Parallélisation d'un algorithme d'appariement d'images quasi-dense

Luiz Gustavo Leão Fernandes

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : Informatique : Systèmes et Communications

préparée au laboratoire INFORMATIQUE ET DISTRIBUTION

dans le cadre de l'Ecole Doctorale Mathématiques et Informatique

présentée et soutenue publiquement

par

Luiz Gustavo Leão FERNANDES

le 8 juillet 2002

PARALLÉLISATION D'UN ALGORITHME D'APPARIEMENT D'IMAGES QUASI-DENSE

Directeur de thèse :

Mme. Brigitte PLATEAU

JURY

Mme. Marie-Paule	CANI,	Présidente
M. Pascal	GUITTON,	Rapporteur
M. Raymond	NAMYST,	Rapporteur
Mme. Brigitte	PLATEAU,	Directeur de thèse
M. Yves	DENNEULIN,	Responsable de thèse

À mes parents,
Valcir et Irene.
Et à toi,
Kelly.

Remerciements

Faire une thèse de doctorat normalement n'est pas une chose simple. Encore moins, si on doit la faire à l'étranger. Dans mon cas, finir cette thèse a été un défi considérable en raison de plusieurs événements qui se sont produits entre-temps. Tout seul je n'aurais pas été capable d'en venir a bout. Je voudrais donc remercier ceux qui ont contribué d'une façon ou d'une autre pour que la rédaction de cette page soit finalement possible.

Je remercie en premier lieu mon responsable de thèse Yves Denneulin. En plus de ses qualités scientifiques et professionnelles, je voudrais le remercier pour ses qualités humaines. À plusieurs reprises, il a su me dire les mots qu'il fallait pour m'encourager. Sa direction et son amitié ont, sans doute, été essentielles pour l'aboutissement de ce mémoire. Je remercie aussi Brigitte Plateau pour m'avoir accueilli au sein du laboratoire ID et pour avoir bien voulu diriger cette thèse.

Je tiens à remercier Marie-Paule Cani qui m'a fait l'honneur de présider le jury de ma soutenance. Merci à Pascal Guitton et Raymond Namyst pour avoir accepté de rapporter sur ce document. À n'en pas douter, leurs remarques m'ont permis d'améliorer la qualité de la rédaction.

Un grand merci aussi aux gens du laboratoire ID qui m'ont accueilli si bien. L'ambiance y a toujours été des plus amicales. Je ne vais pas tous les citer car il y n'a pas assez de place, mais je voudrais remercier les permanents qui ont été là depuis le départ : Jean-Louis Roch (un grand connaisseur de foot), Jacques Briat, Denis Trystram, Jacques Chassin de Kergommeaux, Jean-Marc Vincent et Thierry Gautier. Merci aussi à Hélène Emin pour son efficacité et sa sympathie.

Je remercie aussi le gouvernement du Brésil et le CNPq *(Conselho Nacional de Pesquisa e Desenvolvimento*) de m'avoir donné le soutien financier nécessaire pendant ces années de thèse.

Ma venue en France n'aurait pas été possible si au Brésil je n'avais pas été encadré par Philippe Navaux dans mes premières années dans la recherche scientifique. Sa lucidité et son équilibre m'ont toujours servi d'exemple.

Du côté personnel, je commence mes remerciements par mes parents Valcir Dasso Fernandes et Irene Leão Fernandes. Ils ont, chacun à sa façon, contribué pour que mon rêve devienne réalité et je suis sûr qu'aujourd'hui ils se sentent plus heureux que moi.

Je voudrais aussi remercier tous les amis que j'ai connus au long de ces années en France. Les amis brésiliens, français, enfin de toutes nationalités qui ont été présents dans les meilleurs moments de mon séjour à Grenoble (c'est-à-dire : les fêtes, les voyages, le ski, le foot, x-blast, etc ...). Parmi tous, je voudrais dire un merci spécial à Gregory Mounié et Frédérique Chaudier pour l'accueil et l'aide qu'ils nous ont offerts pendant les jours qui précédaient ma soutenance. Un merci spécial aussi à Alfredo Goldman et à Ana Paula Felippe car c'est avec eux que ma femme et moi avons passé les moments les plus marrants pendant toutes ces années de thèse.

J'ai eu la chance de partager mon bureau de travail au labo avec une personne formidable. Je parle de Nicolas Maillard. Au départ c'était un collègue, à la fin il est devenu mon meilleur ami. Je lui dois mon apprentissage du Français et un tas d'autres choses. Les heures de discussion (et il y en a eu des heures ...) pendant les après-midis de thèse resteront les conversations les plus intéressantes et enrichissantes de cette riche période. Un de nos premiers échanges culturels a été sur la Guerre des Étoiles, alors "que la force soit avec toi" mon ami, pour toujours.

J'ai eu aussi la chance d'avoir partagé plusieurs de mes plus belles années de vie avec mon cousin qui est devenu mon frère aîné : Paulo Fernandes. Certainement, c'est lui à qui je dois l'inspiration de certaines des mes valeurs les plus chères. Je ne sais pas comment te remercier, mon vieux, car je te dois trop. J'ai perdu de vue le nombre de fois où tu m'as montré le chemin. Ma victoire est aussi la tienne et tu le sais bien. Le plus beau de tout c'est que, peu importe ce que nous attend demain, je suis sûr qu'on va s'amuser comme des gamins et cela n'a pas de prix.

Finalement, je voudrais remercier celle qui a tout laissé pour me suivre dans cette aventure française. Kelly, je sais combien ces années loin de tes proches ont été dures pour toi et je te remercie pour avoir tout supporté à mon côté avec beaucoup d'amour. Tu m'a appris énormément de choses pendant tout ce temps. Ta joie, ton amitié, ta compréhension, ta capacité, ton intelligence ont souvent été les raisons qui me faisaient continuer à croire. Nous y sommes arrivés tous les deux (toi avant moi, il faut le dire ...) et il n'y a rien au monde qui puisse changer cela. Ça y est, on peut rentrer à la maison tranquilles.

Avant de terminer, je voudrais dire que chaque personne a ses convictions sur la façon de vivre. Moi aussi, j'ai les miennes. Je n'ai jamais pensé au doctorat comme étant un processus purement scientifique ou professionnel, encore moins comme étant un contrat moral. Pour moi, c'était l'opportunité que le gouvernement de mon pays m'avait donnée pour ouvrir mes horizons. Que ce soit au niveau scientifique, mais aussi et principalement au niveau humain. À la fin, tout ce que j'ai envie de vous dire c'est qu'avec l'aide de toutes les personnes citées ci-dessus, je me sens très heureux d'avoir atteint mes objectifs sans oublier mes convictions personnelles. Je pars avec le titre de docteur, mais plus important que cela est l'immense enrichissement d'esprit que la vie en France a pu m'apporter.

Table des matières

Liste des tableaux

Table des figures

Chapitre 1

Introduction

Sommaire

La création de scènes intermédiaires virtuelles à partir de deux scènes réelles du même sujet prises de différents points de vue est une technique de plus en plus utilisée de nos jours. Les applications potentielles sont nombreuses et leur intérêt dans l'emploi d'une telle technique est une meilleure utilisation des ressources à disposition. Un exemple d'application qui peut gagner beaucoup avec l'utilisation de la technique de synthèse d'images virtuelles à partir d'images réelles est les logiciels qui font la gestion de téléconférences. Dans ce cas, l'application de la synthèse d'images virtuelles est d'autant plus utile si le réseau utilisé présente un débit limité.

En fait, il existe plusieurs autres types d'applications potentielles qui pourraient gagner énormément avec l'emploi de la technique de synthèse d'images par ordinateur. On peut citer les simulateurs de vol ou de conduite, le commerce électronique, la conservation de monuments historiques, les jeux, l'entraînement à l'intervention en milieu hostile, la compression d'images autorisant un déplacement de l'observateur et bien d'autres encore...

Mais, il se trouve que la création de vues intermédiaires virtuelles à partir d'une paire d'images réelles est une procédure très coûteuse en temps de calcul. Ce coût est proportionnel à la complexité des images et à leurs dimensions. Des images petites et sans trop de détails peuvent facilement être traités, mais hélas cela n'est pas souvent le cas. Normalement, les dimensions des images qu'on veut traiter sont plus conséquentes et, assez souvent, il s'agit d'images de scènes d'extérieur qui présentent une variété énorme de détails, d'objets difficiles à traiter et de *textures*. Jusqu'à aujourd'hui, les processeurs existants sur le marché ne se sont pas montrés à la hauteur du défi. Le volume de calcul est trop souvent plus grand que ce qu'ils peuvent supporter pour donner des réponses dans un temps acceptable.

Dans ce contexte, la solution qui s'impose ne peut être autre que la mise en parallèle de toute ou, au moins, une des étapes (la plus coûteuse en temps d'exécution) de la procédure de génération de vues virtuelles à partir d'images réelles. Cette conclusion a été celle d'autres chercheurs avant nous et plusieurs travaux dans ce domaine ont produit des résultats intéressants (une présentation plus détaillée en est faite dans le chapitre 5).

Néanmoins, ces travaux ont été menés dans un contexte différent de celui d'aujourd'hui. D'abord, ils étaient basés sur une technique plus ancienne qui était moins robuste car elle ne pouvait pas traiter certaines situations problématiques [LQ00b]. Puis, ils ont été implantés sur des machines parallèles très performantes, mais aussi très chères, car à l'époque (presque dix ans pour certains cas) le concept de grappe de processeurs était très nouveau et les réseaux avaient un débit trop faible.

L'évolution des processeurs a permis d'envisager à long terme d'autres solutions pour la création de vues virtuelles à partir d'images réelles. Récemment, une

nouvelle technique a été proposée dans le travail de doctorat de Maxime Lhuillier [Lhu00] préparé au laboratoire GRAVIR-IMAG dans le centre de recherches INRIA Rhône-Alpes. Cette technique est plus robuste que les précédentes car elle adopte une approche qui ne fait pas d'économies de calcul. La variété d'images qui peut être traitée avec cette nouvelle technique est plus large et la qualité visuelle des images virtuelles qui en résultent est meilleure. Mais les temps d'exécution sont redevenus trop lents pour que cette technique soit employée dans des applications réelles, non académiques.

Lorsque le problème nous a été présenté, nous nous sommes posés la question de l'objectif à atteindre pour la mise en parallèle. À notre avis, une solution pour ce problème devrait offrir des temps d'exécution capables de permettre l'application de la nouvelle technique dans des situations réelles. En plus, il faut éviter les modèles de programmation parallèle tournés vers les machines trop chères et peu répandues (des machines vectorielles par exemple). Ainsi, nous sommes arrivés à la conclusion qu'une version parallèle de la nouvelle technique ne serait intéressante que si elle pouvait être exécutée de façon distribuée sur plusieurs processeurs connectés *via* un réseau rapide et avec un gain de performance suffisant pour que la nouvelle technique de synthèse d'images proposée dans [Lhu00] soit viable dans la pratique.

Ainsi, le choix clair pour le type d'architecture parallèle à viser dans nos études est les grappes de processeurs. Ces architectures deviennent de plus en plus répandues grâce à leur simplicité conceptuelle et à leur coût réduit. De nos jours, des grappes de 8, 16 ou 20 processeurs peuvent être assemblées assez facilement, avec un coût acceptable et un support d'utilisation simple.

Après cette analyse que nous venons de faire du problème, notre opinion est de que, dans les conditions énumérées, l'étude de la parallélisation de la nouvelle technique de synthèse d'images proposée dans [Lhu00] se justifie car les bénéfices seront indéniables. En cas de réussite, la technique auparavant restreinte à certaines applications scientifiques et académiques pourra être appliquée aussi à des applications tournées vers un usage généralisé.

1.1 Problème étudié

Le cadre du problème que nous voulons paralléliser dans ce travail est dans le domaine de la synthèse d'images. Plus précisément, nous allons en fait consacrer nos travaux de recherche à une nouvelle technique proposée dans le travail de thèse de Maxime Lhuillier [Lhu00] intitulé "Modélisation pour la synthèse d'images à partir d'images".

Pour valider les diverses nouvelles techniques proposées dans cette thèse, l'auteur a produit un outil pour faire la synthèse d'images virtuelles à partir d'une

paire d'images source du monde réel. Nous allons nous baser sur le code de cette application pour implanter nos idées de parallélisation.

Pour arriver à l'implantation de cet outil, l'auteur a du faire face à plusieurs problèmes d'ordre théorique. Il cherchait des alternatives aux méthodes traditionnels de synthèse d'images virtuelles de façon à pouvoir traiter avec plus de qualité un spectre plus large de types d'images. Ainsi, de nouvelles techniques et algorithmes ont été nécessaires pour répondre à ses exigences. Mais cette approche a eu un prix : plus de calcul et par conséquent un temps d'exécution plus important pour des images des dimensions à peine moyennes.

Lorsque cet outil nous a été présenté, l'identification de l'étape du mécanisme global de synthèse d'images responsable de la consommation de la plus grosse parcelle de temps avait déjà été faite. Il s'agissait d'un nouvel algorithme d'appariement d'images basé sur la modification d'une ancienne technique de croissance de régions. Cet algorithme, appelé par l'auteur algorithme de propagation, est employé pour établir le plus grand nombre possible de correspondances entre les points (ou pixels) qui forment les deux images sources.

L'algorithme de propagation démarre avec un ensemble réduit de paires de points de la surface des images appelées appariements de départ. Les points qui forment les appariements de départ sont ceux qui possèdent des caractéristiques facilement remarquables selon un critère quelconque. À partir de ces paires, la technique de croissance de régions est employée pour apparier la plus grosse extension possible de la surface des images.

Il existe plusieurs approches différentes qui peuvent être utilisées pour une technique de croissance de régions. Dans son travail, M. Lhuillier a fait le choix d'une stratégie appelée "meilleur d'abord", c'est-à-dire que les appariements de départ sont pris suivant un ordre établi par un critère qui peut être lié à l'intensité ou la couleur des pixels. L'innovation proposée dans la thèse de M. Lhuillier est la re-alimentation de l'ensemble d'appariements de départ avec les nouveaux appariements calculés dans l'itération précédente de l'algorithme. En résumé, il s'agit d'un algorithme d'appariement qui possède deux caractéristiques fondamentales :

- l'emploi d'une recherche globale dans son ensemble de paires de départ pour déterminer la prochaine paire à utiliser ;
- l'actualisation de cet ensemble de paires de départ à la fin de chaque nouvelle itération de l'algorithme avec la paire la plus récemment appariée.

Nous venons donc de définir l'élément qui sera l'objet de nos études de parallélisation dans cette thèse : **le nouvel algorithme d'appariements dense d'images, aussi appelé algorithme de propagation**. Nous verrons dans la suite (chapitre 3) les mesures de temps d'exécution qui confirment cette étape comme étant de loin

la plus lente de toutes les étapes nécessaires pour aller jusqu'à la création effective de nouvelles vues virtuelles dans un outil de synthèse d'images à partir d'images réelles.

1.2 Objectifs scientifiques

L'algorithme de propagation que nous venons de décrire dans la section précédente s'insère dans la catégorie des problèmes irréguliers adaptatifs. Cette catégorie de problèmes présente une évolution des données de départ au fur et à mesure que le calcul est réalisé. Ce genre de problème possède un comportement dynamique qui commence en ayant un haut degré de parallélisme qui diminue très rapidement jusqu'à un degré beaucoup plus bas.

De plus, cet algorithme utilise une méthode basée sur une recherche globale dans un ensemble de paires initiales pour trouver celle qui potentiellement peut déclencher une propagation plus étendue. L'application d'une telle stratégie vise l'obtention de niveaux plus élevés pour la qualité de l'appariement final des images source. Or, une telle approche est très difficile à paralléliser efficacement. Il nous faudra donc trouver une stratégie qui soit capable de la remplacer dans un contexte distribué.

Finalement, comme nous avons opté pour viser des machines du type grappes de processeurs, le modèle de programmation parallèle utilisé pour l'implantation sera celui de l'échange de messages. Ainsi, la quête de solutions parallèles pour l'algorithme de propagation sera orientée par les caractéristiques et limitations de ce modèle.

En résumé, nous pouvons affirmer que *notre travail vise la proposition d'une version parallèle pour l'algorithme d'appariement quasi-dense d'images lequel est basé sur une stratégie adaptative globale. Cette version doit être capable de préserver la qualité du résultat final obtenu par la version séquentielle de l'algorithme de propagation tout en réduisant le temps d'exécution dans un contexte de programmation tourné vers les grappes de processeurs.*

La démarche suivie est la suivante :

1. analyse du contexte visant la compréhension du problème ;
2. propositions de solutions cadrées par le modèle de programmation choisi ;
3. implantations de solutions parallèles de l'algorithme de propagation respectant les limitations du logiciel de synthèse d'images dans lequel l'algorithme s'insère ;
4. validation des résultats à travers des comparaisons avec le temps d'exécution de la version séquentielle.

1.3 Organisation du manuscrit

Ce document est composé de cette introduction plus sept chapitres organisés en cinq parties plus un glossaire contenant les définitions de certains termes spécifiques au domaine de l'imagerie. La description du contenu et de la structure de chaque chapitre est la suivante :

- Le premier chapitre contient l'Introduction du manuscrit laquelle présente le contexte de ce travail, le motivation de cette recherche et ses objectifs scientifiques, la démarche suivie pour conclure sur l'organisation du manuscrit.

- La deuxième partie est composée de deux chapitres qui introduisent l'application cible utilisée dans ce travail. Le chapitre deux fait une description des phases nécessaires pour arriver à la synthèse d'images réelles selon [Lhu00]. Dans le chapitre trois, nous nous focalisons sur la phase de l'application qui nous intéresse, c'est-à-dire l'algorithme de croissance de régions basé sur une stratégie de propagation "meilleur d'abord".

- La troisième partie présente rapidement le domaine du calcul parallèle. Les concepts de base sont révisés et organisés de façon à permettre au lecteur peu familier avec le domaine du calcul parallèle de comprendre les avantages et les inconvénients de l'utilisation d'une grappe de processeurs.

- La quatrième partie est divisée en trois chapitres et présente les solutions proposées dans ce travail pour la parallélisation d'un algorithme de croissance de régions utilisant une grappe de processeurs. Le chapitre 5 introduit les premières études sur le problème à travers la discussion sur l'implantation de deux approches différentes et leurs résultats. Dans la suite, le chapitre 6 part des conclusions du chapitre 5 pour arriver à une approche qui essaye de profiter des avantages de deux premières. Ensuite, le chapitre 7 décrit deux idées pour affiner un peu plus la version proposée dans le chapitre 6.

- La conclusion contient une analyse des contributions apportées par notre travail. Une discussion sur les travaux futurs est aussi présentée.

- Le glossaire à la fin du manuscrit a pour objectif fournir une aide rapide au lecteur qui n'est pas familiarisé avec les termes du domaine de l'imagerie. Dans le texte, les termes qui en font partie sont écrits en italique.

Première partie

Application Cible

Chapitre 2

Synthèse d'Images à partir de Vues Réelles

Sommaire

2.1 Introduction

L'objectif principal de la synthèse d'images à partir de vues réelles est de permettre, uniquement à l'aide de quelques photos d'une scène quelconque, la création de nouvelles vues virtuelles intermédiaires de cette même scène en simulant le déplacement de la caméra qui a pris les photos.

Utiliser des images réelles pour créer des images de synthèse offre un double avantage : l'élimination du difficile problème de la modélisation géométrique et photométrique complète du monde réel et l'accélération de l'étape de rendu. En fait, les vues disponibles de la scène contiennent les informations géométriques nécessaires et les informations de *texture* et couleurs sont déjà rendues car les objets sont éclairés par une source de lumière réelle. Ces informations peuvent donc être reprises et transformées par un système de synthèse d'images afin de générer de nouvelles vues de cette même scène.

Les applications potentielles sont nombreuses : les simulateurs de vols ou de conduite, le commerce électronique, la conservation de monuments historiques, les jeux, l'entraînement à l'intervention en milieu hostile, la compression d'images autorisant un déplacement de l'observateur, etc.

Dans ce chapitre nous présentons un résumé des concepts de base et des étapes nécessaires pour réaliser la synthèse d'images à partir de vues réelles. Nous ne cherchons pas à faire un panorama complet du domaine. Nous avons choisi d'orienter notre analyse par le travail de [Lhu00], duquel ce chapitre est inspiré. Cela se justifie car une des étapes de son travail sera l'objet duquel seront faites les études de parallélisation (le sujet principal de cette thèse). Il s'agit de l'étape de mise en correspondance, laquelle aura ses concepts de base présentés dans la section 2.3 et qui fera l'objet d'une analyse plus détaillée dans le chapitre 3.

Enfin, pour illustrer les concepts présentés dans ce chapitre, nous allons utiliser des vues obtenues avec l'exécution séquentielle de l'application citée ci-dessus. Ces vues sont des résultats intermédiaires fournis par l'application utilisant une paire d'images qui représentent une scène réelle d'extérieur. Elles sont générées à la fin de l'exécution de chaque étape importante de l'algorithme.

2.2 Les étapes

L'approche traditionnelle pour la synthèse de nouvelles vues d'une scène 3D à partir de vues existantes consiste à construire un modèle 3D unique, dont le rendu s'effectue de manière classique. Malheureusement, la construction d'un tel modèle dans un contexte général sans intervention manuelle est très compliquée, essentiel-

lement à cause des problèmes mal posés de la mise en correspondance et de la reconstruction.

Une approche plus réaliste consiste à représenter la scène par la collection d'images donnée au départ et a y prélever et projeter directement la *texture* dans l'image à synthétiser en fonction de la position de la caméra. Dans [Lhu00], un nouveau système de rendu basé-image a été développé pour la synthèse d'images. Il se décompose en trois étapes : la mise en correspondance, les triangulations et le rendu des nouvelles vues.

La première étape est la plus délicate et est aussi celle qui est la plus gourmande en temps d'exécution. Il s'agit de définir les correspondances entre les pixels des deux images. Une analyse rapide du principe de la technique employée ainsi que les concepts plus importants pour sa compréhension globale sont présentés.

Dans la seconde étape, un algorithme de conversion de ces correspondances en une structure plus adaptée à la troisième et dernière étape est discuté. Cette structure décrit la transformation entre les deux images source à l'aide d'une technique de triangulations contraintes, en tenant compte des difficultés de visualisation comme les zones partiellement occultées dans une des images sources.

Finalement, la troisième étape décrit le mécanisme pour créer la nouvelle vue.

2.3 Mise en correspondance

L'appariement (mise en correspondance) d'images est une de plus importantes lignes de recherche pour la communauté spécialisée en vision par ordinateur. Les applications des techniques d'appariement vont de la reconnaissance d'objets à la perception 3D et sont utiles en domaines de recherche comme la robotique et l'indexation de vidéos.

Le point de départ pour un algorithme d'appariement est l'identification, dans les images source, des points qui possèdent des caractéristiques qui serviront à les différencier des autres. Ces points sont localisés dans les lieux les plus texturés des images source. Une fois les points identifiés, il faut chercher pour chacun d'eux son correspondant dans l'autre image. En général, le choix des correspondants est basé sur une mesure de ressemblance. À la fin de cette étape, on dispose d'un nombre de paires de points qui constitueront l'ensemble des appariements de départ (germes) pour initialiser un algorithme du type croissance de régions [Mon87]. Cet algorithme propagera l'appariement des lieux les plus texturés des images vers ceux qui le sont moins.

Dans cette section, une présentation des hypothèses de base de la mise en cor-

respondance sera faite. Après, on expliquera un mécanisme de détection des points à apparier, le concept de mesure de ressemblance pour terminer sur une analyse des trois principaux types d'algorithmes de croissance de régions.

2.3.1 Hypothèses de base

La mise en correspondance consiste à déterminer quelles sont les projections qui se correspondent dans un ensemble d'images source. En d'autres termes, c'est un algorithme qui cherche a déterminer quels sont les points 2D qui représentent les projections d'un même point 3D (figure 2.1). Un appariement est donc un n-uplet de points 2D en correspondance, avec $n \geq 2$.

