling : Progress and Research Issues, pages 143–148. GIS World Books, Fort Coline : CO, 1996.

[Morris 1988] D. G. Morris et R. G. Heerdegen. *Automatically drained catchment boundaries and channel networks and their hydrological applications.* Geomorphology, vol. 1, pages 131–141, 1988.

[Munagala 1999] Kameshwar Munagala et Abhiram Ranade. *I/O-complexity of graph algorithms.* In Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, SODA '99, pages 687–694, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[Najman 1996] Laurent Najman et Michel Schmitt. *Geodesic Saliency of Watershed Contours and Hierarchical Segmentation.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, no. 12, pages 1163–1173, 1996.

[Najman 2003] L. Najman et Michel Couprie. *Watershed algorithms and contrast preservation.* In Ingela Nyström, Gabriella Sanniti di Baja et Stina Svensson, editeurs, Discrete Geometry for Computer Imagery, 11th International Conference, DGCI 2003, Naples, Italy, November 19-21, 2003, Proceedings, volume 2886 of *LNCS - Lecture Notes in Computer Science*, pages 62–71. Springer-Verlag, November 2003.

[Nanée 2004] CHAHINIAN Nanée. *Paramétrisation multi-critère et multi-échelle d'un modèle hydrologique spatialisé de crue en milieu agricole.* PhD thesis, UNIVERSITE DE MONTPELLIER II, 2004.

[Nodine 1993] Mark H. Nodine, Michael T. Goodrich et Jeffrey Scott Vitter. *Blocking for external graph searching.* In Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS '93, pages 222–232, New York, NY, USA, 1993. ACM.

[O'callaghan 1984] J. F. O'callaghan et D. M. Mark. *The extraction of drainage networks from digital elevation data.* Computer Vision, Graphics and Image Processing, vol. 28, pages 328–344, 1984.

[Osma-Ruiz 2007] V. Osma-Ruiz, J. I. Godino-Llorente, N. Sáenz-Lechón et P. Gómez-Vilda. *An improved watershed algorithm based on efficient computation of shortest paths.* Pattern Recogn., vol. 40, no. 3, pages 1078–1090, 2007.

[Peckham 1995] S. D. Peckham. *River network extraction from large DEMs.* PhD thesis, 1995.

[Pilesjo 1996] P. Pilesjo et Q. Zhou. *A multiple flow direction algorithm and its use for hydrological modelling.* In Geoinformatics'96 Proceedings, pages 366–376, FL, April 26-28 1996. West Palm Beach.

[Planchon 2002] Olivier Planchon et Frédéric Darboux. *A Fast, Simple and Versatile Algorithm to Fill The depressions of digital elevation models.* CATENA, vol. 46, no. 2–3, pages 159–176, 2002.

[Qin 2007] C. Qin, A.-X. Zhu, T. Pei, B. Li, C. Zhou et L. Yang. *An adaptative approach to selecting flow partition exponent for multiple flow direction algorithm.* International Journal Of Geographical Information Science, vol. 21, pages 443–458, 2007.

[Quinn 1991] P. Quinn, K. Beven, P. Chevallier et O. Planchon. *The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models.* Hydrological Processes, vol. 5, pages 9–79, 1991.

[Quinn 1995] P. F Quinn, K. J. Beven et R. Lamb. *The ln(a/tanβ) index : How to calculate it and how to use it within the topmodel framework.* Hydrol. Process., vol. 9, no. 2, pages 161–182, 1995.

[Rambabu 2007] C. Rambabu et I. Chakrabarti. *An efficient immersion-based watershed transform method and its prototype architecture.* Journal of Systems Architecture, vol. 53, no. 4, pages 210 – 226, 2007.

[Roerdink 2001] J. B. T. M. Roerdink et A. Meijster. *The watershed transform : Definitions, algorithms and parallelization strategies.* vol. 41, no. 1-2, pages 187–228, January 2001.

[Saunders 1999] William Saunders. *Preparation of DEMs for Use in Environmental Modeling Analysis.* Esri User Conference, San Diego, California, july 1999.

[Seibert 2007] Jan Seibert et Brian L. McGlynn. *A new triangular multiple flow direction algorithm for computing upslope areas from gridded digital elevation models.* Water Resources Research, vol. 43, 2007.

[Soille 2004] Pierre Soille. *Morphological carving.* Pattern Recognition Letters, vol. 25, no. 5, pages 543 – 550, 2004. <ce :title>Discrete Geometry for Computer Imagery (DGCI'2002)</ce :title>.

[Stoev 2000] S. L. Stoev. *RaFSi - A fast watershed algorithm based on rainfalling simulation.* In WSCG, 2000.

[Tarboton 1988] D. G. Tarboton, Rafael L. Bras et Ignacio Rodriguez-Iturbe. *The fractal nature of river networks.* Water Resources Research, vol. 24, no. 8, pages 1317–1322, 1988.

[Tarboton 1997] D. G. Tarboton. *A new method for the determination of flow directions and upslope areas in grid elevation models.* Water ressources Research, vol. 32, pages 309–319, 1997.

[Tarjan 1983] R. E. Tarjan. Data structure and network algorithms. SIAM - Society for Industrial and Applied Mathematics, 1983.

[Tribe 1992] A. Tribe. *Automated recognition of valley lines and drainage networks from grid digital elevation models : A review and a new method.* Journal of Hydrology, vol. 139, pages 263–293, 1992.