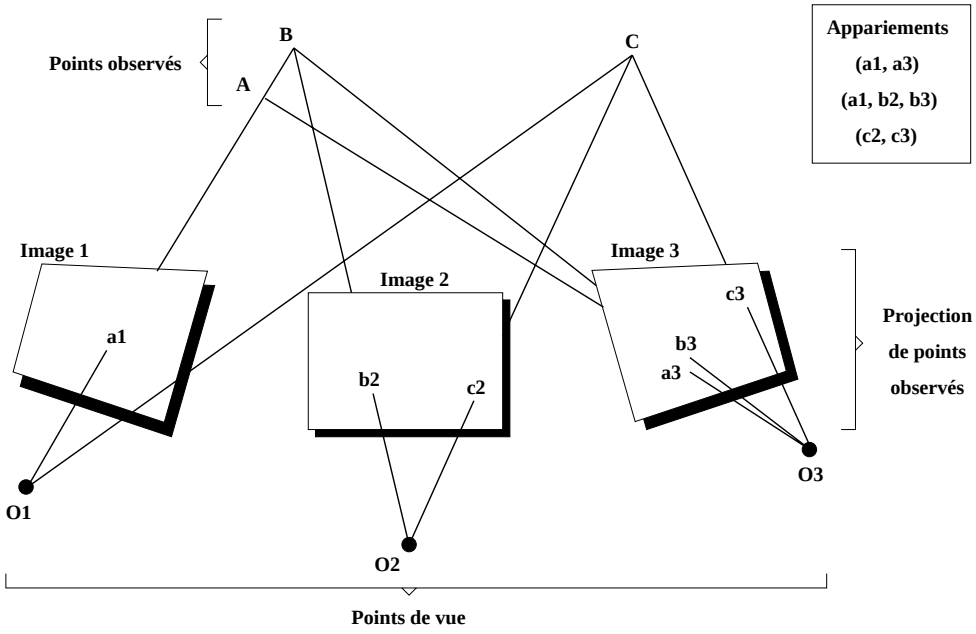


FIG. 2.1 – Définition d'un appariement

Sur la figure 2.1, l'ensemble d'appariements est égal à (a_1,a_3),(a_1,b_2,b_3),(c_2,c_3). Pour trouver un appariement de c_2 dans l'image I_3, il nous faudrait un moyen de calculer c_3. Cependant, sans plus d'informations, c_3 peut se trouver n'importe où en I_3 et ni son aspect, ni sa position ne sont liés à ceux de c_2 en I_2.

Une façon de contourner ce problème est de supposer que c_2 et c_3 sont de projections d'un même point (C dans la figure). De cette façon, les algorithmes d'appariement peuvent travailler avec l'hypothèse que les signaux des images I_2 et I_3 se ressemblent autour des points c_2 et c_3. Ainsi, il est possible de calculer systématiquement des mesures de ressemblance entre les voisinages des pixels c_2 dans I_2 et de ses correspondants potentiels dans I_3, tout en conservant les candidats

les plus ressemblants.

Cette méthode fonctionne bien, sauf si l'un des cas suivants se produit :

1. Premier cas : le point C appartient à la *courbure d'une surface* laquelle n'a pas le même aspect depuis les points de vue O_2 et O_3 ;
2. Deuxième cas : le point C appartient à un objet d'un matériau lequel n'a pas le même aspect depuis les points de vue O_2 et O_3 ;
3. Troisième cas : le point C est occulté dans une des vues par un objet opaque quelconque qui se trouve sur le trajet du rayon lumineux O_3-C.

Si on considère que travailler avec une surface plane, parallèle aux caméras et avec tous les points de vue confondus n'a pas beaucoup d'intérêt, le premier cas se produit toujours. Dans ces circonstances, les mesures de ressemblance doivent être robustes du fait que la surface de l'image est tangente aux directions de vue et que les points de vue sont éloignés. Ainsi, pour limiter l'influence des erreurs locales qui en résultent, il est nécessaire que les mesures de ressemblances intègrent un grand nombre des pixels voisins.

Le deuxième cas est le moins souvent vrai des trois. Il se produit lorsque les matériaux présents dans l'image présentent des *réflexions spéculaires*. De tels reflets ne peuvent pas être appariés car leur position sur l'objet dépend de la position de l'observateur. Les changements de luminosité moins radicaux peuvent être contrebalancés avec succès par des mesures de ressemblance dites centrées qui se basent sur des gradients d'intensité locaux plutôt que sur l'intensité des pixels.

Enfin, le troisième cas est souvent vrai. Dans des scènes d'extérieur, les occultations sont presque toujours présentes. Elles le sont moins dans des scènes d'intérieur et si les points de vue sont suffisamment rapprochés. Il existe des mesures de ressemblance robustes à ce problème, elles sont basées sur la définition d'une fenêtre autour du pixel en question suivi du calcul d'une mesure de corrélation traditionnelle seulement sur les pixels qui sont ou non occultés.

2.3.2 Mesures de ressemblance

Les mesures de ressemblance locales sont la base des toutes les méthodes d'appariement. A travers ces mesures, l'algorithme d'appariement est capable d'établir une distance de similarité entre M pixels dans N images (avec généralement $N = 2$). Dans l'immense majorité des cas, les mesures de ressemblance sont implantées par des mesures de corrélation. De telle mesures intègrent les différences d'intensité sur des voisinages rectangulaires (généralement carrés) des pixels considérés.

Dans [Bla98], plusieurs mesures de ressemblance sont étudiées. Nous avons choisi d'illustrer dans ce travail la mesure de ressemblance adoptée par [Lhu00] dans l'implantation de l'application cible de cette thèse. Il s'agit de la mesure ZNCC (corrélation croisée centrée et normalisée), dont la définition est donnée par :

$$ZNCC_k = \frac{\sum_{D \in V_k} (I_1(X_1+D) - \overline{I}_1^k(X_1))(I_2(X_2+D) - \overline{I}_2^k(X_2))}{\sqrt{\sum_{D \in V_k} (I_1(X_1+D) - \overline{I}_1^k(X_1))^2} \sqrt{\sum_{D \in V_k} (I_2(X_2+D) - \overline{I}_2^k(X_2))^2}}$$

où $\overline{I}_1^k(X_1)$ et $\overline{I}_2^k(X_2)$ sont les moyennes des *luminances* dans les voisinages de dimension V_k centrés en X_1 et X_2. Cette mesure est robuste aux *réflexions spéculaires* et ne pénalise pas les zones où la *luminescence* varie beaucoup (arêtes incluses), par contre elle pénalise les zones où la *luminescence* varie peu. Une autre caractéristique marquante de cette mesure est le fait qu'elle suppose que les deux voisinages centrés en X_1 et X_2 sont les projections d'une même portion de surface dans l'espace. Cela a pour conséquence que ces voisinages ne doivent pas intersecter les contours d'occultations dans les images, et que la distorsion entre les deux vues due à la perspective est négligeable.

2.3.3 Détection des points de départ

Une question des plus importantes dans l'appariement d'images est de savoir quels points des images peuvent être mis en correspondance. Évidemment, ce n'est pas possible d'apparier tous les points qui composent les images. En réalité, un algorithme de mise en correspondance ne peut pas décider quel point apparier avec un autre d'une autre image si plusieurs points dans la même image lui ressemblent. Une condition nécessaire pour que l'on puisse trouver un correspondant fiable à un point sans faire d'hypothèses sur la scène est de pouvoir distinguer ce point des autres.

Pour illustrer, l'image à gauche dans la figure 2.2 montre trois possibilités des localisation de points dans une image :

1. sur un coin (point A) ;
2. sur une arête (point B) ;
3. dans une région uniforme (point C) ;

Pour chaque point, on regarde si on est capable de le distinguer de tous les points de son voisinage immédiat. Dans la figure 2.2, on note que A se distingue de ses voisins dans toutes les directions dans l'image, B ne se distingue pas de ses voisins situés sur l'arête et enfin C ne se distingue d'aucun de ses voisins. Donc, dans cet exemple, seulement A peut être apparié individuellement. Cette description simplifiée résume les principes de la mesure de confiance de Moravec [Mor77], laquelle établit un moyen de sélectionner les points d'une image pour

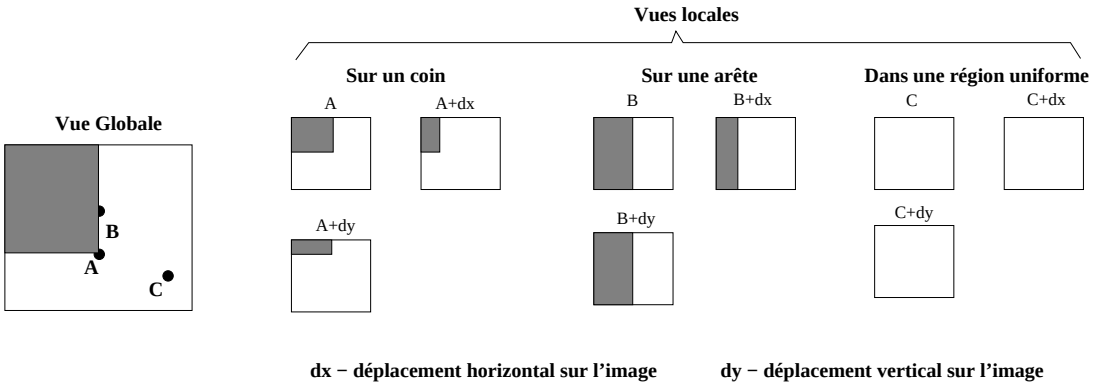


FIG. 2.2 – Différenciation des points d'une image.

lesquels le voisinage change de façon significative.

La mesure de confiance de Moravec est considérée comme étant la plus simple des mesures de confiance et bien d'autres mesures ont été définies [HS88] [SMB98] [ST94]. Les reproches les plus souvent faits à cette mesure sont la non autorisation de rotations et de facteurs d'échelles ainsi que de faire contribuer tous les points du voisinage avec la même importance dans son calcul.

Dans l'implantation de l'application cible, [Lhu00] a choisi d'utiliser comme mesure de confiance une variante de points de Harris [HS88] proposée par [SMB98]. Cette mesure sélectionne les points d'intérêt qui sont détectés comme étant les points qui réalisent les maxima locaux des différentes *textures* présentes dans l'image (fonction d'auto-corrélation).

Dans la figure 2.3 on peut voir les résultats de l'emploi des méthodes citées sur une paire d'images de test. Les points d'intérêt sont représentés par des croix noires sur les deux images. Jusqu'ici, il n'y a pas forcément de correspondance entre tous les points d'intérêt détectés.

2.3.4 Algorithmes d'appariement

Une fois les points d'intérêt sélectionnés dans les images sources, il existent deux stratégies possibles : soit on se contente d'apparier seulement les points les plus fiables au sens d'une mesure de confiance (appariement épars), soit on cherche à apparier le maximum de pixels des images à partir des points de départ (appariement dense). Dans le premier cas, les avantages sont la rapidité d'exécution, la fiabilité et la précision des points mis en correspondance. Cependant, la description d'une scène par un ensemble épars de points ne suffit parfois pas pour la synthèse d'images, surtout au niveau des occultations. Dans ces cas une mise en

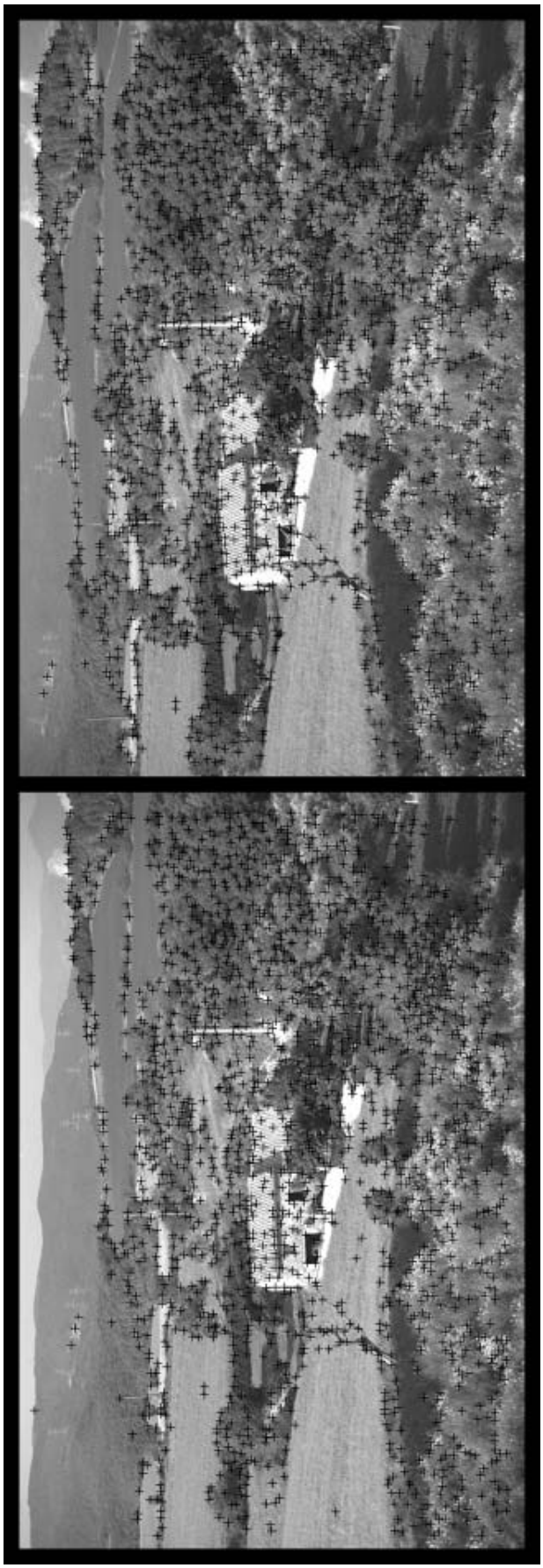


FIG. 2.3 – Exemples de points d'intérêt : représentés par les croix noires.

correspondance plus dense est nécessaire.

Nous avons choisi de présenter dans ce travail un résumé des approches éparse, dense et quasi-dense. Notre intention est de fournir au lecteur des éléments qui permettront de comprendre les choix faits par [Lhu00] dans son travail. Plusieurs travaux énumérant les principales méthodes de mise en correspondance ont été réalisés, notamment dans [BDB94] [Bla98] [DJ89] [Kos93].

Appariement épars

L'appariement épars consiste à ne mettre en correspondance que les points les plus fiables (points d'intérêt) détectés sur les images. Ces points représentent les maxima locaux d'une mesure de confiance donnée. Une description de la méthode classique d'appariement épars est décrite dans la suite.

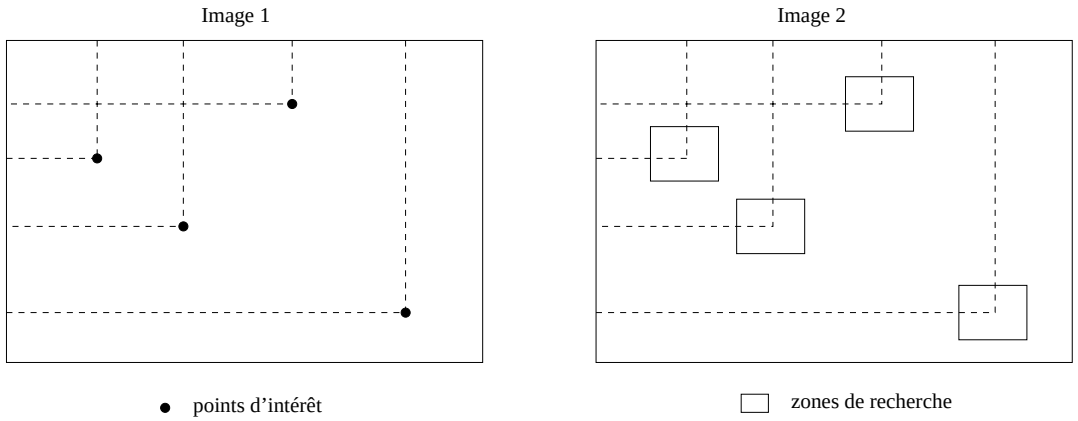


FIG. 2.4 – Principe de l'appariement épars.

Pour chaque point d'intérêt détecté dans la première image, une zone de recherche dans la seconde image est définie (figure 2.4). La définition d'une telle zone de recherche pour la correspondance est nécessaire car les critères locaux de détection et de similarité ne suffisent pas à éliminer de nombreuses fausses mises en correspondance. La zone de recherche est centrée sur la position du point à apparier, et sa taille est à choisir en fonction du contexte. Pour une application qui a besoin de rapidité, par exemple, la mise en correspondance doit utiliser une zone de recherche de dimensions réduites. D'un autre côté, si l'on cherche a privilégier la qualité du résultat final, c'est intéressant d'utiliser une zone de recherche plus grande.

Avec la définition de la zone de recherche dans la seconde image, on doit y définir pour chaque point d'intérêt de la première image des points d'intérêt correspondants possibles qui lui ressemblent en utilisant une mesure de similarité (souvent ZNCC). Il

existe alors plusieurs méthodes pour choisir un correspondant parmi tous les candidats [Bla98]. Une méthode usuelle est la vérification par consistance croisée [Fua91]. Cette méthode consiste pour chaque point de la première image à choisir le meilleur de la seconde image utilisant une mesure de similarité et vice-versa, pour ensuite ne garder que les couples des points communs à ces deux opérations.

Dans la figure 2.5, les croix blanches identifient les appariements obtenus à partir des points d'intérêt. Pour chaque croix représentée dans la première image, il y a une croix correspondante dans la seconde image. Les appariements montrés, seront utilisés dans l'étape suivante comme graines pour l'algorithme d'appariement quasi-dense.

Appariement (quasi-) dense

La mise en correspondance (quasi-) dense se caractérise par une quête d'extension de la zone appariée par rapport aux méthodes éparses. On ne se contente plus d'apparier quelques points qui représentent les endroits les plus facilement identifiables dans la surface des images, mais au contraire on veut étendre au maximum l'association des point aux régions moins texturées des images.

La différence entre les méthodes de mise en correspondance quasi-dense et dense réside dans la forme de leur calcul. Si la première est basée plutôt sur un calcul local autour des points donnés situés dans des régions de *texture* similaire, la seconde fait une recherche plutôt globale sur toute la surface de l'image qu'elle a à sa disposition [SZ99].

Dans la mise en correspondance dite dense chaque possibilité d'appariement possède un coût et le but est de trouver celle qui minimise ce coût. Celui-ci est habituellement obtenu par la somme de deux termes : un terme lié aux caractéristiques des données, lequel accumule les différences (*contraste*, *luminance*, etc) entre chaque pixel de la première image et ses possibles correspondants dans la seconde image, et un terme de *lissage*. Cette approche globale fait que les zones des images avec peu ou aucune information peuvent être appariées de façon unique. Cependant, pour faciliter la convergence de la méthode d'optimisation du coût, il faut disposer d'une initialisation qui soit suffisamment proche de la solution optimale pour espérer sortir des minima locaux et atteindre la mise en correspondance optimale. C'est pour cette raison que pour pouvoir utiliser une méthode dense, il est nécessaire de 1) ou bien supposer que les déformations entre les images sont faibles, ou bien 2) partir d'une solution initiale obtenue à partir d'une méthode quasi-dense. Dans [Fal94], [HS81] et [IB94] on peut trouver des exemples de méthodes denses (globales).

Dans le cas quasi-dense, les appariements sont calculés les uns à la suite des autres dans les zones avec beaucoup d'information, parfois à l'aide d'une méthode éparse, en intégrant des contraintes de continuité entre appariements voisins. Cela

FIG. 2.5 – Exemple d'appariement éparse : représenté par les croix blanches.

revient à dire que l'ensemble final d'appariements consiste en une distribution éparse de régions denses. Cette approche est plus réaliste car plus facile à calculer que l'approche dense. Quelques exemples des méthodes d'appariements quasi-dense (locales) sont décrites en [BVZ98], [Fua91] et [PJF85]

Dans la figure 2.6, on peut voir le résultat de l'application de l'algorithme d'appariement quasi-dense sur les images teste. Les régions correctement appariées sont couvertes par une maille quadrillée. En regardant avec attention, il est possible de voir une zone occulte dans la première vue, proche de l'arbre placé au centre de l'image.

2.4 Triangulations

À la fin de l'étape de mise en correspondance, on dispose d'un ensemble d'appariements, lequel peut être considéré comme l'élément de base qui servira à la création des nouvelles vues synthétisées. Pour certains auteurs [BM97] [Sch96] [SD96], cela suffit car ils utilisent leur ensemble d'appariements (quasi-) dense pour former pixel par pixel la reconstruction de la surface des nouvelles vues. Le principal avantage de ce type de rendu basé sur pixels est qu'un possible faux appariement devient alors insignifiant dans la multitude d'appariements corrects.

Néanmoins, il existe une autre approche qui suggère la création d'une structure intermédiaire qui contient une transformation de l'ensemble d'appariements quasi-dense (ou dense) en triangles dépendants qui se correspondent en chacune des images source. La création de cette structure est faite par une technique appelée triangulation. Il existe plusieurs avantages à l'utiliser une telle technique, parmi lesquelles on peut citer la réduction de la complexité du calcul et de la mémoire nécessaire, l'explicitation des rapports topologiques entre les images et une interpolation plus facile de régions non appariées.

On peut citer plusieurs travaux qui ont proposé des techniques de triangulation [DDP97] [FLR+98] [FL94] [GSB97]. Dans ce rapport, nous allons faire une présentation rapide de la technique proposée par M. Lhuillier [LQ99b] [LQ99a] [Lhu00] [LQ00a] à titre d'illustration. Cette technique a été nommée Triangulation Conjointe de Vues (en anglais *Joint View Triangulations*). Dorénavant, nous ferons référence à cette technique dans le texte à travers l'acronyme **TCV**.

2.4.1 Principes de la TCV

La TCV pour deux vues est composée d'une paire de triangulations contraintes et cohérentes qui se correspondent sur la surface des images. Les triangulations

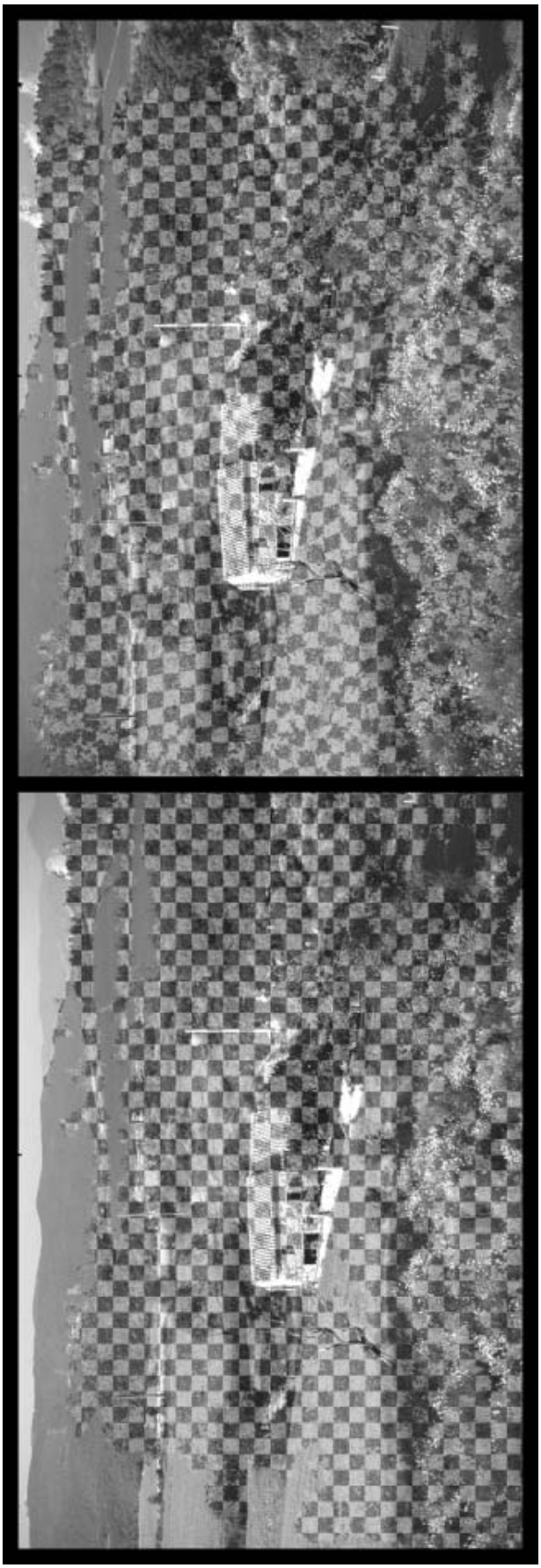


FIG. 2.6 – Exemple d'appariement quasi-dense : régions quadrillées.

sont construites à partir de l'ensemble d'appariements obtenu dans l'étape de mise en correspondance.

L'application de la technique de TCV a pour objectif d'offrir :

1. une représentation basée sur l'image avec un ensemble réduit de primitives ;
2. une séparation entre les régions appariées et celles partiellement occultées pour chaque image source ;
3. une correspondance entre les primitives qui représentent les régions appariées dans la paire d'images source.

La cohérence pour les zones appariées des deux images source est définie par les deux dernières caractéristiques citées ci-dessus.

La TCV est composée de triangles qui peuvent être appariés ou non appariés. Un triangle apparié couvre une région de points (pixels) appariés et par conséquence, les triangles non appariés couvrent les régions de l'image qui ne sont pas appariées. La séparation entre les deux types de triangles est faite par des arêtes contraintes. Pour que la seconde caractéristique soit possible, les régions partiellement occultées doivent coïncider avec les triangles non appariés. Enfin, pour que la troisième caractéristique soit présente, une correspondance bijective entre les sommets des triangles et entre les arêtes contraintes est nécessaire.

Les arêtes contraintes sont importantes, car elles délimitent les frontières entre les zones appariées et non appariées sur chaque image. Une façon d'identifier les arêtes contraintes dans les deux vues est l'utilisation d'une grille régulière pour approcher par un polygone les zones appariées. Cependant, l'utilisation de cette technique produit parfois des artefacts non négligeables pendant la phase de rendu, notamment aux contours des *zones occultées*. Un exemple d'artefact possible est celui de la ligne brisée sur la figure 2.7 où nous pouvons voir que l'image de la boîte grise est tordue dans l'image à droite. Cela serait causé par une triangulation inadéquate.

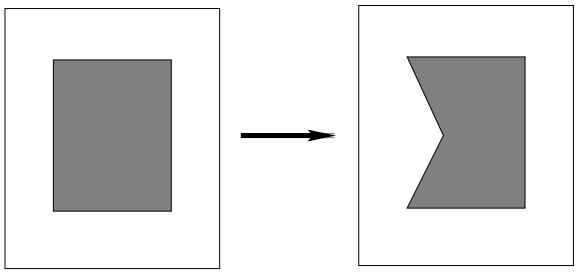


FIG. 2.7 – L'artefact de la ligne brisée.

Pour remédier au problème de la ligne brisée, d'autres types d'arêtes contraintes existent :

- **les arêtes de** *luminance* - ce sont des lignes dans l'image où la norme du gradient de la *luminance* est localement maximale dans le sens du gradient ;
- **les objets fins et verticaux** - ils doivent être identifiés pour recevoir un traitement particulier car ils seront mal approximés polygonalement par les grilles régulières.

Dans l'implantation de son application, [Lhu00] a utilisé la triangulation de Delaunay [PS85] contrainte par les trois types d'arêtes énumérées précédemment. Cette méthode, normalement contrainte uniquement par les sommets des triangles, a pour propriété d'induire la *solution la plus lisse* [Rip90].

2.4.2 Construction des triangles

Dans cette section, nous allons décrire brièvement la méthode de construction de triangles employée par [Lhu00] dans son travail.

Avant de commencer la construction des triangles proprement dite, un prétraitement est nécessaire. Il s'agit d'une régularisation de l'ensemble d'appariements obtenu à la fin de l'étape de mise en correspondance avec l'objectif d'éliminer le bruit résiduel et les mauvais appariements.

Pour réaliser cette régularisation il faut d'abord partitionner la première image en une grille régulière pour approcher par un polygone les zones appariées. Dans cette technique, une division de la première image est faite à l'aide d'une grille régulière en petites portions carrées. Dans chaque portion, il faut considérer tous les points appariés provenant de l'appariement quasi-dense pour essayer d'estimer une homographie afin d'obtenir une portion de surface potentielle. Puis, il est nécessaire d'essayer d'apparier chacune de ces portions avec une correspondante dans la seconde image. À la fin de cette étape, on dispose d'un ensemble de portions appariées dans les deux images libéré des faux appariements et qui convient mieux aux besoins de la phase de construction des triangles.

La TCV peut alors commencer à être construite. Sa construction est basée sur la fusion successive des portions appariées (2.8). Le point de départ est constitué de deux triangles non appariés dans chacune des deux images. Chaque portion appariée dans la phase précédente est alors fusionnée à l'ensemble courant de triangles appariés dans les deux images à condition que les arêtes qui définissent son contour n'intersectent pas le contour courant de ce même ensemble de triangles.

Ce schéma évolutif permet la fusion robuste d'un sous-ensemble de portions

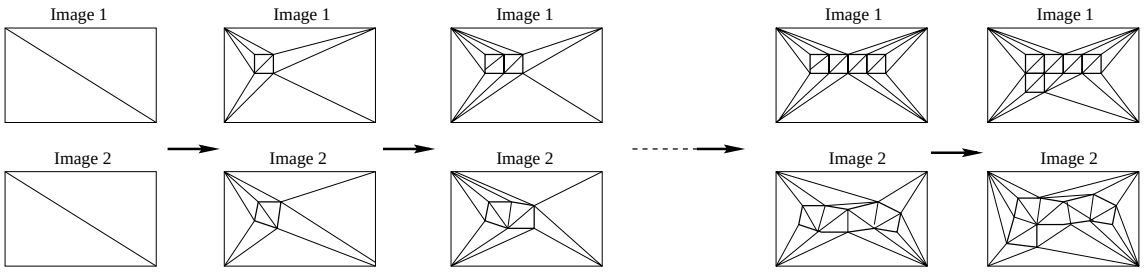


FIG. 2.8 – Formation de la TCV.

appariées qui ne s'intersectent pas tout en gardant la cohérence entre les deux images source. Cela équivaut à une correspondance bijective entre les sommets des triangles et les arêtes qui forment les contours des portions (cf. section 2.4.2). L'algorithme évolue ligne par ligne, du haut vers le bas de la grille créée dans la première image.

Finalement, la structure est améliorée par l'insertion dans l'ensemble des triangles appariées des arêtes de *luminance* et des objets fins et verticaux.

La figure 2.9 montre les résultats obtenus sur les deux vues après l'application de l'algorithme de la TCV sur l'ensemble de points appariés obtenus à la fin de l'étape de mise en correspondance.

2.5 Rendu : création de nouvelles vues à partir de la TCV

Une fois conclue l'étape des triangulations, on dispose d'une structure qui contient une correspondance entre tous les sommets des triangles construits dans les images source. Cette information peut désormais être utilisée pour générer des nouvelles vues intermédiaires entre les deux images source. Dans la suite nous allons décrire les étapes de base pour l'utilisation de la TCV pour le rendu de nouvelles vues.

2.5.1 Principes

Il y a un certain nombre de caractéristiques qui sont désirables lors de la réalisation d'un rendu de nouvelles vues. Parmi elles, on peut citer :

1. la continuité lorsqu'on fait la transition d'une image source à l'autre ;
2. l'absence des zones qui ne contiennent pas de renseignements ;

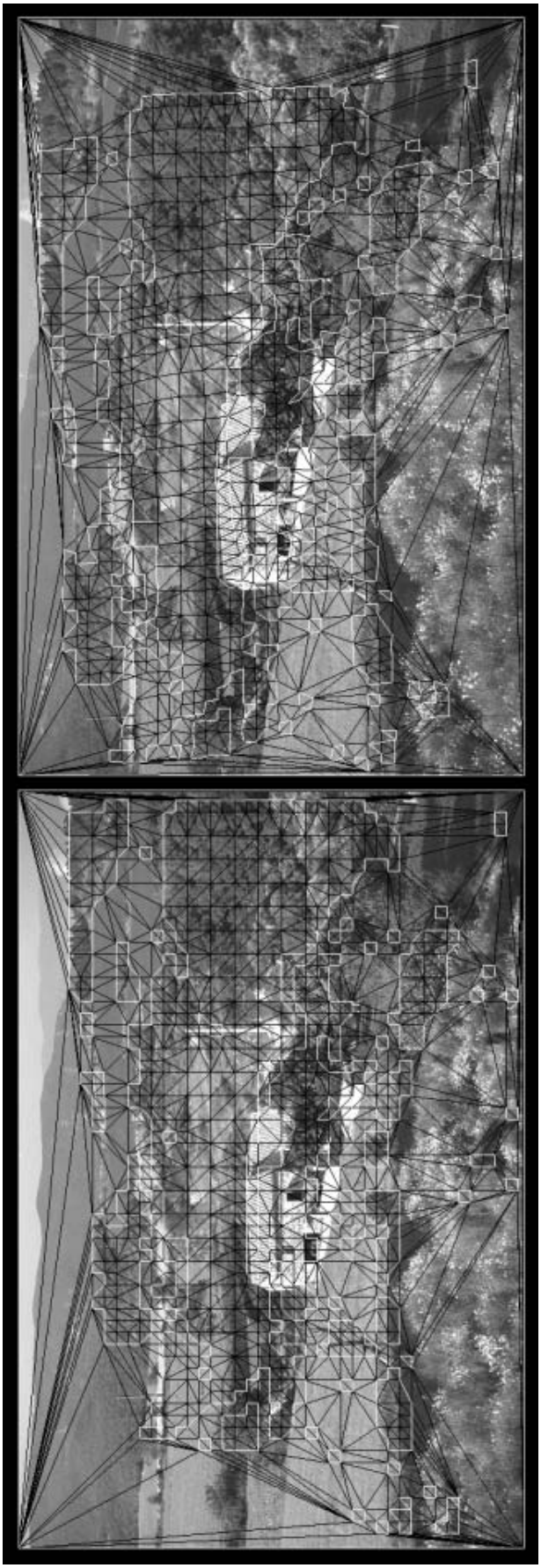


FIG. 2.9 – Exemple de triangulation.

3. l'élimination de parties cachées par des objets en premier plan par rapport à d'autres dans les images source ;
4. la robustesse même en présence d'information imprécise, incomplète ou inexistante.

Pour y arriver, il est nécessaire de choisir une technique qui, à partir de l'ensemble des triangles, soit avant tout capable d'éliminer les parties cachées. Plus précisément, cela veut dire que, pour deux points se projetant au même endroit sur la nouvelle vue, la technique doit être capable d'identifier lequel des deux est derrière l'autre et donc doit être ignoré. Dans le cas où l'information de profondeur des points est disponible partout dans les images, la technique de l'algorithme du Peintre [FvDFH91] est suffisante pour fournir les trois premières caractéristiques citées ci-dessus. Par contre, cette technique n'est pas capable de traiter des cas où l'information de profondeur est inexistante, imprécise ou seulement partielle. Pour ce faire, il est nécessaire d'ajouter quelques modifications à l'algorithme du Peintre comme propose [Lhu00].

Avant de passer à l'analyse de l'algorithme proposé par [Lhu00], il nous faut encore examiner de plus près comment l'algorithme du Peintre manipule l'ensemble des triangles appariés résultants de la TCV.

L'objectif de l'algorithme du Peintre est d'afficher tous les triangles par ordre de profondeur décroissante, tel un peintre dessinant l'arrière plan puis l'avant plan et les détails finals par dessus. Tous les triangles (sauf ceux en contact avec le bord des images) sont projetés afin de garantir l'absence de zones non renseignées. Comme les triangles non visibles des deux images contiennent les zones partiellement occultées par un objet plus proche de l'une des caméras, l'algorithme les affiche d'abord et ensuite il affiche les triangles visibles par dessus.

2.5.2 Calcul du rendu

La première étape de l'algorithme de rendu consiste à calculer la position des projections des sommets des triangles de la TCV dans la nouvelle image synthétisée. La position u'' dans l'image synthétisée se calcule par projection ou interpolation affine des sommets correspondants u et u' des triangles en fonction de la trajectoire désirée de la caméra virtuelle. Dans le cas d'un sommet d'occultation, u figure explicitement dans la TCV et u' est le point correspondant implicite.

Une fois les projections des sommets des triangles calculées sur l'image synthétisée, il faut passer à la projection des triangles (non visibles et visibles). Cela est fait séparément pour les deux images sources $I(u)$ et $I'(u')$ pour obtenir les images intermédiaires $\widetilde{I}(u'')$ et $\widetilde{I}'(u'')$.

L'algorithme commence par la projection des triangles non visibles de deux images originales (section précédente). Un poids w est associé à chaque triangle pour mesurer son degré de distorsion inverse. Le poids w est choisi proportionnel au rapport entre la surface du triangle dans la première image et celle du triangle dans la seconde. Les triangles non visibles contenant au moins un sommet d'occultation sont projetés d'abord, puis viennent tous les autres par ordre de distorsion inverse w croissante. Tous les triangles dont un des sommets est un des coins de l'image sont ignorés.

Ensuite, l'algorithme passe à la projection des triangles visibles. Pour chacun des triangles visibles, définis par ses sommets $v_0v_1v_2$ dans la première image et par $v'_0v'_1v'_2$ dans la seconde image, le rendu est effectué dans l'ordre croissant du maximum des normes de disparités $Max\|v_0 - v'_0\|, \|v_1 - v'_1\|, \|v_2 - v'_2\|$ ou dans l'ordre décroissant des profondeurs si celles-ci sont connues.

Finalement, pour obtenir l'image synthétique finale, il faut fusionner les deux images intermédiaires $\widetilde{I}(u'')$ et $\widetilde{I}'(u'')$ obtenues jusqu'à ce point par une interpolation des couleurs. La couleur finale de l'image synthétique $I''(u'')$ est donnée par :

$$I''(u'') = \frac{(1-\lambda)w(u'')\widetilde{I}(u'')+\lambda w'(u'')\widetilde{I}'(u'')}{(1-\lambda)w(u'')+\lambda w'(u'')}$$

où λ est un paramètre qui représente la position de l'observateur entre les deux images sources, tel que $I''(0) = I$ et $I''(1) = I'$.

Chapitre 3

L'Algorithme de Propagation

Sommaire

3.1 Introduction

Ce chapitre sera consacré à l'analyse plus détaillée de la phase de mise en correspondance de l'application de synthèse d'images qui a été le sujet du travail de thèse de [Lhu00]. Plus spécifiquement, c'est la technique d'appariement quasi-dense proposée, laquelle est basée sur un nouvel algorithme, qui nous intéresse. Un tel intérêt se justifie car c'est cet algorithme, appelé aussi algorithme de propagation, qui sera l'objet d'une étude de parallélisation dans notre travail.

Pour mieux comprendre les choix faits dans [Lhu00] , il faut rappeler que l'objectif visé est la synthèse d'images réelles. La principale caractéristique de ce genre d'images est que les *textures* sont plus réalistes car elles proviennent du monde réel. Ceci est d'autant plus vrai lorsqu'il s'agit de scènes d'extérieur, pour lesquelles des *textures* réalistes sont particulièrement difficiles à obtenir par une voie complètement synthétique.

Évidemment, synthétiser de telles scènes présentent des nombreuses difficultés. Celles-ci peuvent être par exemple des occultations résultantes du déplacement d'un piéton entre les deux images source, ce qui pose un vrai problème pour les méthodes de mise en correspondance. La difficulté croît encore lorsque les images source contiennent des structures fines et verticales comme des troncs ou des pylônes électriques, ce qui se produit très souvent dans les scènes urbaines et naturelles.

Pour traiter ces problèmes, le choix des méthodes de mise en correspondance dense (globales) s'avère limité puisqu'elles font intervenir une contrainte de *lissage*. Cette contrainte suppose que les distorsions entre les images sont faibles car elle empêche la sensibilité aux détails, caractéristique qui, dans les images réelles, est assez rare.

D'un autre côté, modéliser une scène seulement à partir de primitives éparses (points et arêtes) semble bien convenir pour l'objectif de synthèse, pourvu que celles existant dans les images source soient en nombre suffisant [ZFC99]. Par contre, les scènes ne peuvent pas comporter beaucoup de discontinuités de profondeur dans la distinction entre l'arrière et l'avant plan, ce qui n'est pas non plus le cas de figure des images réelles.

Les deux raisons présentées dans les paragraphes précédents justifient donc le choix de traiter le problème de la synthèse de vues réelles à partir d'une méthode quasi-dense (locale). L'inconvénient de cette méthode est justement l'absence de *lissage* global ce qui a pour conséquence le non appariement des zones qui n'ont pas suffisamment de *texture*. C'est l'étape suivante de triangulation qui traitera de ce problème.

Dans ce chapitre nous allons d'abord examiner en détail l'algorithme de propaga-

tion et les choix qui le concernent. Dans la suite on montre un ensemble démonstratif de résultats obtenus en terme de qualité des appariements gérés et du temps nécessaire pour les obtenir. Finalement, un bilan sur les avantages et inconvénients de l'algorithme proposé est présenté.

3.2 Algorithme de propagation

L'algorithme proposé par [Lhu00] dans son travail de thèse est basé sur une méthode classique de croissance de région pour la segmentation d'images [Mon87]. Cependant, au contraire de celui-ci qui utilise la homogénéité des pixels comme critère de croissance, la version proposée par[Lhu00] utilise une mesure similaire basée sur le score de corrélation des appariements qui a été présentée dans [LQ00b]. La donnée de départ de l'algorithme de propagation est obtenue dans une phase préliminaire à travers un appariement épars des points les plus fiables (points d'intérêt) détectés sur les images source. Ces appariements sont utilisés comme germes par un algorithme du type croissance de région pour propager les mises en correspondance vers les voisinages de ces points en utilisant des contraintes de disparité et une stratégie «meilleur d'abord». À la fin de l'algorithme, on obtient un ensemble d'appariements quasi-dense.

Dans cette section, on introduit d'abord la définition de ce qui dans ce travail sera considéré comme le voisinage d'un appariement. Ensuite, la propagation est discutée plus en détails, le choix de la mesure de confiance est expliqué, l'implantation de l'algorithme est présentée et finalement une comparaison avec d'autres méthodes existantes est faite.

3.2.1 Voisinage d'un appariement

La définition d'un voisinage de correspondances possibles pour un appariement est nécessaire pour limiter l'espace de recherche de l'algorithme autour des pixels à apparier. Les dimensions de ce voisinage doivent représenter un compromis entre la qualité des résultats et le temps d'exécution.

Le voisinage $\mathcal{V}(a, A)$ des pixels a et A est défini comme étant tous les pixels compris dans les fenêtres 5x5 dont les positions centrales sont ces deux points. Pour chaque pixel b dans $\mathcal{V}_5(a)$ dans la première image, une liste d'appariements candidats est construite. Cette liste est formée de tous les pixels compris dans une deuxième fenêtre 3x3 qui définit un voisinage autour de la position correspondante B dans $\mathcal{V}_5(A)$ dans la deuxième image comme cela est illustré dans la figure 3.1. La fenêtre 3x3 peut être éventuellement tronquée pour rester dans les limites de la fenêtre 5x5.

En d'autres termes, si on considère les positions des pixels, la recherche des candidats dans la fenêtre 3X3 établit que la différence entre la disparité $B - b$ d'un

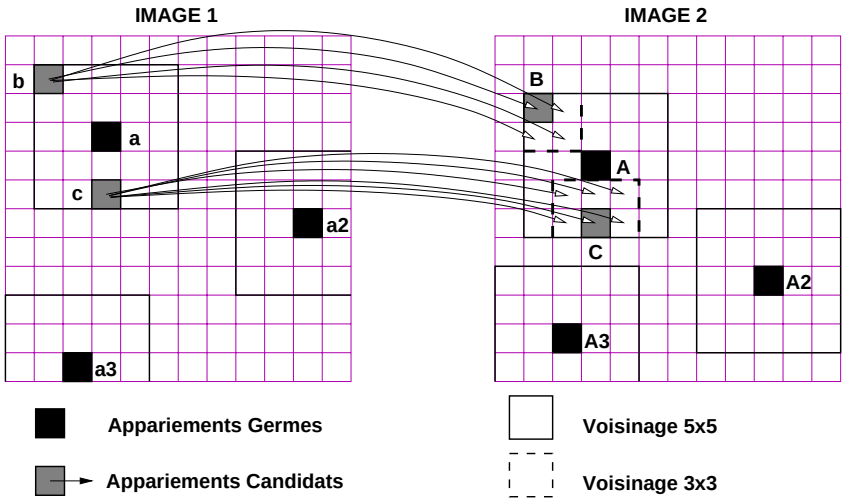


FIG. 3.1 – Voisinages et appariements.

appariement (B, b) dans $\mathcal{V}(a, A)$ et la disparité $A - a$ ne doit pas dépasser un pixel, c'est-à-dire $(B - b) - (A - a) \in \{-1, 0, 1\}^2$.

La définition complète du voisinage $\mathcal{V}(a, A)$ de l'appariement (a, A) est ainsi donnée par :

$$\mathcal{V}(a, A) = \{(b, B), b \in \mathcal{V}_5(a), B \in \mathcal{V}_5(A), ((B - b) - (A - a)) \in \{-1, 0, 1\}^2\}$$

3.2.2 Implantation

L'algorithme de croissance de régions proposé par [Lhu00] dans son travail sera présenté dans la suite. La donnée de l'algorithme est l'ensemble d'appariements obtenus à partir des points d'intérêt et qui sera appelé dorénavant **germes**. Il est implanté à l'aide d'un tas afin de sélectionner rapidement le meilleur germe pour la mesure ZNCC (cf. paragraphe 2.3.2, page 27) et pour permettre l'ajout d'autres germes à chaque itération. Le résultat est une carte des déplacements(**carte**) maintenue injective qui contient les bons appariements trouvés par l'algorithme de propagation. Une mesure de confiance $s(x)$ basée sur une différence de *luminance* est utilisée pour arrêter la propagation de l'appariement vers les zones a n'ayant pas assez d'information de façon à ne pas dépasser un seuil de luminance s_0 ($s(a) < s_0$). La définition complète de la mesure de confiance $s(x)$ est donnée par :

$$s(x) = max\{|I(x + \Delta) - I(x)|, \Delta \in \{(1, 0), (-1, 0), (0, 1), (0, -1)\}\}$$

où $I(x)$ est la *luminance* du pixel x et Δ représente un déplacement autour du pixel x.

Dans l'algorithme 1 on peut suivre l'évolution de la propagation pas à pas.

Algorithme 1 Algorithme de Propagation

1: $Carte \leftarrow \emptyset$
2: **Tant que** $Germes \neq \emptyset$ **Faire**
3: retirer la meilleure paire (a, A) de $Germes$
4: $Local \leftarrow \emptyset$
5: {stocker dans $Local$ les paires candidates}
6: **Pour tous** $(x, y) \in \mathcal{V}(a, A)$ **Faire**
7: **Si** $((x, *), (*, y) \notin Carte)$ **et** $(s(x) > t, s(y) > t)$ **et** $(ZNCC(c, d) > 0.5)$ **Alors**
8: $Local \leftarrow Local \cup \{(x, y)\}$
9: **Fin si**
10: **Fin pour**
11: {garder dans $Germes$ et $Carte$ les paires finales}
12: **Tant que** $Local \neq \emptyset$ **Faire**
13: retirer la meilleure paire (x, y) de $Local$
14: **Si** $(x, *), (*, y) \notin Carte$ **Alors**
15: $Carte \leftarrow (x, y),\ Germes \leftarrow (x, y)$
16: **Fin si**
17: **Fin tant que**
18: **Fin tant que**

Le principe de l'algorithme est de propager la mise en correspondance à partir des appariements germes. Ceux-ci sont des points de Harris améliorés [SMB98], appariés par corrélation de la *luminance* à leur voisinage selon la méthode de vérification par consistance croisée [Fua91].

La stratégie de l'algorithme de propagation proposé est justifiée par le fait que les appariements germes sont composés par des points d'intérêt, lesquels sont des maxima locaux de la texture des images sources. Ainsi, les voisins de ces appariements sont aussi fortement texturés, ce qui permet une très bonne

propagation même si ceux-ci ne sont pas non plus des maxima locaux.

Le score ZNCC est utilisé pour la sélection des germes et pour la propagation, car il est plus exigeant que les sommes de valeurs absolues ou carrées des différences de *luminances* dans les zones uniformes et plus tolérant dans les zones texturées où l'approximation du signal par l'échantillonnage est la plus mauvaise.

Tous les appariements germes sont des points de départ de propagations concurrentes. À chaque instant, l'appariement qui réalise le meilleur score ZNCC est retiré de l'ensemble des appariements germes courant. On l'utilise pour chercher de nouveaux appariements dans son voisinage. Ils sont ajoutés simultanément à l'ensemble des appariements germes courant et à l'ensemble des appariements finalement acceptés, à moins que la variation de la *luminance* y soit établie trop faible par une mesure de confiance.