[Ullman 1990] Jeffrey D. Ullman et Mihalis Yannakakis. *The input/output complexity of transitive closure.* In Proceedings of the 1990 ACM SIGMOD international conference on Management of data, SIGMOD '90, pages 44–53, New York, NY, USA, 1990. ACM.

[Vachier 1995] C. Vachier et F. Meyer. *Extinction value, a new measurement of persistence.* In IEEE Workshop on Nonlinear Signal and Image Processing, Neos Marmaras, Greece, June 1995.

[Verdin 1999] K. Verdin et J. Verdin. *A topological system for delineation and codification of the Earth's river basins.* Journal of Hydrology, vol. 218, no. 1-2, pages 1–12, May 1999.

[Vincent 1991] L. Vincent et P. Soille. *Watersheds in digital spaces : An efficient algorithm based on immersion simulations.* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 6, pages 583–598, June 1991.

[Vitter 1992] Jeffrey Scott Vitter et Elizabeth A.M. Shriver. *Algorithms for Parallel Memory I : Two-Level Memories.* Rapport technique, Providence, RI, USA, 1992.

[Vitter 1999] Jeffrey Scott Vitter. *External memory algorithms.* chapitre External memory algorithms and data structures, pages 1–38. American Mathematical Society, Boston, MA, USA, 1999.

[Vitter 2001] Jeffrey Scott Vitter. *External memory algorithms and data structures : dealing with massive data.* ACM Comput. Surv., vol. 33, pages 209–271, June 2001.

[Wallis 2009] Chase Wallis, Dan Watson, David Tarboton et Robert Wallace. *Parallel Flow-Direction and Contributing Area Calculation for Hydrology Analysis in Digital Elevation Models.* In PDPTA, pages 467–472, 2009.

[Wise 1998] S. M. Wise. *The effect of GIS interpolation errors on the use of digital elevation models in geomorphology.* In S. N. Land, K. S. Richards et J. H. Chandler, editeurs, Landform Monitoring, Modelling and Analysis, pages 139–164, New York, 1998. John Wiley and Sons.

[Zanoguera 1999] F. Zanoguera, B. Marcotegui et F. Meyer. *A Toolbox for Interactive Segmentation Based on Nested Partitions.* In IEEE International Conference on Image Processing, Kobe, Japan, October 1999.

Hiep-Thuan DO

Extensibilité des moyens de traitements pour les données issues des vastes systèmes d'informations géographiques

Résumé : Cette thèse s'inscrit dans le cadre de l'évolution des Systèmes d'Informations Géographiques (SIG) et de leur capacité à répondre à des problématiques environnementales qui s'expriment de manière globale et transversale. Dans un contexte où l'information géographique est en plein essor et où la quantité de données disponible ne cesse de croître, le besoin en terme d'outil d'aide à la décision n'a jamais été aussi fort. Cette étude s'attache tout particulièrement au cadre de la résolution de problématiques liées à l'eau et l'environnement lorsque les données deviennent trop volumineuses pour être traitées par des moyens de calculs séquentiels classiques. Nous proposons une plateforme de calculs répartis sur une grappe d'ordinateurs qui parallélise le calcul de la délimitation des bassins versants des grands fleuves et la détermination des flots d'accumulation. A cette fin nous introduisons une technique de calcul parallèle d'une forêt d'arbres couvrants minimums représentant le parcours de l'eau de chaque point du Modèle Numérique de Terrain (MNT) vers la mer. Cette technique débute par une délimitation des cuvettes (ensemble de points allant vers le même minimum local) contenues dans le MNT. Ensuite une hiérarchie de déversement des cuvettes les unes dans les autres est construite jusqu'à obtenir les bassins versants des fleuves. L'étude se poursuit par la description d'un algorithme parallèle pour le calcul très coûteux des flots d'accumulation en chaque point du MNT. Enfin cette thèse présente une version «out-of-core» de nos algorithmes parallèles afin d'étendre la portée de nos travaux à des grappes de dimensions modestes qui ne peuvent pas charger en mémoire la totalité du MNT traité.

Mots clés : Parallélisme, SPMD, Gros volumes de données, Modèle numérique de terrain, Bassin versant, Segmentation d'Images, Arbre Couvrant de Poids Minimum, Ligne de Partage des Eaux.

Extending tools for GIS data

Abstract : My thesis is part of the development of Geographic Information Systems (GIS) and their ability to respond to environmental challenges that are expressed in a global and transversal way. We consider a context in which geographical information is growing, in addition the amount of data available continues to grow. Therefore, the need a tool for decision support has never been stronger. This study aim to solve problems related to water and the environment when the data become too large for sequential computing. The main objective of the thesis proposes a platform for distributed computing on a cluster of computers that parallelizes the watershed computing of major rivers and the determination of the flow accumulation. The idea is based on the construction of a minimal spanning tree, via a hierarchy of graphs, modeling the water route on the DEM toward the ocean. The technique begins from computing catchment basins that are set of pixels for which a drop of water will end the same local minimum. After that, a hierarchy of basins is computed in order to give the catchment basins of the rivers in the DEM. The study continues with a description of a parallel algorithm for computing the global flow accumulation for automatic drainage network extraction in large digital elevation models. Finally, the thesis presents a version «out-of-core» of our parallel algorithms to extend the scope of our work in clusters of size small that cannot load into memory the entire treated DEM.

Keywords : Parallelism, SPMD, Large Datasets, Digital Elevation Model, Catchment basin, Image Segmentation, Minimum Spanning Tree, Watershed

LIFO
Bâtiment IIIA
Rue Léonard de Vinci B.P. 6759
F-45067 ORLEANS Cedex