La contrainte d'unicité de l'appariement et l'arrêt de l'algorithme sont garantis parce que l'on choisit les nouveaux appariements parmi ceux qui n'ont pas de pixels figurant dans des appariements déjà acceptés.

3.2.3 Stratégie «meilleur d'abord»

L'application de la stratégie «meilleur d'abord» pour retirer les appariements de l'ensemble d'appariements germes apporte des avantages dans le traitement de deux problèmes importants de la mise en correspondance : les zones partiellement occultés et les faux germes.

L'usage simultanée d'une stratégie «meilleur d'abord» globale et de la contrainte d'unicité permet de traiter le cas des zones partiellement occultées. Grâce à elle, les appariements de pixels réalisant de bons scores de corrélation sont acceptés avant que la majorité des appariements potentiels à mauvais scores soient tentés. En particulier, les objets à l'avant plan et à l'arrière plan sont appariés (avec un bon score en général) avant que des appariements dans les zones partiellement occultées correspondantes soient tentés (avec un mauvais score en général).

Une fois que l'avant plan et l'arrière plan sont appariés, la propagation ne peut plus s'étendre dans les zones partiellement occultées correspondantes puisqu'elle est arrêtée dans l'autre image grâce à la contrainte d'unicité : le contour de la zone partiellement occultée correspond dans l'autre image à un contour entourant une zone de surface nulle, entourée par des zones déjà appariées.

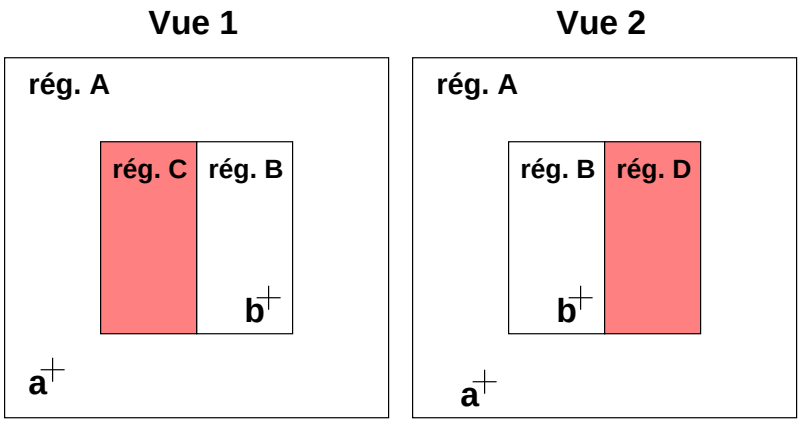


FIG. 3.2 – Stratégie «meilleur d'abord» et le problème des zones partiellement occultées.

Dans la figure 3.2 on peut voir un exemple du traitement du problème des zones partiellement occultées par la stratégie «meilleur d'abord». On y voit deux vues d'une scène avec un arrière plan A, un avant plan B et des zones partiellement occultées C et D. Si les appariements corrects de A et B ont des scores de corrélation meilleurs que les mauvais appariements formés avec des pixels de C et D, grâce à l'ordre «meilleur d'abord», les zones A et B sont remplies et appariées par la propagation à partir des germes a et b avant que la propagation ne tente de progresser dans les zones C et D. Une fois que A et B sont appariées, la propagation est arrêtée par la contrainte d'unicité à la frontière de C dans l'image 1 (resp. D dans l'image 2) parce que la frontière correspondante dans l'image 2 (resp. 1) correspond à une zone C de surface nulle, dont la position est déjà occupée par d'autres zones déjà appariées. Si la stratégie «meilleur d'abord» n'était pas employée, rien n'empêcherait l'appariement de zones C et D d'être produit avant celui des zones A et B car des faux germes situés sur C et D pourraient déclencher une propagation avant les bons germes situés sur A et B.

En ce qui concerne la robustesse aux faux germes, le risque de fausse propagation est fortement réduit par la stratégie "meilleur d'abord" appliquée à l'ensemble des appariements germes. À la différence d'autres méthodes similaires existantes pour l'appariement de points d'intérêt à l'aide de corrélation, l'algorithme de propagation proposé a seulement besoin des appariements les plus fiables plutôt que de prendre un maximum d'entre eux. Cette caractéristique rend l'algorithme moins vulnérable à la présence de mauvais appariements dans l'ensemble des germes. Dans certains cas extrêmes, un seul bon germe suffit pour provoquer une avalanche de bons appariements dans les parties texturées des images. Dans un tel scénario, les mauvais germes ont peu de chance de se développer.

3.2.4 Qualité d'un appariement

La qualité des appariements obtenus par l'algorithme de propagation peut être analysée sous deux aspects : le nombre de paires de points correctement appariés et le résultat visuel sur la surface des images. Le premier permet de connaître le pourcentage de la surface des images qui a été appariée. Le second offre un moyen de comparer visuellement les zones appariées sur les images source.

Dans la suite de ce rapport, la qualité des résultats sous ses deux formes sera un important élément de comparaison entre les appariements obtenus par le programme séquentiel et par ses versions parallèles. Cette comparaison est nécessaire car les modifications qui seront faites sur l'algorithme séquentiel pour le rendre parallèle peuvent conduire à un comportement différent du même à cause des possibles changements dans l'ordre «meilleur d'abord». Cela produira forcément des effets sur la qualité du résultat final.

3.3 Illustration

Dans cette section, nous allons présenter des résultats obtenus à partir de l'exécution séquentielle de l'application complète développée par [Lhu00]. Notre objectif est d'offrir au lecteur un échantillon des performances de l'application séquentielle pour permettre une comparaison dans la suite aux résultats des versions parallèles qui seront proposées.

Pour chaque exemple, nous allons proposer des informations sur la qualité des appariements obtenus et les temps d'exécution. En ce que concerne l'évaluation de la qualité du résultat, nous allons présenter l'étendue de l'appariement obtenu sur les images après l'exécution de la phase de propagation ainsi que le nombre d'appariements trouvés. Pour les temps d'exécution, en plus du temps utilisé pendant la phase de propagation, nous montrons aussi les temps de calcul de toutes les autres phases de l'application.

Les résultats présentés dans la suite ont été calculés sur un Pentium III 733 Mhz sur Linux. Chaque valeur incluse dans un des tableaux suivants est le résultat d'une moyenne arithmétique simple de dix exécutions en éliminant les valeurs extrêmes. Les figures qui montrent l'étendue de la propagation sur les images exemples sont le résultat d'une seule exécution. Cela n'est pas un problème puisque le calcul séquentiel des appariements est déterministe. Les petites différences dans les temps des dix exécutions sont dues aux incertitudes apportées par le système d'exploitation.

3.3.1 Les paires d'images tests

Avant de passer à l'analyse des résultats, il est nécessaire de connaître les caractéristiques des paires d'images qui seront traitées. Nous avons choisi quatre paires dont les caractéristiques distinctes permettent d'évaluer les différentes capacités et limitations de l'application séquentielle.

Ces quatre paires d'images seront les mêmes utilisées pour l'évaluation des versions parallèles de l'algorithme de propagation que nous proposerons dans ce rapport. À partir d'ici nous les référencerons à travers des noms qui représentent le sujet principal des vues, à savoir : **fleur**, **maison**, **tronc**, et **rivulet**.

Fleur

Les vues montrées dans la figure 3.3 sont la seule paire de notre ensemble d'exemples qui représentent des images d'une scène réelle qui **n'est pas un paysage d'extérieur**. Il s'agit plutôt d'une macrophotographie d'une fleur qui présente des conditions de lumière très particulières. Les images sont en couleurs et sont de dimension de 368x384, ce qui fait 141 312 pixels.

(a) Vue 1 (b) Vue 2

FIG. 3.3 – La paire d'images **Fleur**

Dans cette paire, on peut constater qu'il n'y a pas beaucoup de zones non texturées sur les images (dans ce cas, les zones noires), que le mouvement de la caméra est très léger et qu'il n'y a ni occlusions, ni objets fins (peu épais).

Avec de telles caractéristiques, l'algorithme de propagation ne doit pas rencontrer trop de difficultés pour réaliser un appariement couvrant presque la totalité de la surface des images. Ainsi, on dispose d'un exemple qui nous permettra d'évaluer l'évolution en termes de temps et qualité de l'appariement final obtenu par la propagation dans des circonstances favorables sur une paire d'images plutôt réduite.

Maison

Pour cette deuxième paire (figure 3.4), on passe à une scène réelle d'**extérieur**. Les images sont toujours en couleurs et les dimensions sont de 768x512, ce qui fait 393 216 pixels.

(a) Vue 1 (b) Vue 2

FIG. 3.4 – La paire d'images **Maison**

Cette paire présente toutes les difficultés d'une scène réelle d'extérieur : des zones non texturées (ciel, pelouse et montagnes en arrière plan), des zones texturées à répétition (la végétation surtout au premier plan), des objets fins (poteau derrière la maison) et une petite zone d'occlusion grâce à l'arbre en face de la maison. De plus, le déplacement de la caméra lors des prises de vue est assez conséquent.

Toutes les caractéristiques énumérées ci-dessus font de cette paire un test redoutable pour l'algorithme de propagation. De plus, avec presque trois fois plus de pixels à apparier, elle nous offre des temps d'exécution plus importants qui permettront de mieux évaluer le gain apporté par la parallélisation.

Rivulet

Les images de la troisième paire 3.5 sont aussi en couleurs et montrent elles aussi une scène réelle d'extérieur. Les images ont les mêmes dimensions que la paire

précédente, c'est-à-dire 393 216 pixels (512x768), et la disposition de la scène est à la verticale.

(a) Vue 1 (b) Vue 2

FIG. 3.5 – La paire d'images **Rivulet**

Cet exemple présente des images presque sans zones non texturées et, en plus, ses zones texturées sont assez différenciées. Un faible déplacement latéral de la caméra découvre d'importantes zones cachées par la géométrie de la scène.

Dans cet exemple, l'algorithme de propagation pourra s'étendre sur une grande partie de la surface des images grâce à la présence d'un très grand nombre d'appariements germes. Et, le plus intéressant, il sera possible de vérifier le comportement de l'algorithme face à de grandes zones occultées.

Tronc

La quatrième paire testée (figure 3.6) est l'unique exemple d'images présentées en tons de gris. Il s'agit de scènes réelles d'extérieur. Les dimensions sont de

360x240, c'est-à-dire 86 400 pixels.

Dans cet exemple, on retrouve des caractéristiques très difficiles à traiter pour la propagation. Les images en tons de gris offrent moins de marge pour différencier les points à partir de leur couleur. Le grand déplacement de la caméra produit une transformation assez importante de la perspective des scènes. Le ciel sans *texture* occupe une partie significative de la surface des scènes et n'offre pas d'appariement germes pour la propagation. Les zones texturées à répétition comme le parterre au premier plan occupent elles aussi beaucoup de place dans les scènes et sont sources de problèmes pour la propagation puisque l'algorithme a du mal à progresser à cause de la similitude entre les voisinages. Le tronc au premier plan cache des zones considérables derrière lui et qui produit des zones partiellement occultées dans ces frontières.

(a) Vue 1 (b) Vue 2

FIG. 3.6 – La paire d'images **Tronc**

Tous ces éléments agrégés représentent un défi difficile pour la version séquentielle de la propagation. La zone appariée sera plus petite que dans les autres exemples et cela ajouté aux dimensions réduites des images pourra donner en un temps d'exécution très bref et par conséquence un obstacle dur pour les versions parallèles.

3.3.2 La qualité des appariements

Nous allons présenter maintenant la qualité des résultats obtenus sur les quatre paires d'images sources. D'abord, on montre la qualité visuelle des appariements, pour ensuite quantifier ces résultats par l'intermédiaire de l'analyse du nombre de paires de points correctement appariés.

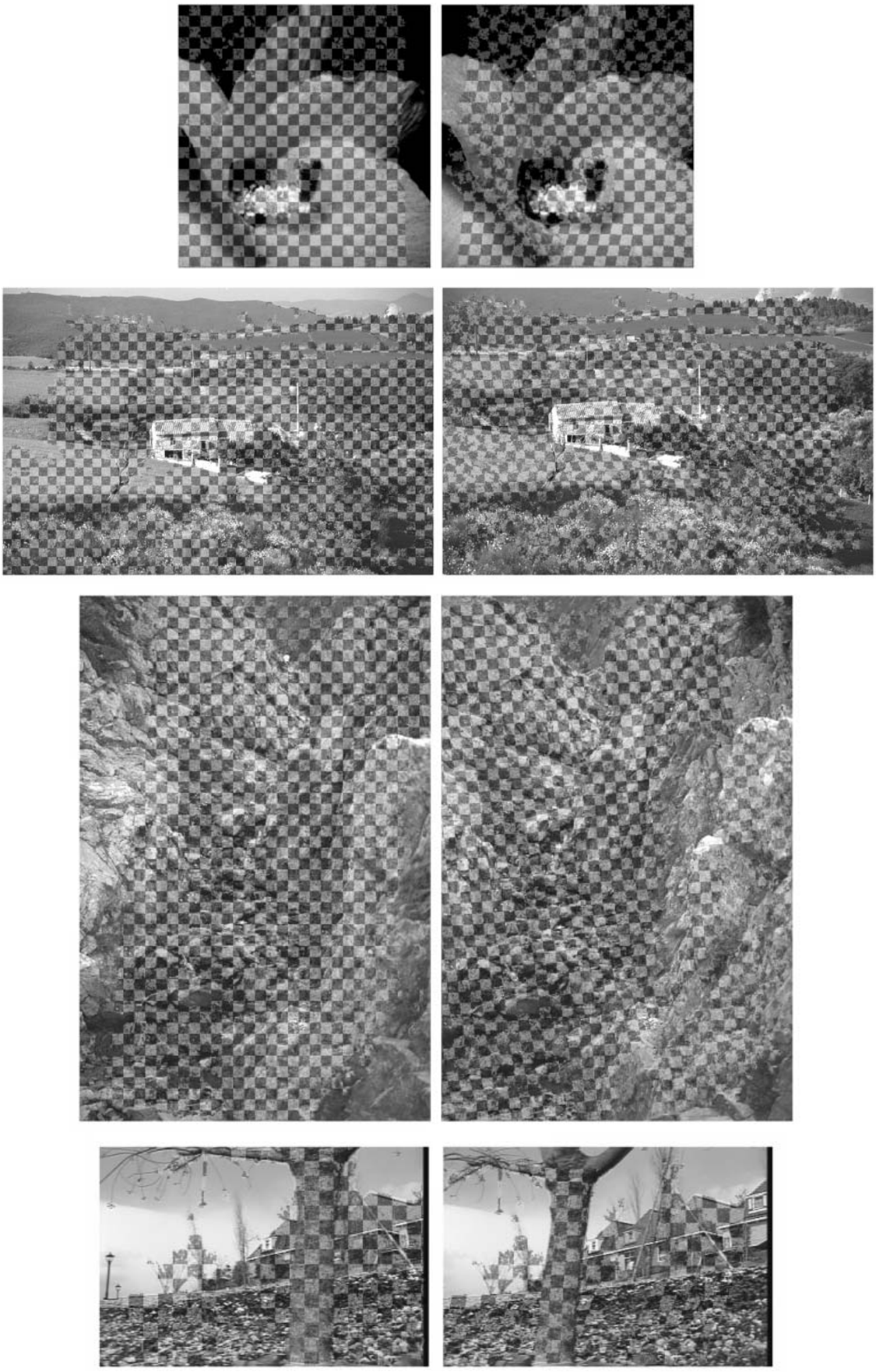


FIG. 3.7 – La propagation (zone quadrillée) sur les 4 paires d'images exemples

Dans la figure 3.7 on peut visualiser l'étendue de la propagation sur la surface des quatre paires d'images proposées. Cela permet d'analyser le bon (ou mauvais) comportement de la propagation dans les régions plus difficiles à traiter comme les zones occultées, les objets fins, les zones peu texturées, etc.

TAB. 3.1 – Les quantités d'éléments trouvés pour chaque phase de l'application sur tous les exemples.

Phase	Fleur	Maison	Rivulet	Tronc
Pts. d'intérêt (vue 1/ vue 2)	205 / 246	1835 / 2025	2851 / 2877	340 / 381
Appariement épars (paires)	114	280	325	123
Propagation (paires)	**111 538**	**230 399**	**238 746**	**30 047**
Triangulation (triangles)	26	68	88	70

Le tableau 3.1 présente les quantités de résultats de chaque phase de l'application de synthèse d'images pour les quatre exemples.

Sur la paire **Fleur**, on retrouve 81,7 % de la surface des images appariées, chiffre attendu car cette paire d'images était censée ne pas offrir trop de difficultés à l'évolution de la propagation.

Pour la paire **Maison**, 58,6 % des pixels des surfaces des images ont été correctement appariés. Ce chiffre est assez bon si on considère le mouvement de la caméra beaucoup plus prononcé que dans l'exemple **Fleur**. De plus, l'algorithme a correctement traité l'objet fin existant dans les images (le poteau blanc juste à côté de la maison) ainsi que la zone d'occlusion cachée par l'arbre en face de la maison. Les difficultés sur les zones peu texturées ou avec des *textures* répétitives sont elles aussi confirmées.

La paire **Rivulet** présente 60,7 % de la surface de ces images appariées. Un résultat sans surprise, car ses caractéristiques sont assez proches de l'exemple **Maison**. L'intérêt majeur de cet exemple est la plus grande surface des zones occultées lesquelles n'ont pas été correctement appariées.

Finalement, la quatrième paire **Tronc** montre tout le mal que la propagation a eu à cause des difficultés présentes avec seulement 34,8 % de surface appariée.

3.3.3 Le temps d'exécution

Les temps d'exécution des exemples sont une conséquence directe de la taille et des difficultés présentes sur les images adoptées. Dans le tableau 3.2, on peut voir les temps de calcul nécessaires pour chacune des phases de l'algorithme global de l'application de synthèse d'images. La propagation est, de loin, la phase qui

consomme le plus de temps.

TAB. 3.2 – Les temps d'exécution pour chaque phase de l'application sur tous les exemples.

Phase	Fleur	Maison	Rivulet	Tronc
Points d'intérêt	0.46s	2.24s	2.26s	0.29s
Appariement épars	0.09s	0.60s	1.21s	0.12s
Propagation	**5.11s**	**19.75s**	**17.44s**	**3.14s**
Triangulation	1.64s	3.30s	4.65s	1.15s

3.4 Propagation : bilan final

Dans ce chapitre, nous avons voulu faire une présentation de l'algorithme de propagation (appariement quasi-dense) proposé dans la thèse de [Lhu00]. Des arguments en faveur de cette nouvelle méthode d'appariement ont été développés pour justifier son utilisation dans une application de synthèse d'images à partir de vues réelles. Une analyse plus détaillée de l'algorithme a également été faite.

On a vu que les méthodes éparses ne produisent pas toujours suffisamment d'information, principalement par rapport aux contours d'occultation. D'un autre côté, on a vu que les méthodes denses globales ont des problèmes de convergence lorsque elles sont mal initialisées et que, de plus, la présence d'une contrainte de *lissage* rend difficile le traitement des *zones occultées* et des régions avec trop de détails sur les images. La propagation a été proposée pour remédier à ces problèmes. Celle-ci a été conçue pour profiter de tous les travaux sur la mise en correspondance éparse comme forme d'initialisation. Elle se distingue des autres méthodes de propagation existantes par sa capacité à pouvoir traiter le problème des occultations.

Dans cette section, nous allons énumérer les avantages et les inconvénients de la méthode de propagation étudiée. En plus, quelques observations concernant la suite de notre travail sont faites.

3.4.1 Avantages

La propagation croît progressivement dans les deux images à partir des mises en correspondance éparses les plus fiables, suivant une stratégie «meilleur d'abord». Voici ses avantages :

- **Traitement des occultations et des disparités importantes**. Dans le cas des *scènes rigides*, il est possible de traiter les grands intervalles de disparité

entre les images source. L'algorithme permet aussi de mettre en correspondance les structures fines comme les pylônes électriques et les troncs d'arbres. L'utilisation d'une contrainte d'unicité dès la phase de propagation permet aussi de réduire grandement les mises en correspondance erronées sur les zones partiellement occultées ;
- **Traitement des artefacts de mise en correspondance**. L'utilisation de fenêtres de petite taille lors de la propagation (5x5) limite les artefacts de mise en correspondance (celui de la ligne brisée, par exemple) aux voisinages des contours d'occultation ;
- **Robustesse aux faux germes**. La propagation est robuste vis à vis des faux germes grâce à la stratégie "meilleur d'abord" et grâce au fait qu'elle n'apparie que dans les zones où il y a beaucoup d'information (caractéristique classique des méthodes locales).

3.4.2 Inconvénients

Les inconvénients de la propagation sont ceux des méthodes locales, c'est-à-dire les problèmes d'ambiguïtés liés à l'absence d'une contrainte de *lissage* globale. Plus précisément :

- **Traitement des zones uniformes**. Les zones des images avec peu d'informations (un ciel trop homogène, par exemple) ne peuvent pas être appariées ;
- **Traitement des *textures* répétitives**. Les images à *textures* trop répétitives peuvent poser des problèmes : si un appariement germe est décalé d'une période, alors la propagation résultante l'est aussi. Ce phénomène est limité si on applique la *contrainte épipolaire* ou si au voisinage de la *texture* répétitive il existe une zone plus texturée et non répétitive : grâce à l'ordre "meilleur d'abord" global, l'appariement se propage de façon continue (donc sans décaler d'une période) de la zone la plus texturée à la moins texturée.

Deuxième partie

Contexte du travail

Chapitre 4

Calcul Parallèle

Sommaire

4.1 Introduction

Le calcul parallèle est devenu une technologie clé dans l'augmentation de la capacité d'utilisation des ordinateurs modernes. Cela s'est produit grâce à une demande de plus en plus forte pour des hautes performances obtenues à bas coût dans les applications du monde réel.

Avec l'utilisation du calcul parallèle (ou tout simplement parallélisme) les systèmes informatiques modernes sont capables d'attaquer des problèmes qui exigent une quantité de calcul auparavant insurmontable par le calcul séquentiel traditionnel. De plus, le parallélisme est aussi un outil important lorsqu'on traite des problèmes qui nécessitent naturellement de faire coopérer plusieurs ordinateurs distants.

D'un autre côté, il n'est pas si simple d'employer le calcul parallèle car il s'agit d'un domaine qui peut être envisagé pour résoudre un très large éventail de problèmes avec des caractéristiques assez différentes. De ce fait, il existe un grand nombre de formes d'expression du parallélisme, chacune adaptée à certaines classes de problèmes et avec ses propres particularités. Il arrive aussi qu'un même problème puisse être traité par plus d'une approche de parallélisme avec des reflets directs sur les performances obtenues. Cela a pour conséquence une gamme des possibilités plus vaste que celle offerte par le calcul séquentiel et ne facilite guère le choix de la technique la plus adéquate au problème que l'on veut traiter.

L'utilisation du parallélisme est d'autant plus compliquée car dans la pratique elle ne dépend pas uniquement des caractéristiques du problème mais aussi de celles de l'architecture des machines à disposition. Et, là encore, on dispose d'une multitude de possibilités qui vont des machines qui offrent le parallélisme au niveau d'instructions jusqu'à la programmation de systèmes distribués où plusieurs ordinateurs distants (qui peuvent, eux, avoir plusieurs processeurs) sont reliés par un réseau.

Tout cela rend difficile la tâche de ceux qui veulent proposer un tour d'horizon sur les différentes formes de parallélisme sur un nombre réduit de pages. Nous avons choisi de présenter dans ce chapitre un sous-ensemble des possibilités du calcul parallèle orienté par nos besoins, ainsi que les concepts qui nous seront utiles.

Le chapitre s'organise ainsi : dans la première section nous faisons un rapide tour d'horizon des types d'architectures pour le parallélisme. Ensuite, les modèles de programmation parallèle sont présentés. Dans la troisième section, nous énumérons certains critères d'évaluation de programmes parallèles qui seront employés dans ce travail. Et, finalement, on aborde les facteurs qui déterminent les performances d'un programme parallèle.

4.2 Architectures pour le parallélisme

Historiquement, les machines parallèles ont été classifiées en fonction de la multiplicité des flots d'instructions et de données. Ainsi, dans [Fly72] on retrouve, parmi d'autres, les deux principaux types de machines parallèles :

- les machines SIMD (*Single Instruction Multiple Data*) où les processeurs exécutent la même opération simultanément sur des données différentes ;
- les machines MIMD (*Multiple Instruction Multiple Data*) où les nœuds de calcul exécutent globalement le même travail mais pas forcément la même opération au même moment.

Plus simple à utiliser, les machines SIMD ne permettent cependant pas de traiter efficacement tous les types de problèmes. De ce point de vue, les machines MIMD sont d'un usage plus général. C'est principalement pour cette raison (ainsi que pour leur coût moins élevé) que la plupart des ordinateurs parallèles d'aujourd'hui sont des machines MIMD.

À cause du grand nombre de machines MIMD existantes et des critères trop généraux de la classification de [Fly72], il est courant actuellement [Tan92] [vdSD96] de subdiviser les machines MIMD en deux groupes : les multiordinateurs et les multiprocesseurs. Le type d'interconnexion utilisée pour relier mémoires et processeurs, bus ou commutateur, génère d'autres sous-divisions pour chacun de ces groupes.

Les multiprocesseurs, aussi appelés systèmes fortement couplés, ou encore machines à **mémoire partagée**, sont caractérisés par plusieurs processeurs partageant un même espace d'adressage. Dans le groupe des multiordinateurs, ou systèmes faiblement couplés, ou encore machines à **mémoire distribuée**, les nœuds qui définissent la machine parallèle sont indépendants les uns des autres, c'est-à-dire chaque nœud est composé d'un processeur et d'une mémoire locale. Finalement, il existe aussi des machines à architecture **hybride** qui sont des multiordinateurs de multiprocesseurs.

4.2.1 Mémoire partagée

Les architectures parallèles à mémoire partagée présentent une mémoire contiguë unique accessible directement par leur ensemble de processeurs (figure 4.1). Actuellement, la plupart des machines à mémoire partagée sont des systèmes symétriques (SMP - *Symetric MultiProcessing*) où tous les processeurs ont les mêmes fonctions et se disputent les ressources du système de façon uniforme.

L'existence d'une mémoire commune entre les processeurs simplifie le travail de parallélisation des algorithmes. La coopération entre les processeurs est plus facile

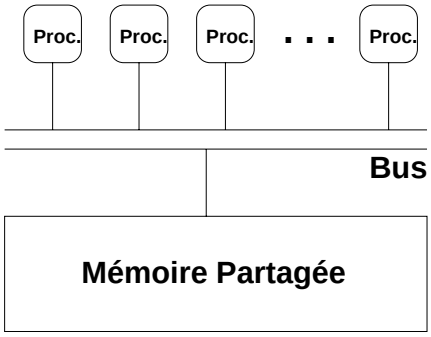


FIG. 4.1 – Architecture à mémoire partagée.

car les communications sont faites par le biais de lectures et écritures de zones de mémoire partagées, ce qui rapproche la logique de construction de programmes de celle de la programmation séquentielle traditionnelle. D'un autre côté, l'existence d'une mémoire commune est aussi le point faible des ces architectures car elle oblige les processeurs à se disputer pour y accéder.

Une solution proposée pour limiter le problème du goulot d'étranglement dans l'accès à la mémoire partagée est l'utilisation de plusieurs niveaux de mémoires cache. Cela peut apporter une réduction du temps réel d'accès à une donnée en fonction de sa localisation dans le système de mémoires, mais rajoute une complexité en plus qui est le maintien de la cohérence de cache. L'emploi de ces techniques n'empêche néanmoins pas que les architectures SMP d'aujourd'hui restent limitées à quelques dizaines de processeurs au maximum [Ber97]. Par conséquent, ce sont des architectures qui ne sont pas adaptées au parallélisme à très grande échelle.

4.2.2 Mémoire distribuée

Une architecture à mémoire distribuée est composée à la base par un ensemble de nœuds connectés entre eux par un réseau de communication (figure 4.2). Chaque nœud est composé d'un processeur et d'une mémoire locale et toutes les interactions entre eux doivent passer obligatoirement par le réseau. De telles architectures peuvent atteindre une taille beaucoup plus importante en terme de nombre de processeurs et de capacité mémoire car elles n'imposent pas de contention entre les processeurs pour accéder à la mémoire, la seule contrainte à l'extension étant la capacité du réseau de communication. Ainsi, des machines composées de plus d'une dizaine de milliers de processeurs ont vu le jour. Ce sont les architectures massivement parallèles (MPP - *Massively Parallel Processing*).

Les MPP peuvent se présenter sous la forme d'une énorme variété de configurations dépendant de la technologie d'interconnexion utilisée entre les nœuds et de la

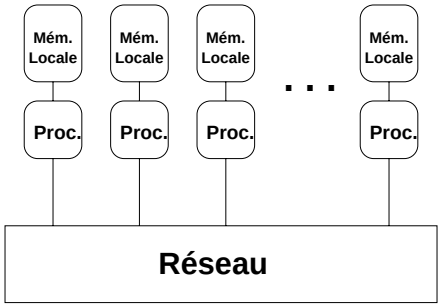


FIG. 4.2 – Architecture à mémoire distribuée.

topologie adoptée. L'évolution de ces architectures a été fortement influencée par les avancées des technologies d'interconnexion des nœuds. Aujourd'hui, certaines configurations proposées pour les machines à mémoire distribuée ressemblent à des réseaux d'ordinateurs. Un bon exemple de cela sont les grappes, devenues très répandues. Elles sont composées d'éléments produits en grande série comme des processeurs et mémoires utilisées pour la fabrication de micro-ordinateurs.

L'avantage évident des architectures à mémoire distribuée est leur adaptation à un parallélisme à grande échelle au contraire des architectures à mémoire partagée. Par contre, la mise en œuvre des applications parallèles sur de telles machines est moins simple. En règle générale, la possibilité d'accès à une mémoire non locale n'est pas prévue. De plus, le coût de communication entre les processeurs est plus élevé par rapport aux architectures à mémoire partagée. C'est le programmeur qui doit prendre en compte la localisation de données sur les mémoires locales et les distribuer selon ses besoins entre les nœuds tout en essayant de minimiser le nombre de communications.

4.2.3 Architecture Hybride

Plus récemment, un nouveau concept d'architecture parallèle est apparu. Son originalité réside dans la combinaison des deux types d'architectures présentées précédemment. Ce sont des machines composées d'un ensemble de nœuds multi-processeurs reliés entre eux par un réseau à haut débit. Avec cette configuration on retrouve la mémoire partagée au sein de chaque nœud et la mémoire distribuée entre l'ensemble des nœuds.

Les architectures hybrides essayent donc de profiter de la facilité d'accès aux données offerte par les mémoires partagées tout en gardant l'extensibilité des machines à mémoire distribuée. En pratique, une telle configuration peut être obtenue, par exemple, à travers une grappe de machines SMP (avec des mécanismes

de gestion supplémentaires).

Actuellement, les plus grandes machines construites utilisant la philosophie hybride possèdent des dizaines de milliers de processeurs. Cependant, profiter de toute cette puissance de calcul n'est pas évident. Le développement d'applications parallèles sur les machines hybrides présente de nouvelles difficultés, la plus délicate étant la gestion des différents niveaux de mémoire. Les modèles de programmation parallèle classiques n'ont pas été conçus pour un tel type d'organisation de mémoire. La mise en œuvre d'applications parallèles pour ces machines n'est donc pas toujours bien maîtrisée.

4.3 Modèles de programmation parallèle

Un modèle de programmation parallèle est un moyen d'établir des liens entre le développement d'une application sur un type d'architecture précis tout en faisant abstraction des détails techniques qui peuvent varier. Pour être utile, un modèle de programmation doit satisfaire un certain nombre de propriétés [Ski95] telles que :

- méthode précise de développement ;
- simplicité de compréhension ;
- indépendance de l'architecture ;
- implantation efficace possible ;
- moyens d'estimation du coût du modèle (rapport entre les performances parallèles et les ressources nécessaires à son exécution).

Les modèles parallèles peuvent être classés selon différents niveaux d'explicitation du parallélisme. On peut isoler six catégories de modèles [ST98] :

- **Modèle à parallélisme implicite** - toutes les activités liées à l'exécution parallèle sont complètement cachées au programmeur. La parallélisation est obtenue par la transformation du code séquentiel en code parallèle par des compilateurs parallélisateurs. Exemple : PROLOG Parallèle [FD90] ;
- **Modèle à parallélisme explicite** - le parallélisme est décrit de façon explicite sans pourtant définir la façon dont l'application est divisée en tâches, ni le placement (ordonnancement) (cf. paragraphes 4.5.3 et 4.5.2) de celles ci ni leurs communications. Exemple : Multilisp [Hal86] ;
- **Modèle à décomposition explicite** - les langages offrent des primitives explicites pour définir les tâches à exécuter en parallèle. Cependant, la réalisation du placement (ordonnancement) et des communications reste toujours cachée au programmeur. Exemple : Athapascan-1 [GRCD98] et Cilk [BJK+95] ;
- **Modèle à placement explicite** - les programmeurs prennent en charge la décomposition en différentes tâches et réalisent leur placement sur les proces-

seurs. Par contre, communications et synchronisation sont assurées de façon transparente. Exemple : Linda [ACGK88] ;
- **Modèle guidé par événements** - les mouvements de donnés et les synchronisations ne peuvent être faites que *via* la production d'un événement préétabli lequel déclenche des traitements. Exemple : technique basée sur les *actors* [Agh90] et [Agh85] ;
- **Modèle à parallélisme totalement explicite** - tous les aspects de décomposition, de placement, de communication et de synchronisation sont à la charge du programmeur de l'application parallèle. La programmation parallèle basée sur ce modèle est difficile car tous les détails de la parallélisation sont apparents.

La parallélisation implicite à partir d'un programme séquentiel demande peu de travail en comparaison d'une parallélisation totalement explicite. Par contre son utilisation en dehors du cadre des architectures à mémoire partagée est souvent difficile et limitée à des résultats décevants. Pour la plupart des problèmes (surtout dans le cadre des architectures à mémoire distribuée), la parallélisation totalement explicite est la seule envisageable pour obtenir de bonnes performances. Cela se produit car souvent la résolution parallèle d'un problème nécessite une vision globale de celui-ci et les outils automatiques utilisés par les autres modèles ne peuvent en avoir qu'une vision locale liée à un algorithme séquentiel particulier. Par l'intermédiaire du modèle totalement explicite le programmeur est libre pour construire un algorithme parallèle spécifique parfois très différent de l'algorithme séquentiel. Dans le cadre de ce travail, nous allons nous focaliser sur le modèle à parallélisme totalement explicite.

Un deuxième critère pour classer les modèles de parallélisme est la source du parallélisme. Dans ce cas, on distingue deux groupes de modèles : le **parallélisme de données** et le **parallélisme de contrôle**. Le premier groupe étant approprié à la programmation de machines SIMD tandis que le second, plus générique, s'adapte aux machines MIMD. Une présentation des modèles appartenant à ces groupes est faite dans la suite.

4.3.1 Parallélisme de données

Le parallélisme de données est considéré comme étant la forme la plus directe de parallélisme. Il doit être utilisé lorsque une même opération peut être appliquée en même temps à plusieurs (voir tous) éléments d'un ensemble de données. Dans ce genre de parallélisme, les unités de traitement de données sont dupliquées et une seule unité de contrôle les commande. Cette source de parallélisme est fréquemment utilisée lorsque l'on effectue des opérations sur des vecteurs (ce qui se produit très souvent dans le calcul scientifique). On peut citer à titre d'exemple les problèmes d'algèbre linéaire dense.

Dans ce mode d'expression du parallélisme, la charge de travail de chaque processeur est définie par la distribution des données et les communications sont utilisées dans la redistribution des données sur l'ensemble des processeurs. Schématiquement, le programme parallèle est une succession de phases de calculs et de phases de communications pour la redistribution des données.

Deux modèles sont les plus répandus dans l'exploitation du parallélisme de données : le modèle vectoriel et le modèle pipeline.

Modèle Vectoriel

Dans la programmation vectorielle, on cherche à identifier les opérations simples et indépendantes sur un ensemble de données organisées (un vecteur typiquement) pour les effectuer simultanément. Un exemple serait l'exécution d'une opération d'addition sur deux vecteurs avec n éléments sur n unités de calculs indépendantes. Le temps d'exécution d'une telle opération serait le même qu'une addition de deux données simples.

Le modèle vectoriel vu ainsi peut être considéré comme une forme de parallélisme parfait. Cependant, il est difficile qu'il apparaisse de façon aussi pure. Ainsi, dans le cas d'un produit scalaire par exemple, les produits des coordonnées deux à deux font apparaître un parallélisme vectoriel, mais l'obtention du résultat final est faite par la fusion des résultats intermédiaires.

La programmation vectorielle est très répandue et certains langages comme Fortran disposent d'instructions spécifiques pour l'exprimer.

Modèle Pipeline

Le principe du modèle pipeline est basé sur le découpage d'une opération s'effectuant en temps borné en sous-opérations de même durée. Cela permet la création d'une chaîne d'exécution de façon à ce que n sous-opérations sur n données différentes puissent être réalisées en parallèle. Un exemple didactique serait une ligne de n pompiers qui se transmettent des seaux jusqu'au dernier qui jette l'eau contenu dans le seau sur le feu.

Au début, il y a une phase de remplissage du pipeline, c'est-à-dire le premier seau avance dans la ligne suivi du deuxième, du troisième jusqu'à ce que le dernier pompier reçoive le premier seau. À ce moment le pipeline est plein, chaque pompier a un seau et peut le passer au pompier suivant. Le temps total pour le calcul de k opérations (k seaux à transporter), en supposant une durée unitaire de toutes les opérations, serait $n + k$ au lieu de $k * n$ en calcul séquentiel. Le pipeline serait d'autant plus efficace que k (le nombre de seaux à transporter) serait grand devant

n (le nombre de pompiers).

Le modèle pipeline est très utile car il permet aux architectures qui l'emploient d'intégrer efficacement certaines dépendances entre les données très difficiles à traiter en parallèle (comme un produit scalaire par exemple).

4.3.2 Parallélisme de contrôle

Le parallélisme de contrôle consiste à décrire un algorithme parallèle sous la forme d'un graphe orienté sans cycle. Les nœuds du graphe sont des suites d'opérations élémentaires exécutées en séquentiel, ils sont les tâches du graphe. Les arcs du graphe indiquent les contraintes de précédence entre les tâches, c'est-à-dire la permission d'exécution d'une tâche qui dépend d'une donnée déterminée seulement à partir du moment où sa(ses) tâche(s) précédente(s) a(ont) terminé son(leur) calcul sur cette même donnée.

Un tel graphe, appelé graphe de précédence ou graphe de tâches (figure 4.3), définit un ordre partiel sur les tâches. Les tâches non ordonnées par cet ordre partiel peuvent être exécutées en parallèle. Le travail des processeurs est ici guidé par les dépendances entre les tâches.

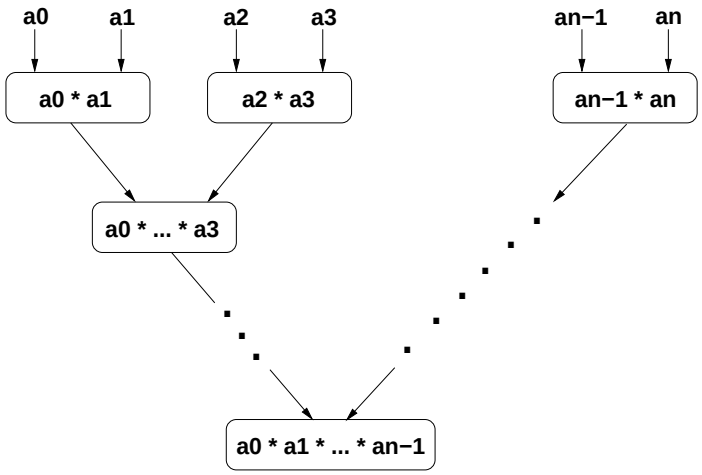


FIG. 4.3 – Exemple de graphe de précédence : produit itéré par accumulation.

Dans le parallélisme de contrôle totalement explicite on retrouve aujourd'hui plusieurs modèles accompagnés de leur paradigme de programmation. Les trois plus répandus sont analysés dans la suite.

Modèle à mémoire partagée

La programmation par mémoire partagée est une solution adaptée à l'utilisation des machines SMP (multiprocesseurs). La communication est faite directement par variables partagées contrôlées par des mécanismes de synchronisation (typiquement des sémaphores et des verrous). La plupart des systèmes d'exploitation existants pour ce type de machine mettent à disposition une interface de programmation implantant ce modèle. Son inconvénient est le fait d'être trop lié à un style d'architecture de machine, ce qui réduit la portabilité des programmes.

Modèle par échange de messages

L'échange de messages est la technologie de base qui permet la réalisation de communications entre les processeurs d'une architecture à mémoire distribuée. Son implantation est faite *via* deux primitives très simples, *send* et *receive*, dont les paramètres sont l'identification du processeur partenaire pour la communication et le message à envoyer. Ce modèle, essentiellement le même pour n'importe quelle machine à mémoire distribuée, a donné de nombreuses variantes. Un standard pour l'interface de programmation par échange de messages a été élaboré afin d'assurer la portabilité et la facilité d'utilisation, il s'agit de MPI (*Message Passing Interface*)[SOHL+96].

Dans ce modèle, le problème du contrôle du flux des messages est aussi très important. Deux sémantiques existent : synchrone (avec rendez-vous)et asynchrone (canal non borné). Pour une communication synchrone, émetteur et récepteur doivent demander à communiquer pour que la transmission ait lieu. Dans les communications asynchrones, l'émetteur fait son émission à n'importe quel moment et le message sera mémorisé et délivré au récepteur quand celui-ci le demandera. La plupart des bibliothèques de communication offre un modèle à canal borné et laisse à la charge du programmeur le calcul de la taille des tampons (qui stockent les messages) nécessaires à son application pour éviter les blocages.

Modèle à appel de procédure à distance

L'appel de procédure à distance (RPC - *Remote Procedure Call*) [BN84] étend le mécanisme conventionnel d'appel de procédure des langages comme C. Un RPC est un appel de procédure entre deux processeurs distincts, l'appelant et l'appelé. Quand un processus d'un processeur réalise un appel vers une procédure à distance sur un autre processeur, le processus de l'appelé reçoit les paramètres d'entrée, exécute la procédure et renvoie les résultats vers l'appelant.

Le RPC est un mécanisme synchrone. Lorsqu'un processeur réalise un appel à distance il reste bloqué jusqu'à la réception du résultat envoyé par le processeur appelé. Ceci est un facteur limitant du degré de parallélisme. L'exécution de la procédure appelée peut être faite de plusieurs façons. Les plus répandues

sont la définition d'un processus comme le serveur qui s'occupe d'exécuter les appels de procédure l'un après l'autre ou la création d'un serveur pour chaque appel.

4.4 Critères d'évaluation

Dans cette section, nous allons introduire les principaux critères d'évaluation que nous allons utiliser pour mesurer la performance de nos programmes parallèles. Ce sont des critères d'un niveau intermédiaire de détail, c'est-à-dire qu'ils ne sont pas appropriés pour toutes les situations : ils sont orientés vers les architectures avec multiprocesseurs et ne prennent pas en compte, par exemple, le fonctionnement de la mémoire cache. Néanmoins, il s'agit de critères qui se sont montrés très utiles dans un large spectre de problèmes de parallélisation d'algorithmes [Fos95].

Les critères de performance qui seront présentés dans la suite sont basés sur une mesure en particulier : le temps d'exécution. Celui-ci est considéré comme étant une fonction de la taille du problème n, du nombre de processeurs p utilisés ainsi que d'autres caractéristiques de l'architecture et de l'algorithme :

$$T = f(n, p, ...)$$

Le temps d'exécution d'un algorithme parallèle peut être défini comme le temps qui s'écoule dès que le premier processeur démarre son calcul jusqu'à ce que le dernier processeur complète son exécution. Pendant l'exécution, chaque processeur peut être dans trois états différents ce qui résulte en trois composantes du temps d'exécution :

- **Temps de calcul** (T_{calc}) - ce temps représente le temps qu'un processeur passe effectivement à perpétrer le calcul directement utile pour l'application ;
- **Temps de communication** (T_{comm}) - le temps utilisé par le processeur pour gérer et réaliser les communications. Il est proportionnel au volume des communications. Pour un fonctionnement efficace d'un programme parallèle ce temps doit être le plus petit possible ;
- **Temps d'attente** (T_{att}) - c'est le temps d'inactivité du processeur provoquée par l'attente soit d'un événement, soit d'une communication, soit d'une synchronisation. Sa présence est le signe soit d'un déséquilibre de charge entre les nœuds de calcul, soit d'un mauvais entrelacement des calculs et des communications. Il doit être également le plus faible possible.

Ainsi, on pourrait décrire le temps d'exécution d'un programme parallèle comme étant la somme de tous ces temps sur tous les processeurs i divisée par le nombre de processeurs p :

$$T = \frac{1}{p} \times \left(\sum_{i=0}^{p-1} T_{calc}^{i} + \sum_{i=0}^{p-1} T_{comm}^{i} + \sum_{i=0}^{p-1} T_{att}^{i} \right)$$

Néanmoins, cette description du temps parallèle ne correspond pas à la réalité de la plupart des cas car elle est basée sur l'idée que tous les processeurs vont terminer leur travail au même temps. Cette formule ne permet pas de prendre en compte le temps du processeur le plus lent lorsqu'on travaille avec des machines hétérogènes ou avec des applications irrégulières (cf. paragraphe 4.5.4). Dans ce type de situation, le temps parallèle est tout simplement le résultat de l'addition des trois types de temps décrits ci-dessus uniquement pour le **processeur le plus lent** qui est celui qui retardera le résultat final.

4.4.1 Accélération et efficacité

L'accélération mesurée d'un algorithme sur p processeurs (A_p) est définie comme le rapport du temps d'exécution séquentiel du meilleur algorithme T_{seq} divisé par le temps d'exécution parallèle de l'algorithme sur p processeurs (T_p) :

$$A_p = \frac{T_{seq}}{T_p}$$

Plus précisément, T_p est le maximum des temps d'exécutions mesurés sur chacun des processeurs. On compare toujours l'accélération obtenue par un programme parallèle à l'accélération optimale laquelle se produirait théoriquement lorsque T_{seq}/T_p est égale à p (lorsque l'on n'a pas de surcoût du au parallélisme et T_p est le résultat direct de la division de T_{seq} par p).

L'efficacité d'un algorithme (E_p) est définie comme le rapport de l'accélération de l'algorithme sur p processeurs divisée par le nombre de processeurs.

$$E_p = \frac{A_p}{p} = \frac{T_{seq}}{p \times T_p}$$

Il est souvent exprimé par un pourcentage qui représente l'utilisation moyenne des processeurs par rapport à une parallélisation parfaite. Une parallélisation parfaite signifie que tous les processeurs sont en permanence utilisés pour effectuer des opérations utiles pour l'application (E_p se rapproche de 100%).

4.4.2 Déséquilibre de charge

Une deuxième mesure qui offre des précisions sur l'exécution d'un programme parallèle est le déséquilibre de charge (D_p) de travail entre les processeurs. Par son intermédiaire, on peut calculer le pourcentage du temps d'exécution parallèle (T_p) durant lequel le processeur le plus rapide T_{min} reste en attente après avoir fini son travail :

$$D_p = \frac{T_p - T_{min}}{T_p}$$

Vérifier le déséquilibre de charge entre les nœuds d'un programme parallèle permet d'évaluer la distribution du travail parmi les processeurs. Plus cette distribution est équitable (le valeur de D_p est proche de zéro), plus l'accélération obtenue par le parallélisme sera importante.

4.4.3 Qualité du résultat

Les mesures de performance citées précédemment sont des outils que l'on peut utiliser pour affiner un algorithme parallèle d'une application donnée. Elles permettent d'évaluer le comportement du programme parallèle en offrant des renseignements par rapport à l'utilisation que celui-ci fait des ressources des machines dont il dispose.

Mais cela n'est pas du tout car, bien souvent, lorsque l'on veut paralléliser un algorithme séquentiel, les modifications que l'on doit faire sont tellement importantes que l'on se rapproche de la création d'un nouvel algorithme. Et, à ce moment là, il est au moins nécessaire de préserver la qualité de résultat que l'algorithme séquentiel était capable d'offrir.

Cette qualité se présente sur la forme de la correction du calcul final lorsque l'on travaille avec des algorithmes numériques tels que le produit matriciel par exemple. Mais elle peut aussi se présenter sur une forme moins stricte, comme dans le cas du problème que nous intéresse dans cette thèse c'est-à-dire l'étendue de la zone de propagation lors de l'application d'un algorithme de croissance de régions dans l'appariement de deux images. La parallélisation de cet algorithme peut engendrer des pertes ou des gains dans le nombre de paires de points correctement appariés selon la stratégie adoptée. Ainsi, pour avoir une parallélisation réussie, il faut une réduction du temps d'exécution (avec une bonne utilisation des ressources de préférence) qui préserve au moins la qualité d'appariement obtenue avec le programme séquentiel.

4.5 Facteurs de performance

La parallélisation totalement explicite comporte plusieurs étapes : il faut identifier la source de parallélisme que le problème présente pour ensuite découper le problème en tâches. Il faut aussi définir un ordre d'exécution et associer à chacune des tâches un processeur chargé de leur exécution. Toutes ces étapes font de la parallélisation totalement explicite d'un problème un travail d'expertise, il n'est pas possible de chercher une approche type car chaque problème possède ses propres caractéristiques. Les sections qui suivent résument les questions importantes

concernant la parallélisation totalement explicite d'un problème.

4.5.1 Choix de la granularité

Choisir la granularité d'un programme parallèle consiste à définir ce qu'est une tâche lors de la description d'un algorithme parallèle sous la forme d'un graphe de tâches.

La granularité d'un programme parallèle peut être diminuée par le découpage de chaque tâche en sous-tâches plus petites. On parle alors d'un parallélisme à **grain fin**. En pratique, la diminution de la granularité augmente le parallélisme potentiel et permet d'utiliser plus de processeurs. Inversement, augmenter la granularité consiste à regrouper des tâches. Dans ce cas on parle d'un parallélisme à **grain gros**. Augmenter la granularité permet de réduire le temps nécessaire à la gestion du parallélisme.

Il existe un rapport important entre le type des machines et la granularité qu'elles peuvent traiter efficacement. Les ordinateurs parallèles à mémoire partagée peuvent gérer un parallélisme de grain fin avec un sur-coût faible, mais le nombre de processeurs d'une telle machine est limité (cf. paragraphe 4.2.1). Ainsi, le parallélisme potentiel important induit par le grain fin ne peut pas toujours être exploité de la meilleure façon. D'un autre côté, les machines à mémoire distribuée permettent d'exploiter le maximum de parallélisme (cf. paragraphe 4.2.2). Mais, dans ce cas, un grain trop fin entraîne souvent trop de communications entre les processeurs et un grain trop gros mène à des déséquilibres de charge entre les nœuds de calcul. Le choix de la granularité est alors un compromis entre le maximum de parallélisme potentiel, un sur-coût de gestion minimum et un bon équilibre de charge entre les nœuds de calcul.

4.5.2 Ordonnancement

L'ordonnancement d'un graphe de précédence consiste à attribuer une date et un site de début d'exécution aux tâches qui composent le graphe. Ces dates doivent être compatibles avec le nombre de processeurs disponibles et aussi avec l'ordre partiel défini par le graphe de précédence. La stratégie d'ordonnancement à employer dépend des caractéristiques du graphe de tâches. Si le problème est régulier (voir paragraphe 4.5.4), le graphe de tâches est complètement déterminé et, dans ce cas, on utilise des stratégies d'ordonnancement statique. L'ordonnancement statique d'un graphe de précédence quelconque est un problème NP-complet, mais il en existe de bonnes heuristiques [CR87] [Gui95].

Dans la plupart des problèmes, le graphe de précédence dépend des données en entrée du programme. Il s'agit alors d'un problème irrégulier (voir paragraphe 4.5.4) et ce n'est pas possible d'utiliser l'ordonnancement statique. Dans ce cas, on peut utiliser parfois une stratégie d'ordonnancement en ligne laquelle ordonne les tâches au fur et à mesure de leur création. L'utilisation de l'ordonnancement en ligne nécessite un mécanisme de migration de tâches qui permet d'arrêter l'exécution d'une tâche et de la reprendre sur un autre processeur.

Pour certains problèmes, il est possible d'utiliser des solutions intermédiaires entre les ordonnancements statiques qui nécessitent la connaissance complète du graphe de tâches et les ordonnancements en ligne qui utilisent peu d'informations sur le graphe de tâches. Un exemple d'une telle solution intermédiaire est la stratégie d'ordonnancement dynamique développée pour la parallélisation d'un algorithme de dynamique moléculaire opérationnelle en [Ber97].

4.5.3 Placement

Le placement des tâches d'un graphe de précédence consiste à associer un site d'exécution à chacune des tâches du graphe. Dans les architectures parallèles à mémoire distribuée, le placement doit chercher à minimiser les communications en plaçant sur le même processeur des tâches ayant beaucoup de données à échanger. Un autre devoir du placement est la réalisation d'une régulation de la charge de travail entre les processeurs visant un meilleur équilibre. Une bonne stratégie de placement passe forcément par la recherche d'un compromis entre l'équilibre de charge et la minimisation des communications.

À l'exemple de l'ordonnancement, la stratégie de placement à adopter dépend de ce que l'on connaît des caractéristiques du problème. Si le problème est régulier (voir paragraphe 4.5.4), on utilise un placement statique [Gui95]. À partir du résultat de l'ordonnancement, on construit un graphe non orienté dont les sommets reçoivent des valeurs selon les coûts de calcul et dont les arêtes sont associées aux coûts de communication des données. Le placement statique est une optimisation combinatoire dont la fonction à minimiser rend compte du déséquilibre de charge et des communications dues au placement.

Si le problème est irrégulier (voir paragraphe 4.5.4), on utilise des algorithmes de placement dynamique et des mécanismes de régulation dynamique de la charge de calcul sur les processeurs car le coût des tâches est inconnu ou imparfaitement estimé. En d'autres mots, il faut placer les tâches sur les processeurs en cours d'exécution. Les méthodes de répartition dynamique comportent quatre mécanismes essentiels [**?**] :

– **Évaluation de la charge** - mécanisme pour déterminer la charge de chacun

des processeurs ;
- **Évaluation du déséquilibre de charge** - détermination de la surcharge ou de la sous-charge de chacun des nœuds par l'intermédiaire de la comparaison des charges de chacun des processeurs ;
- **Sélection des tâches à migrer** - sélection des tâches dont la migration est la plus efficace pour l'équilibre de charge ;
- **Migration des tâches** - déplacement effectif d'une tâche de calcul d'un nœud vers un autre.

Il est clair que les mécanismes de répartition dynamique de la charge augmentent le volume des communications. Ainsi, un bon algorithme de répartition doit minimiser les communications et doit également consommer un minimum de temps de calcul pour ne pas ralentir l'application. Une difficulté importante qui s'oppose à l'application d'un algorithme de répartition de charge dynamique est qu'il est souvent difficile de définir précisément ce qu'est la charge d'un processeur car cela dépend de la sémantique du programme. Pour résoudre ce problème, beaucoup d'applications intègrent des mécanismes de régulation spécialement adaptés au problème.

4.5.4 Régularité/irrégularité

Certaines taxinomies [Ran93] classent les algorithmes parallèles en fonction de la relation entre la complexité de calcul et la complexité de communication. Il est donc envisageable qu'un programme parallèle offre un rapport calcul/communication bien adapté à la machine cible. Savoir si une application est adaptée à une machine suppose la connaissance préalable du comportement de l'application en termes de phases de calcul et communication et leurs durées. Cette connaissance se représente par un graphe de dépendance dont les arêtes sont associées à des coûts. En fonction de ce graphe de dépendance, les algorithmes parallèles sont classés en **réguliers** et **irréguliers** [GRV95]

Un algorithme régulier est celui dont le comportement est connu avant son exécution ou qui peut se prédire à chaque étape de son exécution. Cela nous permet d'organiser un programme parallèle en tâches qui possèdent un rapport calcul/communication équivalent et composées de façon à minimiser les communications.

En ce qui concerne les algorithmes irréguliers, on ne peut prédire leur comportement qu'au moment de leur exécution ou alors, pour connaître leur comportement, il faut pouvoir calculer l'état global de l'application, ou encore ce comportement présente une grande variabilité. Ce type d'algorithme est caractérisé par la création dynamique d'un nombre important de tâches de grain divers car le type de découpage du programme parallèle comprend des sous-problèmes de taille variable. L'ordonnancement de ces sous-problèmes de façon équitable, pour éviter

les déséquilibres de charge sur différents nœuds participant à l'exécution, n'est pas trivial.

4.6 Conclusion

Dans ce chapitre, nous avons voulu offrir au lecteur un panorama des possibilités existantes dans le calcul parallèle. L'idée était de présenter les différents aspects du domaine du parallélisme sans entrer trop dans les détails pour pouvoir ensuite insérer le travail de cette thèse dans ce contexte.

L'application cible de ce travail s'insère dans le groupe des applications dites **irrégulières** car il est impossible de prévoir son comportement en avance. Chaque nouvelle paire d'images à apparier présente une évolution différente de l'algorithme de propagation. Si l'on décide de découper les images, il y aura toujours un nombre imprévisible d'appariements de départ sur chaque surface. Si l'on décide de baser la division du travail sur les appariements de départ, l'indéterminisme est toujours présent car on ne peut pas prévoir quelle sera la surface des images appariées à partir d'une seule paire de départ.

Le type d'architecture que nous prétendons utiliser est une **architecture à mémoire distribuée**. Ce choix a été guidé par la disponibilité de plus en plus grande de ce genre de machine. La dissémination des grappes des processeurs est une réalité et cela nous permettra d'avoir des versions parallèles de l'algorithme de propagation portables et extensibles. L'extensibilité est une caractéristique désirable car ainsi les méthodes proposées dans ce travail pourront être utilisées sur des images de taille variable (plus l'on peut facilement rajouter des processeurs, plus les images que l'on peut traiter sont grosses). Le choix des architectures à mémoire distribuée implique dans complexité plus grande dans le développement des version parallèles car il faudra gérer la communication entre les nœuds et plus tard la distribution du travail parmi eux. Le modèle de programmation parallèle utilisé est le **modèle par échange de messages**.

Finalement, les critères d'évaluation que nous allons utiliser sont ceux qui ont été présentés dans ce chapitre (section 4.4) : l'accélération pour vérifier si les ressources ne sont pas sous-exploitées, le déséquilibre pour vérifier la correcte distribution du travail parmi les nœuds, la qualité du résultat pour garder un niveau de l'appariement final proche de la version séquentielle et, principalement, le temps d'exécution pour comparer l'évolution des différentes versions.

Troisième partie

Les Solutions Proposées

Chapitre 5

Études initiales

Sommaire

5.1 Introduction

Nous avons vu dans les chapitres précédents le contexte dans lequel ce travail s'insère. Désormais, nous allons discuter et présenter les solutions qui nous semblent les plus adaptées au problème de la mise en parallèle de l'algorithme de croissance de régions basé sur une stratégie "meilleur d'abord".

Dans ce chapitre, nous voulons montrer au lecteur les hypothèses de base qui nous ont servi de point de départ pour notre étude de parallélisation. Ces hypothèses ont été formulées en prenant en compte les limitations imposées par certaines caractéristiques de l'algorithme séquentiel et le type de machine sur lequel nous voulons exécuter notre application de synthèse d'images réelles.

Il est important de rappeler au lecteur que nous sommes dans le cadre d'un travail expérimental, lequel cible une application spécifique, ce qui nous oblige à rester très pragmatiques par rapport aux résultats recherchés. De plus, les solutions qui seront proposées dans la suite de ce chapitre et des prochains doivent essayer de garder l'efficacité de l'algorithme séquentiel original en termes de qualité visuelle des images obtenues. Malgré cela, nous avons essayé de faire en sorte que les adaptations faites à l'algorithme initial puissent être facilement généralisées à n'importe quelle application utilisant une technique de croissance de régions basée sur une stratégie de progression "meilleur d'abord" globale.

Ce chapitre commence par une analyse des parallélisations proposées pour d'autres méthodes usuelles de croissance de régions. Ensuite, des hypothèses de base pour notre travail sont formulées en fonction des caractéristiques originales de l'algorithme proposé par [Lhu00]. Pour chacune des hypothèses, une étude de viabilité est faite et des résultats pratiques sont montrés. Finalement, des pistes sont énumérées pour la suite.

5.2 Croissance de régions en parallèle

La technique générale de croissance de régions est employée depuis très longtemps dans le domaine de la segmentation d'images. Son principe de base est l'utilisation de caractéristiques des images pour grouper des pixels voisins et former des régions. Ces régions sont alors fusionnées avec d'autres régions pour en former autres plus étendues. Une région peut correspondre à une partie significative ou à un objet du monde réel.

La simplicité et l'adaptabilité de la technique de croissance de régions ont permis son utilisation sur plusieurs autres domaines scientifiques et techniques. Cette diffusion, ajoutée à sa grande gourmandise de calcul, a créé un besoin qui

très rapidement a abouti à plusieurs études et propositions de parallélisation.

Ainsi, nous pouvons retrouver divers travaux de parallélisation de la technique de croissance de régions appliquée à un grand nombre de domaines. Les machines parallèles utilisées ont varié selon l'époque et les architectures existantes. Nous ne prétendons pas faire ici une liste exhaustive de tous les travaux de parallélisation de la technique de croissance de régions faits jusqu'aujourd'hui. Mais nous voulons en citer quelques uns pour donner au lecteur une notion de la diversité des chercheurs qui ont étudié le problème de la parallélisation de la technique de croissance de régions appliquée uniquement au domaine de l'imagerie.

Plusieurs approches existent pour résoudre le problème de croissance de régions ([AP92], [BB], [Zuc76]). Une des approches les plus répandues est celle basée sur la stratégie "Division et Fusion" (*Split and Merge*) [HP74]. Dans cette approche, l'algorithme employé pour former les régions possède une étape initiale dans laquelle les images sont découpées en régions carrées qui vérifient un critère donné d'homogénéité. Après ce découpage, les régions carrées sont alors fusionnées à tour de rôle en régions plus grosses toujours en conformité avec le critère d'homogénéité jusqu'à ce que plus aucune fusion ne soit possible. L'étape de fusion est réalisée *via* la formation d'une représentation du problème à travers un graphe non directionnel où chaque connexion possède un poids associé. Les extrémités du graphe représentent les régions dans l'image et les connexions entre les extrémités représentent les relations de voisinage entre les régions

Les premières versions parallèles de l'algorithme de croissance de régions basée sur l'approche "Division et Fusion" ont été implantées sur des machines du type SIMD et employaient des structures dynamiques pour stocker les informations sur les régions identifiées dans les images ([Til88], [WLR90]).

Plus tard, dans [CRFS] et [CRFS94] les auteurs proposent des implantations parallèles de la technique de croissance de régions basée sur l'approche de "Division et Fusion" en utilisant seulement des tableaux d'une ou deux dimensions pour stocker les informations sur les régions identifiées dans les images. Les tableaux de deux dimensions contiennent les informations relatives aux pixels telles l'intensité par exemple. Les tableaux d'une dimension sont utilisés pour garder l'information sur le graphe qui modélise le problème. Deux versions parallèles ont été implantées : une basée sur le modèle de parallélisme de données (machine SIMD)et l'autre en utilisant le modèle par échange de messages (machine MIMD). Celle qui nous intéresse le plus est la deuxième. Dans cette version, l'image est découpée sur la grille de processeurs à disposition. Chaque processeur reçoit une sous-image de l'image originale et y fait indépendamment sa division en carrés homogènes. Après, chaque nœud établit les extrémités et connexions du graphe associé à sa sous-image. Des informations sur les bords sont échangées entre les processeurs de façon à permettre l'établissement des connexions à des extrémités sur d'autres

processeurs. Dans la suite, les processeurs coopèrent pour fusionner les régions carrées homogènes et pour mettre à jour les extrémités et connexions de leurs graphes. L'algorithme évolue ainsi jusqu'à ce que la fusion des régions ne soit plus possible. Cette version a été implanté sur la machine CM-5 (*Connexion Machine 5*). Une technique similaire de parallélisation de l'algorithme de croissance de régions basée sur l'approche "Division et Fusion" a aussi été développée presque à la même époque par quelques chercheurs du *Northeast Parallel Architecture Center* de l'Université de Syracuse aux États-Unis ([ORF92], [ORF93]).

Une autre étude expérimentale des versions parallèles d'algorithmes pour la segmentation d'images basée sur la technique de croissance de régions (aussi basée sur la stratégie "Division et Fusion") a été présentée dans [JBHD95]. Dans ce travail, développé au sein de l'*Intitute for Advanced Computer Studies* de l'Université de Maryland aux États-Unis, les auteurs se concentrent dans la proposition d'une nouvelle version de l'algorithme pour la détermination des composants connectés d'une image dans lequel une nouvelle approche parallèle est présentée pour effectuer l'étape de fusion. L'implantation de ce nouvel algorithme a été portée sur une variété de différentes machines telles que CM-5, IBM SP-1 et SP-2, Cray T3D, Meiko Scientific CS-2, Intel Paragon et grappes de processeurs.

Finalement, nous citons aussi une étude de la parallélisation de l'algorithme de croissance de régions basé sur la stratégie "Division et Fusion" qui a été proposée dans [MGG99]. Les versions ont été conçues pour le modèle de programmation parallèle basé sur l'échange de messages et les implantations ont été testées sur une machine Cray-T3E. Les auteurs présentent quatre implantations parallèles qui englobent des mécanismes de régulation de la charge de travail pour augmenter l'efficacité de leurs solutions parallèles. Ils proposent deux stratégies de régulation de charge : une statique et l'autre dynamique.

Les travaux cités ci-dessus illustrent la variété d'études faites sur la parallélisation de l'algorithme de croissance de régions basé sur l'approche "Division et Fusion". Néanmoins, il faut souligner que nous étudions dans ce travail une technique différente de croissance de régions. Il s'agit d'une technique qui a été proposée très récemment dans le travail de [Lhu00] et son originalité réside dans l'adoption d'une stratégie "meilleur d'abord" pour sélectionner le prochain appariement germe de la propagation à partir d'un ensemble d'appariements de départ lequel est mis à jour avec l'addition de chaque nouvelle paire appariée par l'itération précédente de l'algorithme. Ainsi, l'étude de la mise en parallèle de cette nouvelle technique de croissance de régions reste, à notre connaissance, un problème original. Plus encore si l'on considère les particularités relatives au contexte de synthèse d'images à partir d'images réelles, telles que les restrictions apportées par l'insertion de l'algorithme dans une chaîne d'algorithmes utilisant les mêmes données pour en donner un exemple.

5.3 Approches

À partir de l'analyse de l'algorithme de propagation faite dans le chapitre 3, nous avons choisi deux groupes de données qui vont orienter nos tentatives de parallélisation : les images source et les appariements de départ.

5.3.1 Parallélisation basée sur les images

La paire d'images source offre un ensemble de données constant sur lequel évolue l'algorithme de propagation. Une parallélisation orientée par les images doit forcément passer par un découpage de celles-ci en zones ou régions. Pour chaque zone de la première image, une correspondante existe dans la deuxième. Chacune de ces paires de zones permet une évolution isolée de l'algorithme de propagation à partir des appariements de départ qui y sont entièrement.

Cette approche offre l'avantage d'être simple à programmer car peu de modifications doivent être ajoutées à l'algorithme séquentiel. D'un autre côté, l'isolation des régions, la division elle même des images et le manque d'un contrôle central sont des facteurs qui peuvent influencer sur la qualité finale de l'appariement global.

5.3.2 Parallélisation basée sur les appariements de départ

Avant le démarrage de la phase de propagation, des étapes précédentes de l'application ont fait une analyse des images source pour y détecter des points avec des caractéristiques plus marquantes. Ces points sont alors appariés avec des points de l'autre image de la paire pour former un ensemble d'appariements qui seront utilisés comme des graines pour initialiser l'algorithme de propagation.

Nous considérons que cet ensemble d'appariements de départ peut aussi servir de critère pour la distribution du travail parmi les nœuds dans une parallélisation de l'algorithme de propagation. Les appariements de départ peuvent être distribués en sous-ensembles selon un ordre préétabli et attribués à un nœud. La propagation commencée sur un nœud quelconque n'est pas contrainte par une limite de zone comme dans l'hypothèse précédente. Par contre, il n'existe pas de contrôle central du calcul effectué sur chaque processeur, ce qui peut entraîner selon les caractéristiques des images traitées beaucoup de travail redondant.

5.4 Découpage des images

Dans cette section nous allons expliquer comment a été implantée dans la pratique la première hypothèse de parallélisation sur le code original de l'algorithme

séquentiel. Nous allons ensuite présenter les résultats obtenus pour conclure sur une analyse des limitations de cette méthode.

5.4.1 Implantation

Les problèmes que nous avions pour modifier le code séquentiel de l'application étaient :

- établir une façon automatique de découper les images ;
- établir une façon de traiter les paires de points de départ ;
- ajouter ces modifications au code de façon à minimiser la perte de qualité du résultat final obtenu par le code séquentiel.

Dans cette première expérimentation, nous voudrions commencer pour vérifier jusqu'à quel point l'application était irrégulière. Donc, nous avons opté pour une première parallélisation sans aucun souci par rapport à la régulation de charge (laquelle a été considérée a posteriori). Ainsi, sauf lorsque nous dirons le contraire, nous allons travailler toujours avec une zone par processeur.

Pour découper les images au moins quatre options ont été considérées : découpage en tranches horizontales de même taille, découpage en tranches verticales de même taille, découpage en zones de tailles différentes et découpage en rectangles de même taille (figure 5.1).

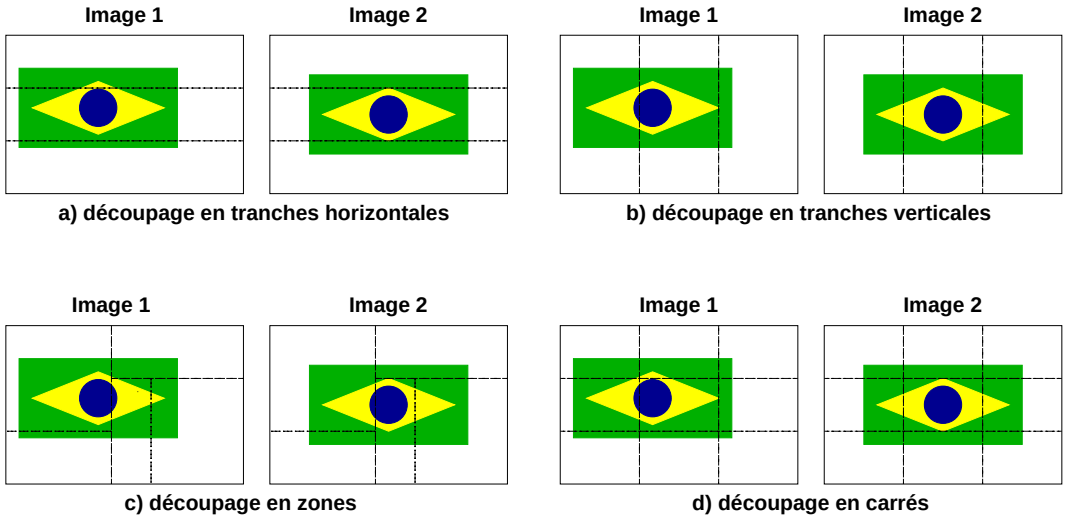


FIG. 5.1 – Types de découpage possibles des images.

Au premier abord, le découpage en zones de taille différente est le plus attirant, puisque il permettrait la recherche d'un équilibre entre le rapport surface de la

zone/nombre de paires sur chaque processeur. Ainsi, chaque processeur aurait une quantité similaire de calcul à faire. Cette option serait par contre très coûteuse et difficile à automatiser car il faudrait considérer la localisation des appariements germes en fonction du nombre de zones au moment de choisir le découpage. Sans parler de la difficulté pour implanter un algorithme qui travaillerait sur des zones de tailles et formes différentes.

Le découpage en régions carrées de même taille a aussi été envisagé, mais nous avons rapidement écarté cette possibilité car très peu d'images auraient les dimensions exactes qui permettrait un découpage précis. En plus, il serait très difficile d'adapter le découpage des images source à des différentes configurations de processeurs, car il ne faut pas oublier que le nombre de carrés serait déterminé directement par le nombre de processeurs employés. En conclusion, nous ne voyons pas les avantages d'un tel découpage pour compenser les inconvénients de son implantation.

On arrive alors à la possibilité de découper les images en régions rectangulaires de même taille. Dans ce cas, le moins coûteux à faire est de prendre une zone qui découpe les images d'un bout à l'autre et ainsi nous sommes arrivés aux tranches. La seule chose qui restait à faire était de choisir entre tranches verticales ou horizontales (selon la largeur ou la hauteur des images).

Or, l'expérience dit que le mouvement naturel de déplacement d'une caméra lorsqu'on prend plusieurs scènes d'un paysage d'extérieur se fait dans le sens horizontal. Cette observation plus le fait que dans notre version nous ne pouvons utiliser que les paires dont les deux points sont entièrement situés dans la surface d'une tranche, mène au choix d'un **découpage horizontal**. Autrement, avec un découpage vertical et un déplacement horizontal de la caméra entre la prise de deux images sources, les points qui composent une paire (chaque point sur une des images source) seraient plus facilement situés sur des tranches différentes et, par conséquent, ignorés par notre algorithme de sélection. Ce problème serait encore plus grave au fur et à mesure que le nombre de tranches augmenterait. Ainsi, le choix d'un découpage horizontal est le plus approprié car il limite la perte d'appariements de départ due au problème des bords (lequel est expliqué plus en détails dans la section 5.4.2, paragraphe intitulé **qualité de l'appariement**).

Une fois choisie la façon de découper les images, il a fallu penser au traitement des points de départ. Dans l'algorithme séquentiel nous avons vu que ces paires de points sont stockées dans une structure de tas et à chaque itération de l'algorithme une nouvelle paire est retirée du tas.

Pour cette première version parallèle de l'application nous avons choisi de dupliquer les données et le code sur tous les processeurs. Ce choix est justifié car nous ne cherchons pas à optimiser l'occupation de la mémoire. L'idée est de vérifier la robustesse de la stratégie "meilleur d'abord" lorsque l'ensemble des points de

départ n'est pas entièrement à sa disposition. L'intérêt étant de savoir quels seront les effets de cette restriction sur la qualité finale de l'appariement des images source.

Nous avons alors associé chaque paire de tranches correspondantes sur les images à un processeur. Ainsi, nous avons toujours n tranches sur n processeurs. Si l'on se souvient que chaque processeur a une copie complète du tas de paires de points de départ, il va de soi qu'il a fallu trier en temps d'exécution les paires appartenant à une tranche donnée. Cela a été fait *via* l'insertion d'un test sur l'algorithme de propagation comme on peut le voir à la ligne 4 de l'algorithme 2. Avant de commencer la propagation, on teste si la paire retirée du tas appartient à la tranche T_{proc} attribuée au processeur *proc*. La hauteur en pixels de T_{proc} étant définie automatiquement en fonction du nombre de processeurs utilisés.

Algorithme 2 Modification sur l'algorithme de propagation

1: $Carte \leftarrow \emptyset$
2: **Tant que** $Germes \neq \emptyset$ **Faire**
3: retirer la meilleure paire (a, A) de $Germes$
4: **Si** $(a, A) \in T_{proc}$ **Alors**
5: $Local \leftarrow \emptyset$
6: stocker dans $Local$ les paires candidates
7: garder dans $Germes$ et $Carte$ les paires finales
8: **Fin si**
9: **Fin tant que**

5.4.2 Résultats

Dans cette section, nous allons présenter les résultats obtenus par l'exécution de notre première version parallèle de l'application développée par [Lhu00]. Notre objectif est d'offrir au lecteur un échantillon des performances de cette version parallèle pour permettre une comparaison avec les résultats séquentiels et ceux des prochaines versions parallèles qui seront proposées.

Pour chaque exemple, nous allons fournir des informations sur la qualité des appariements obtenus, l'accélération et la distribution de travail parmi les processeurs. Un tableau comparatif entre le temps séquentiel et le temps parallèle sera montré aussi. La même chose sera faite pour le nombre d'appariements obtenu sur les images après l'exécution de l'algorithme de propagation.

Les résultats présentés dans la suite ont été calculés sur une grappe de 225 processeurs Pentium III 733 Mhz sur Linux connectés via un réseau Ethernet 100. Les nœuds sont distribués en cinq groupes de 43 gérés par un commutateur du type *mesh* de 1 Go (pour plus de détails, voir http ://www-id.imag.fr/Grappes/icluster/UsersGuide.html). Chaque valeur présentée dans la suite est le résultat d'une moyenne arithmétique simple de dix exécutions en éliminant les valeurs extrêmes. Les figures qui montrent l'étendue de la propagation sur les images exemples sont le résultat d'une seule exécution.

Cette version a aussi été exécutée sur une machine SMP avec quatre processeurs. Les résultats ont été présentés dans [FMD01].

Qualité de l'appariement

Le principal problème de l'approche basée sur les tranches est la perte de plusieurs appariements à cause de la limitation imposée par la division en tranches des images. En effet, il aura toujours des paires de points à la proximité des bords des tranches où un des points reste en dehors de la tranche associée à son processeur Dans la figure 5.2, les paires (i, I), (c, C), et (g, G) montrent cette situation.

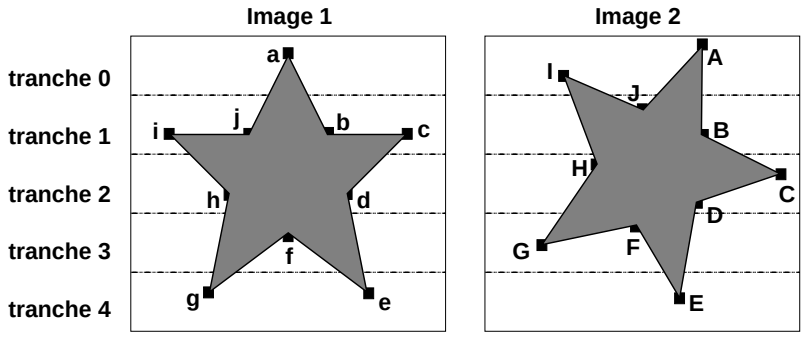


FIG. 5.2 – Problème des bords : exemple simplifié.

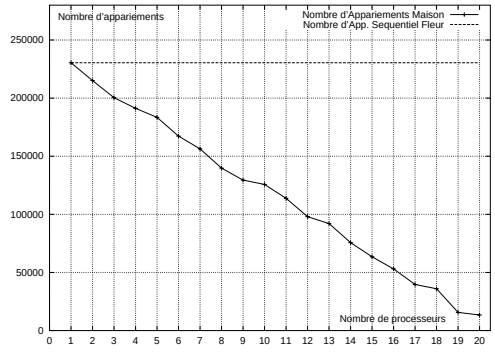
Cette limitation réduit le nombre des appariements graines pour l'algorithme de propagation et par conséquent induit une perte de qualité de l'appariement final. Ce problème devient plus important au fur et à mesure que le nombre de processeurs (tranches) augmente.

Un autre problème qui affecte la qualité de l'appariement final des images est le blocage de la progression de la propagation aux bords d'une tranche. Ainsi, l'algorithme de propagation ne peut pas atteindre certaines régions des images puisqu'elles ne peuvent être appariées qu'à partir d'un seul appariement de départ (qui se trouverait en dehors de la tranche considérée). Ce problème est moins fréquent que le précédent, car l'algorithme de croissance de régions proposé par

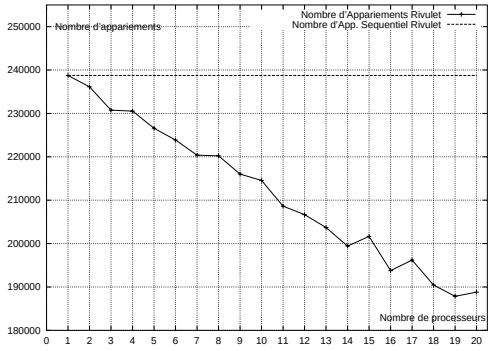
[Lhu00] est très redondant (cf. [LQ00b]). C'est-à-dire : l'ensemble des appariements de départ contient plusieurs paires qui peuvent dans certains cas provoquer une avalanche de nouveaux appariements sur une région texturée et, très souvent, ces régions se superposent.

Par contre, ce problème devient aussi plus important au fur et à mesure que le nombre de processeurs (tranches) augmente et c'est à cause de lui que nous avons été obligés d'analyser visuellement les appariements finals. C'est-à-dire : il faut considérer qu'à partir d'un certain nombre de processeurs selon les dimensions des images nous devrons faire face au problème des tranches qui ne sont pas appariées du tout car elles n'ont pas d'appariements de départ. Désormais, nous allons faire référence à cet aspect de la qualité du résultat par l'expression **qualité visuelle**.

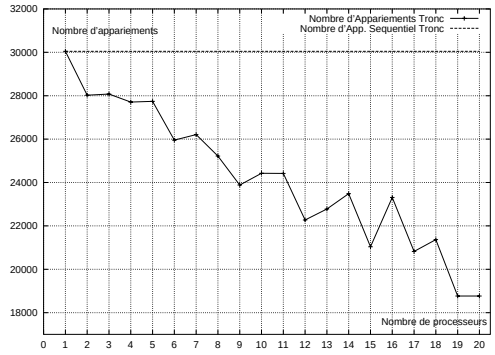
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

FIG. 5.3 – Version Tranches : nombre total d'appariements

Dans cette première version parallèle, nous ne nous sommes pas occupés d'implanter une façon de contrôler la qualité des appariements obtenus. En fait, un des buts de cette parallélisation était de vérifier jusqu'à quel point les modifications ajoutées à l'algorithme séquentiel pourraient compromettre la qualité finale des

appariements. Dès que les premiers tests ont été exécutés, nous avons vu que la dégradation de la qualité finale des appariements était bien présente (cf. l'on peut voir dans la figure 5.3).

Trois des quatre paires d'images sources présentent une diminution progressive du nombre d'appariements obtenus à la fin de la phase de propagation. La seule exception est la paire Fleur qui montre une petite augmentation de paires obtenues par rapport à la version séquentielle, due à notre avis au fait d'être une paire d'images avec des caractéristiques très particulières de *texture*, taille et encadrement. Les trois autres présentent clairement une réduction plus ou moins accentuée selon les caractéristiques des images.

Face à la réduction du nombre d'appariements vérifiée, nous avons du faire face à une question : comment comparer les temps d'exécution des deux versions si le résultat final obtenu n'est pas le même ? En principe, c'est une comparaison qui n'a pas de sens. Notre but lors de l'implantation de cette version était de vérifier si la chute de la qualité de l'appariement final se confirmerait. Nous avons vu qu'elle existe et peut être drastique dans certains cas. Malgré cela, nous croyons qu'une analyse du temps d'exécution parallèle en fonction de la qualité du résultat final peut être utile pour la suite car elle nous permettrait d'avoir une idée du gain réussi avec la mise en parallèle.

Ainsi, pour pouvoir "comparer" l'exécution parallèle de la propagation à la version originale en séquentiel, il nous a fallu établir une borne inférieure sur le nombre d'appariements obtenus à la fin de l'exécution. Pourquoi faire ? Parce que, pour chaque paire d'images, à partir d'un certain point les résultats sont sans intérêt (l'appariement final ne sert plus à rien). Nous avons donc établi arbitrairement que un appariement parallèle valide devrait au moins avoir 80% du nombre de paires de l'appariement final séquentiel. Ce pourcentage correspond visuellement à un appariement avec des petits trous noirs sur la surface des régions qui seraient normalement appariées, mais il est encore suffisant pour permettre à l'application de synthèse d'images de passer aux étapes suivantes et d'arriver au bout avec un peu moins de qualité du résultat final.

TAB. 5.1 – Comparaison : qualité de l'appariement final.

Nombre d'apps.	Séquentiel	Parallèle
Fleur	111538	118607 (16 proc.)
Maison	230399	183425 (5 proc.)
Rivulet	238746	188806 (20 proc.)
Tronc	30047	24414 (11 proc.)

Nous sommes alors arrivés au tableau comparatif 5.1. La règle de 80% n'a pas

pu être appliquée à la paire Fleur, car les images de cette paire présentent des dimensions réduites et la qualité visuelle a été compromise avant la qualité brute (nombre final d'appariements). À partir de 17 processeurs, il y a des tranches vides, sans paires de départ dans leur tas respectifs, et ainsi, même si le nombre d'appariements obtenus sur les autres tranches surpasse le nombre séquentiel, il y a des régions qui ne sont pas appariées.

Finalement, pour illustrer concrètement comment évolue l'algorithme de propagation basée sur le découpage des images en tranches, nous présentons sur la figure 5.4 les résultats intermédiaires de l'exécution sur trois processeurs de la paire d'images Fleur.

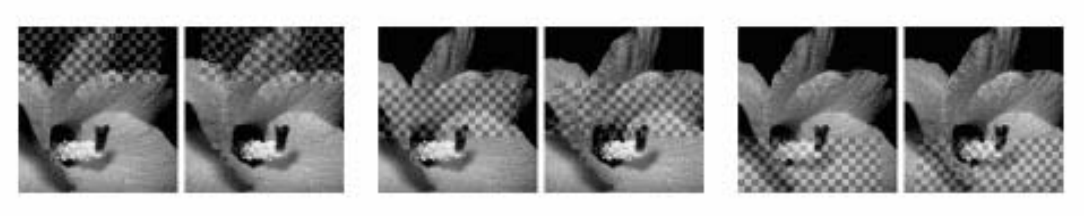


FIG. 5.4 – Exemple : résultat de la propagation sur 3 processeurs.

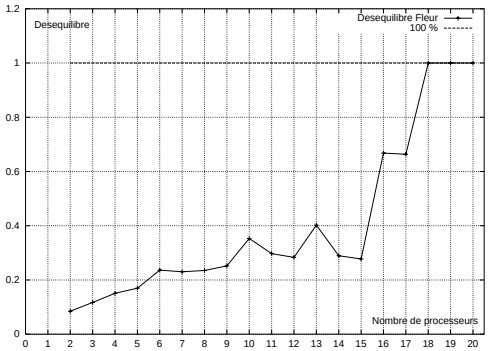
Dans cette figure, chaque paire d'images présente la région appariée par un des trois processeurs employés. Chaque processeur a reçu une tranche des images pour travailler. À gauche, le processeur a fait ses calculs sur la tranche du haut, au milieu c'est sur la tranche centrale que le processeur a évoluer et à droite c'est sur la tranche du bas. Les régions appariées sont indiquées par les zones quadrillées. On peut voir nettement sur chaque paire d'images que la région appariée ne dépasse jamais la surface de la tranche associée au processeur. La qualité de l'appariement final est donnée par l'union des trois ensembles des paires générés par chacun des nœuds.

Déséquilibre de charge

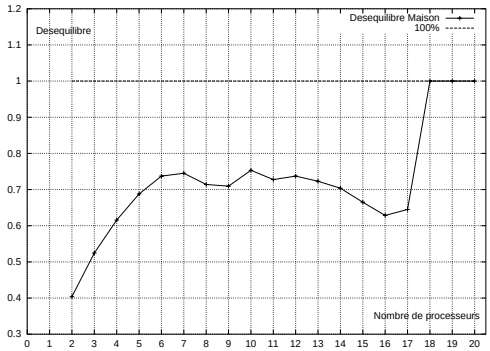
L'objectif de cette étude est de voir si un algorithme de croissance de régions démarré par une distribution aléatoire de paire de points d'intérêt sur la surface des images en utilisant une stratégie "meilleur d'abord" peut présenter des résultats convaincants lorsque l'on essaye de le paralléliser. Pour y arriver, il faut analyser tous les aspects du fonctionnement de l'application et l'un des plus importants est la façon dont le travail de calcul est distribué parmi les processeurs.

Dans cette première version parallèle de l'algorithme, nous n'avons pas mis de mécanismes de régulation de charge. Nous savions déjà qu'une telle application sera forcément irrégulière, mais il fallait confirmer cela. Dans la figure 5.5, nous pouvons

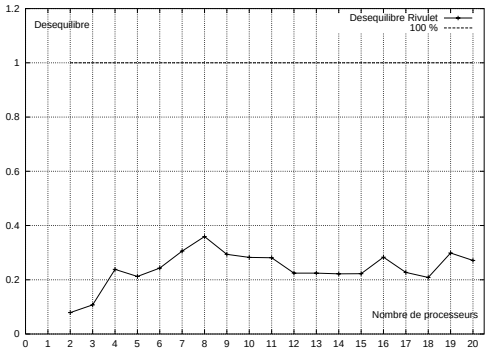
constater le comportement de la version parallèle en terme de déséquilibre de la distribution du travail (cf. définition présentée au chapitre 4, section 4.4.2) sur 2 jusqu'à 20 processeurs.



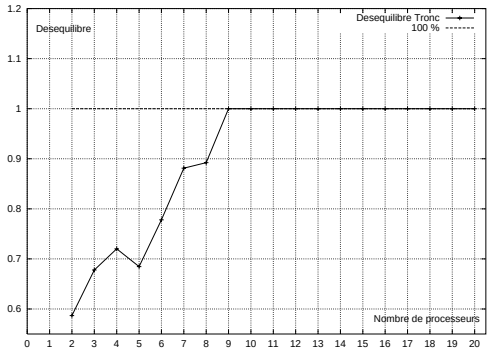
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

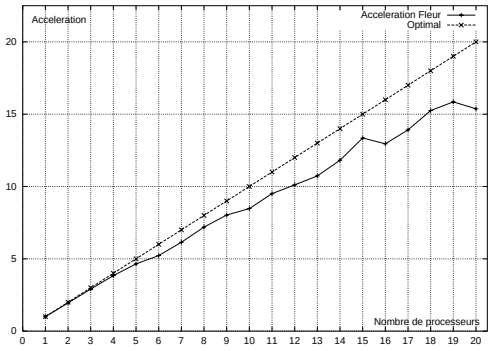
FIG. 5.5 – Version Tranches : Déséquilibre de charge

Il est possible de voir que, effectivement, la différence entre le processeur le plus lent et le plus rapide devient de plus en plus importante au fur et à mesure que le nombre de processeurs augmente et la taille des tranches diminue proportionnellement. L'exception qui confirme la règle est la paire Rivulet qui, en plus de larges dimensions, possède une rare distribution équilibrée des paires des points sur les deux images source. Cette paire, n'a pas été testée jusqu'aux limites de ses possibilités. Dans les trois autres, le déséquilibre de charge monte plus au moins vite selon les caractéristiques des images. Nous pouvons constater dans ces trois paires que, à partir d'un certain nombre de processeurs, le déséquilibre entre le processeur le plus rapide et le plus lent est de 100%. La raison pour cela réside dans le fait que le découpage des images a atteint sa limite. Il y a des tranches sans aucune paire de points de départ sur leurs surfaces et il y donc des processeurs qui ne travaillent

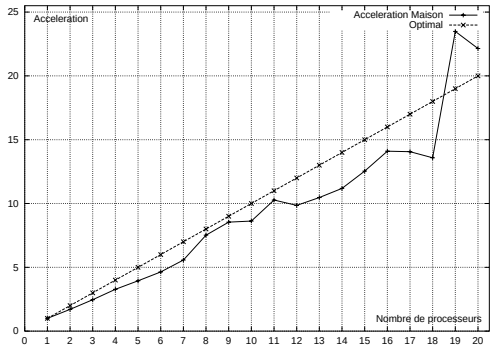
pas. Plus l'image est petite et plus elle a une distribution déséquilibrée des paires de points de départ, plus vite ce phénomène arrivera.

Accélération

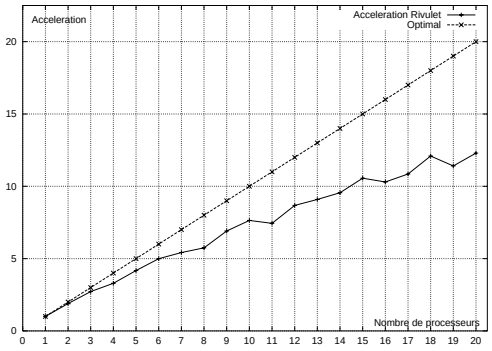
Les considérations sur la qualité finale des appariements et sur le déséquilibre de charge faites, nous pouvons maintenant analyser le gain obtenu en temps d'exécution pour chacun des nos exemples. Les quatre courbes d'accélération sont montrées sur la figure 5.6.



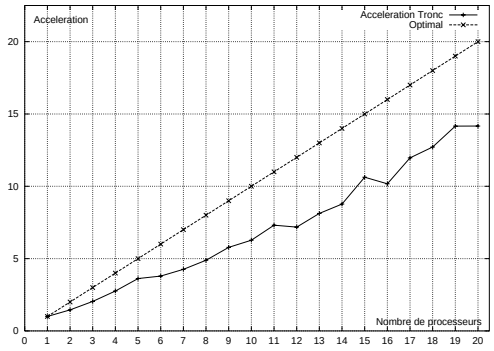
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

FIG. 5.6 – Version Tranches : Accélération

Les courbes montrent l'exécution de la version parallèle de l'application sur des configurations qui vont de deux à vingt processeurs. Les mesures pour chaque configuration sont obtenues à partir d'une moyenne de dix exécutions.

Dans un premier regard, il semblerait que les résultats sont très positifs, car les courbes pour chaque exemple sont très proches de l'optimale. Mais, hélas, cela ne correspond pas à la réalité. Il ne faut pas oublier qu'il existe une réduction très

accentuée dans trois des quatre exemples du nombre d'appariements réussis à la fin de l'étape de propagation. Par exemple, pour l'exécution de la paire Maison sur 17 processeurs le nombre d'appariements final obtenu est inférieur à 20% du nombre d'appariements trouvés sur l'exécution séquentielle. Ce phénomène se répète pour les paires Rivulet et Tronc avec moins d'intensité, il est vrai, mais le pertes sont aussi non négligeables. La seule exception est la paire Fleur qui jusqu'à 16 processeurs à gagné plus de paires correctement appariées à la fin de la propagation que dans la version séquentielle. À partir de 17 processeurs, les pertes sur la qualité visuelle sont trop importantes, même si le nombre absolu d'appariements de l'application est encore plus élevé que dans la version séquentielle.

La conclusion, donc, est que ces courbes mènent à une fausse idée de performance qui serait en plus obtenue avec très peu d'effort. Mais nous pouvons tout de même utiliser la limite de 80% du nombre d'appariements obtenus dans la version séquentielle pour monter un tableau comparatif entre les deux versions 5.2

TAB. 5.2 – Qualité de l'appariement final *versus* temps d'exécution.

Paire	Nb_{procs}	$Nbapp_{seq}$	$Nbapp_{//}$	T_{seq}	$T_{//}$
Fleur	16	111538	118607	5.11s	0.33s
Maison	5	230399	183425	19.75s	4.25s
Rivulet	20	238746	188806	17.44s	1.41s
Tronc	11	30047	24414	3.14s	0.43s

5.4.3 Limitations

Les résultats obtenus par la version parallèle en tranches indiquent que le plus grand problème lorsqu'on découpe les images est la perte des appariements de départ qui entraîne une diminution de la qualité finale d'appariement. Cela arrive parce que la propagation perd certains de ces germes et aussi parce qu'elle ne peut pas progresser en dehors des limites de chaque tranche associée à un processeur.

Les temps d'exécution obtenus en limitant la qualité minimale acceptée à 80% du nombre d'appariements trouvés en séquentiel ne doivent pas être considérés si l'on cherche à garder une qualité proche de la séquentielle (notre cas). Il faut donc, chercher une autre alternative pour la parallélisation de la propagation.

Malgré tout, les résultats de la paire Fleur sont dignes de note. Il y a eu une diminution non négligeable du temps d'exécution et la qualité a aussi augmenté d'à peu près 8%. Nous croyons que cette performance est une exception, car cette paire est la seule composée par des images qui ne sont pas des scènes d'extérieur. En plus, presque toute la surface des deux images est texturée. Pour ces deux raisons la paire représente une moindre difficulté pour l'algorithme de propagation.

5.5 Distribution des appariements de départ

Dans cette section nous allons expliquer comment a été implanté le principe de la deuxième hypothèse de parallélisation sur le code original de l'algorithme séquentiel. Nous allons ensuite présenter des résultats obtenus pour conclure sur une analyse des limitations de cette méthode.

5.5.1 Implantation

Pour cette deuxième version parallèle nous avons voulu changer l'axe de nos investigations. On ne découpe plus les images source, tous les processeurs auront le droit de laisser l'algorithme de propagation évoluer sur toute la surface des images. Dans cette étude nous allons tester les effets de la division de l'ensemble des paires de départ parmi les processeurs. Nous voulons savoir si cette division affectera de façon compromettante la stratégie "meilleur d'abord" qui est au cœur de l'algorithme de croissance de régions utilisé dans la phase d'appariement des images de l'application.

Comme nous avons vu dans le chapitre 4, l'algorithme de propagation est initialisé par un ensemble de paires de points d'intérêt qui ont été identifiées dans une phase précédente de l'application. Cet ensemble de paires est stocké dans une structure de type tas et les paires sont ordonnées selon l'ordre établi par la mesure ZNCC (cf. section 2.3.2).

Nous avons alors décidé de distribuer les éléments de ce tas parmi les processeurs. À la différence des images, on ne connaît pas cet ensemble de paires de départ avant l'exécution. Ainsi, il a été nécessaire de faire un schéma de partage des éléments du tas en temps d'exécution.

Pour faire le partage des éléments du tas de paires de départ nous avons choisi de réserver un des processeurs pour s'occuper de la distribution. Ce processeur sera responsable de l'envoi des paires de points aux autres processeurs à sa disposition. Tous les autres processeurs attendront l'arrivée d'un sous-ensemble de paires de départ pour démarrer la phase de propagation. La figure 5.7 illustre cette description.

L'implantation aura donc cinq phases :

- la préparation des sous-ensembles de paires à envoyer, faite par le processeur de distribution ;
- l'envoi des sous-ensembles de paires de points de départ aux autres processeurs ;
- la réception des paires par les processeurs de travail ;
- l'initialisation des tas locaux dans les processeurs de travail avec les paires de points reçus ;

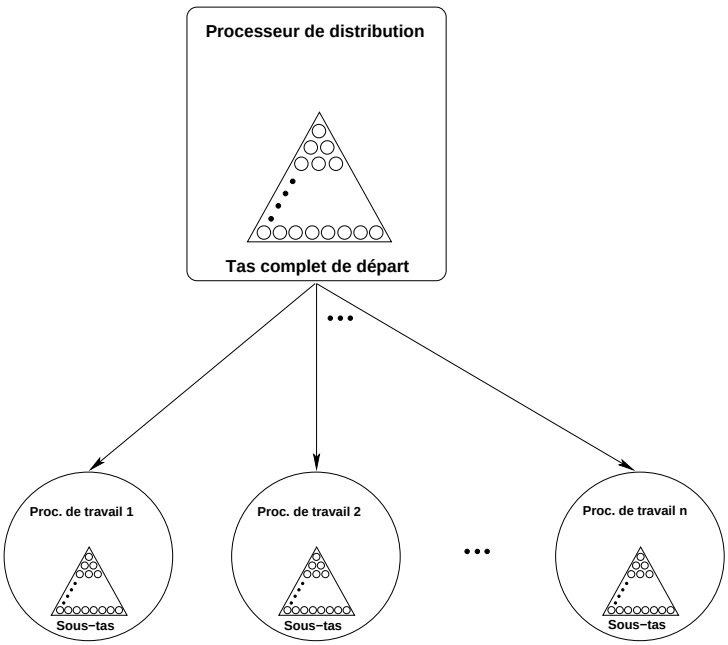


FIG. 5.7 – Version distribuée : schéma initial.

– le calcul des propagations locales sur chaque processeur de travail.

La phase de préparation du processeur de distribution du tas consiste en la construction de sous-ensembles du tas principal en obéissant à une règle de formation simple. On retire chaque germe du tas principal par ordre de valeur de la mesure ZNCC et on le place de la façon suivante : la paire avec la plus grande ZNCC est placée sur le tas qui sera envoyé au premier processeur, le deuxième élément sur le tas du deuxième processeur et ainsi de suite jusqu'au moment où tous les tas de chaque processeur ont reçu le premier élément. À ce moment là, on recommence le placement par le tas du premier processeur.

Nous avons choisi l'ordre par valeur de la mesure ZNCC pour que chaque sous-tas ait des appariements de départ avec une valeur élevée de cette mesure. Ainsi, l'on évite la concentration de paires avec une mesure ZNCC élevée sur un seul processeur. Nous croyons que cela est important car plus la mesure ZNCC d'un germe est élevée plus la propagation qui en résulte est étendue, donc plus de calcul.

La figure 5.8 montre un exemple du placement des six premiers éléments du tas principal sur une configuration hypothétique avec trois processeurs de travail selon la règle de formation décrite dans le paragraphe précédant.

Une fois terminée l'étape de préparation, les sous-ensembles ordonnés des paires de points de départ sont envoyés aux processeurs de travail. Chaque processeur de travail initialise alors son tas local et peut commencer l'exécution de l'algorithme

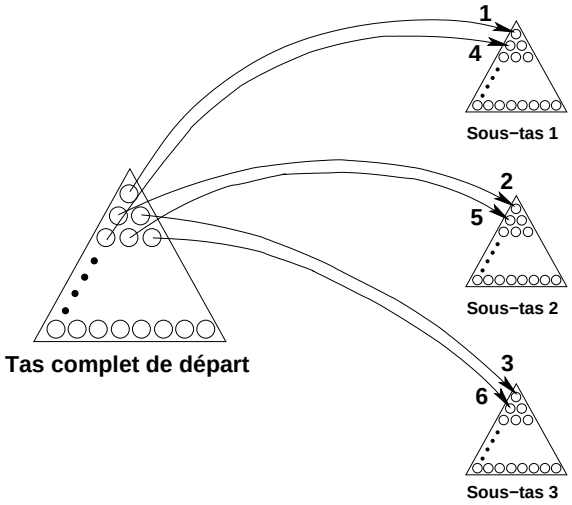


FIG. 5.8 – Division du tas principal : exemple.

de propagation sur son sous-ensemble de paire de points de départ.

5.5.2 Contrôle de la qualité

La description faite dans la section précédente explique comment est réalisée la distribution de données parmi les nœuds à disposition de l'application. Mais il y a une question qui n'a pas encore de réponse : que faire avec les appariements trouvés par les propagations locales de chaque algorithme ?

Dans cette version, nous n'avons pas de contrôle sur les limites de l'évolution de la propagation sur la surface des images. Chaque appariement de départ peut potentiellement déclencher une surface appariée de la taille de toute la surface des images. Ainsi, on ne peut pas simplement faire l'addition des ensembles d'appariements finals trouvés pour chaque processeur comme l'on a fait pour la version des tranches : il y aura sûrement des appariements répétés. Il est donc clair qu'il faut rajouter un mécanisme pour éliminer les résultats doubles trouvés par les processeurs de travail.

Pour résoudre ce problème nous avons opté, dans cette première version distribuée, par un mécanisme de centralisation, c'est-à-dire que nous allons renvoyer chaque nouvelle paire appariée à un processeur qui aura pour fonction de faire le filtrage des paires déjà appariées. Le choix naturel retombe sur le processeur de distribution car il peut faire le tri pendant le temps que les processeurs de travail calculent. Dans la figure 5.9, nous pouvons voir le schéma utilisé.

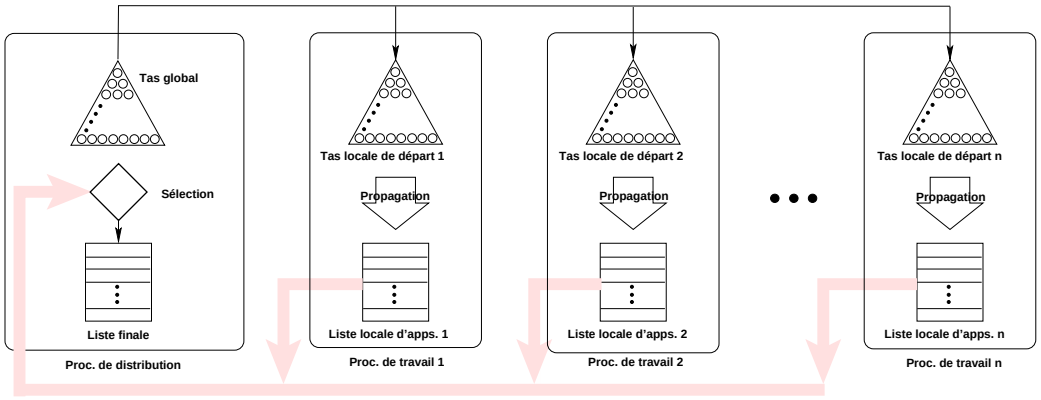


FIG. 5.9 – Schéma distribué avec centralisation.

Le tri de paires est fait d'une façon simple : pour chaque nouvelle paire arrivée, on fait une recherche rapide pour vérifier si elle est déjà présente dans le tas final des appariements. Le coût d'une telle recherche sur un tas peut être considéré très bas, mais nous ne savons pas d'avance le nombre de paires qui seront calculées plus d'une fois. L'expérimentation nous dira si cette stratégie est suffisante ou si elle va compromettre le débit du processeur de distribution.

D'un autre côté, l'utilisation du processeur de distribution pour centraliser l'arrivée des appariements calculés pour les processeurs de travail conduit à un autre avantage : le contrôle de la qualité de l'appariement final, c'est-à-dire le nombre d'appariements trouvés à la fin de la propagation par rapport à la version séquentielle de l'algorithme. Nous avons vu dans la version où la parallélisation est basée sur le découpage des images source en tranches qu'il ne sert à rien d'accélérer l'algorithme si à la fin le nombre de paires trouvées est trop réduit.

Dans le schéma adopté pour cette version distribuée, il est possible d'ajouter une condition d'arrêt pour la progression de la propagation. La vérification de cette condition peut être faite à chaque fois qu'une nouvelle paire appariée est ajoutée à l'ensemble final d'appariements dans le processeur de centralisation. Mais le problème réside dans le fait qu'il faut justement estimer un nombre d'appariements suffisant. Lorsqu'on dispose d'une version séquentielle qui peut être exécutée avant, il est possible de savoir jusqu'où la propagation peut arriver et utiliser une limite très proche de ce nombre pour forcer la propagation à avancer. Mais dans des conditions d'utilisation réelles, cela n'a pas de sens. Une alternative serait une estimation, en regardant les caractéristiques de l'image, de quel pourcentage de la surface des images pourrait être apparié. Le problème est que cette estimation reste très empirique et, par conséquent, très approximative aussi.

Malgré cette limitation, nous voulons tout de même tester l'approche distribuée

sur l'algorithme de propagation pour pouvoir continuer nos études. Ainsi, nous avons opté pour garder le choix de faire l'estimation du nombre d'appariements d'arrêt basée sur le nombre final de paires obtenues par la version séquentielle. Nous commencerons par fixer le nombre d'arrêt à 100% du nombre de paires calculées par la version séquentielle. Si, par les modification faites dans la logique de l'algorithme pour cette version parallèle, cette quantité de paires n'est pas atteinte nous allons refaire les tests à chaque fois avec une réduction de 1% dans le nombre final de paires qui doivent être trouvées jusqu'à trouver un nombre de paires qui puisse être atteint par la version parallèle de l'algorithme.

5.5.3 Taille des tampons

À la différence de l'envoi fait initialement par le processeur de distribution, le retour des paires calculées par les processeurs de travail est beaucoup plus volumineux. Le renvoi des paires calculées par les processeurs de travail vers le processeur de centralisation doit donc être fait en plusieurs étapes. La solution utilisée est l'adoption de tampons intermédiaires qui seront envoyés lorsque les processeurs de travail auront fini de les remplir.

Il est clair que la taille de ces tampons devient critique pour la bonne évolution de cette version parallèle. Nous ne pouvons pas oublier que nous travaillons sur une machine à mémoire distribuée et que la quantité de communications est déterminante pour une bonne efficacité du parallélisme. En plus, dans cette version les envois d'informations du nœud de centralisation vers les nœuds de travail n'arrivent qu'une fois au début. Après, c'est toujours dans l'autre sens. Ainsi, nous pouvons dire que la taille des tampons de renvoi détermine la granularité du parallélisme de cette version basée sur la centralisation des résultats. C'est-à-dire que des tampons trop petits augmenteront le coût de communication, par contre des tampons trop grands pourront surcharger la file d'attente du système associée au processeur de centralisation.

Il nous faudra donc faire une estimation théorique de la taille des tampons pour la confirmer ou non par le biais des expérimentations. Ainsi, il faut établir une façon de calculer le nombre d'envois en fonction de la taille des tampons :

$$Nb_{env} = \frac{Nbapp_{seq} \times t_{paquet} \times f_{redondance}}{t_{tampon}}$$

où :

- Nb_{env} : nombre d'envois global des processeurs de travail vers le processeur de centralisation ;
- $Nbapp_{seq}$: nombre final d'appariements calculées par la version séquentielle de l'algorithme ;

- t_{paquet} : taille du paquet nécessaire pour englober toutes les informations d'une paire appariée (4 octets) ;
- $f_{redondance}$: facteur de redondance. Estimations de combien de fois en moyenne une paire sera calculée (10 fois) ;
- t_{tampon} : estimation de la taille des tampons.

À partir de cette formule, nous avons donc élaboré le tableau 5.3 pour comparer le nombre d'envois nécessaires pour calculer un nombre moyen d'appariements. Nous allons travailler avec une moyenne d'appariement aux alentours de 130000 car le nombre final d'appariements de la version séquentielle de nos exemples varie de 30000 (Tronc) à 230000 (Rivulet). Les tailles proposées pour les tampons sont basées sur des mesures expérimentales faites sur la grappe que nous utilisons.

Il faut rappeler aussi qu'un tampon peut être envoyé sans être plein lorsqu'un processeur termine son travail alors que la limite établie de paires à trouver n'est pas encore atteinte par le processeur de centralisation. C'est-à-dire que la taille d'un tampon ne sera jamais altérée à cause d'un envoi où il y a moins de paires calculées. Ainsi, si le nombre d'envois nécessaire déterminé par la formule ci-dessus est une valeur fractionnaire, il doit toujours être arrondi vers l'entier supérieur le plus proche car nous aurons besoin d'envoyer un tampon entier au processeur de centralisation.

TAB. 5.3 – Comparaison : nombre d'envois en fonction de la taille des tampons.

Taille du tampon	Nb d'envois
1ko	5200
2 ko	2600
4 ko	1300
8 ko	650
16 ko	325
32 ko	163
64 ko	82
128 ko	41
256 ko	21
512 ko	11
1024 ko	6

La tranche de tailles entre 16 ko et 64 ko nous semble la plus indiquée pour deux raisons :

- les meilleurs taux de transmission de messages sur la grappe est obtenue sur des tampons de taille 32 ko ;
- le nombre de paires envoyées à la fois, entre 4000 pour 16 ko et presque 16000

pour 64 ko, semble raisonnable pour éviter des problèmes d'étranglement du processeur de centralisation.

5.5.4 Résultats

Dans cette section, nous allons présenter les résultats obtenus par l'exécution de notre deuxième version parallèle de l'application développée par [Lhu00]. Notre objectif est d'offrir au lecteur un échantillon des performances de cette version parallèle pour permettre une comparaison entre les résultats séquentiels et les prochaines versions parallèles qui seront proposées.

Pour chaque exemple, nous allons proposer des informations sur la qualité des appariements obtenus, l'accélération et le temps d'exécution. Un tableau comparatif entre le temps séquentiel et le temps parallèle sera montré aussi.

Les résultats présentés dans la suite ont été calculés sur une grappe de 225 processeurs Pentium III 733 Mhz sur Linux connectés via un réseau Ethernet 100. Les nœuds sont distribués en cinq groupes de 43 gérés par un commutateur du type *mesh* de 1 Go (pour plus de détails, voir http ://www-id.imag.fr/Grappes/icluster/UsersGuide.html). Chaque valeur présentée dans la suite est le résultat d'une moyenne arithmétique simple de dix exécutions en éliminant les valeurs extrêmes.

Avant de discuter des courbes d'accélération, nous allons montrer le tableau 5.4 qui contient le pourcentage atteint par notre version distribuée de la qualité de la version séquentielle pour chacun des paires d'images source. La méthode employée pour les établir est décrite dans section 5.5.2.

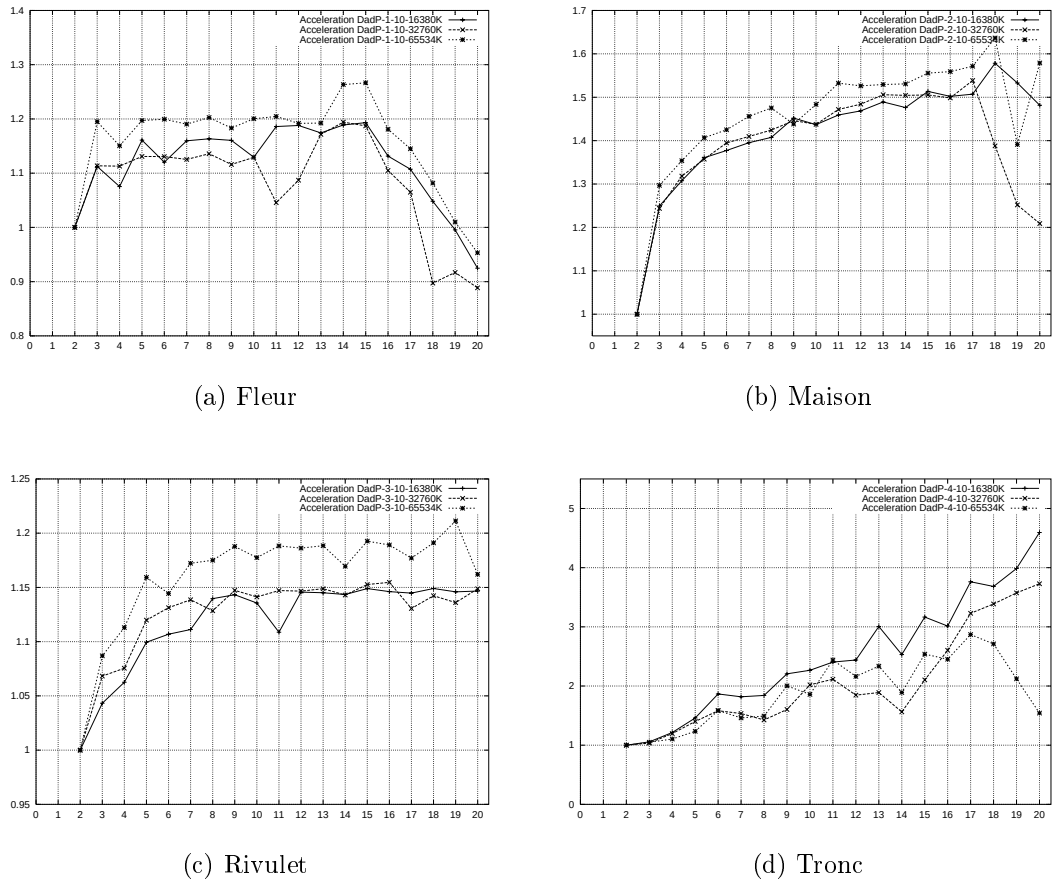
TAB. 5.4 – Qualité de l'appariement final : pourcentage de la version séquentielle.

Paire	% de la version Seq.
Fleur	94.14
Maison	97.67
Rivulet	96.34
Tronc	93.19

Les pourcentages atteints sont assez raisonnables. Avec de tels résultats, les étapes suivantes de l'application de synthèse d'images peuvent évoluer presque sans défauts perceptibles par rapport aux résultats de la version séquentielle. Ces pourcentages seront les mêmes pour toutes les versions parallèles qui dans la suite utiliseront le mécanisme de centralisation des appariements finals décrits pour cette première version distribuée.

Accélération

Les résultats obtenus par l'exécution de la version parallèle distribuée (version **DadP** - Distribuée avec division des Paires de départ) sont assez décevants. Le gain en accélération est presque inexistant dans certains cas. L'ensemble des courbes d'accélération pour les quatre paires d'images est présenté sur la figure 5.10.



(a) Fleur (b) Maison

(c) Rivulet (d) Tronc

FIG. 5.10 – Version Distribuée : Comparaison par taille des tampons

Nous présentons pour chaque paire d'images exemple l'ensemble des courbes d'accélération obtenues en exécutant la version distribuée avec trois tailles différentes de tampon. Les résultats confirment les estimations faites dans la section 5.5.3, la plus petite taille de tampons qui est supportée par l'application est 16 ko, si l'on utilise des tampons de taille inférieure à cette limite la queue de messages du processeur de centralisation sature. De l'autre côté, 64 ko est la taille maximale des tampons, car au-delà de cette taille, l'algorithme de sélection devient un goulot d'étranglement trop important arrivant parfois jusqu'au point de saturation.

Nous pouvons noter aussi que plus l'image est petite plus la taille de 16 ko

pour les tampons est adaptée. Ainsi que pour les grandes images la meilleure performance est réussie par les versions avec une taille de tampon de 64ko. Cela était attendu d'après nos observations précédentes.

De toute façon, nous sommes forcés de constater que la simple distribution des paires des points de départ parmi les processeurs de travail ne donne pas une parallélisation efficace. L'explication qui nous semble appropriée réside dans la redondance des calculs. Il apparaît qu'elle est le facteur clé des ces mauvaises performances, car chaque processeur de travail obtient une surface appariée qui correspond presque à celle de la version séquentielle d'après les images gérées par chacun d'entre eux.

Une autre caractéristique qui est intéressante à noter est la forte chute de performance sur trois des quatre exemples à partir des configurations aux alentours de 17 processeurs. La seule explication que nous pouvons présenter pour ce phénomène est que certaines images présentent un ensemble plus réduit de paires de départ réellement utiles pour la propagation. Par conséquence, après la division de l'ensemble des paires, quelques uns des processeurs de travail reçoivent des paires qui ne démarrent pas des propagations suffisamment étendues et ainsi le nombre final d'appariements qui l'on veut obtenir, et qui est défini à l'avance, n'est atteint qu'avec des calculs plus longs que le nécessaire.

Temps d'exécution

Si l'on regarde simplement les courbes d'accélération, on peut penser qu'il n'y a eu rien de positif dans l'implantation de cette version parallèle basée sur la distribution de paires de départ. Pour cette raison, nous avons décider de montrer aussi les courbes du temps d'exécution.

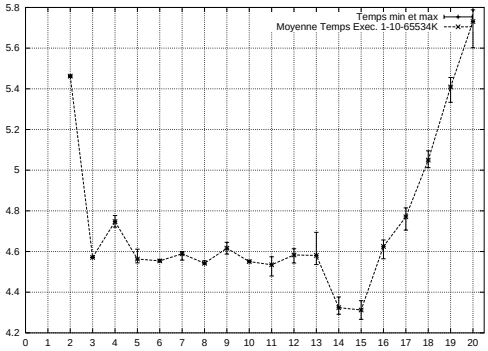
Le tableau 5.5 permet de comparer le temps parallèle pour chaque exemple avec le temps séquentiel. Le nombre de processeurs (Nb_{procs}) indiqué sur le tableau correspond au nombre de processeurs sur lequel le meilleur temps parallèle a été obtenu, le processeur de centralisation inclus. La grosse différence par rapport à la version de tranches est que désormais la qualité finale de l'appariement des images reste presque la même de la version séquentielle. Ces résultats indiquent que l'idée de la distribution n'est pas totalement mauvaise car sans y ajouter des raffinements il y a eu tout de même un gain de temps.

Les courbes de la figure 5.11 permettent de visualiser l'évolution de la réduction du temps d'exécution pour les quatre paires d'images exemples. En analysant ces courbes, l'on peut voir que l'approche adoptée n'arrive pas à maîtriser l'irrégularité du problème traité. Le comportement de la version distribuée de l'algorithme propagation n'est pas suffisamment raffinée pour éviter certaines distorsions dans

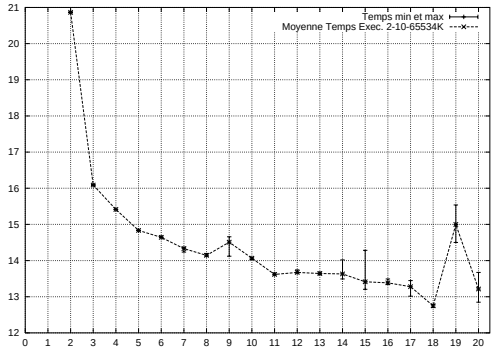
TAB. 5.5 – Comparaison : temps d'exécution.

Paire	Nb_{procs}	T_{seq}	$T_{//-DadP}$
Fleur	15	5.11s	4.31s
Maison	18	19.75s	12.75s
Rivulet	19	17.44s	14.76s
Tronc	17	3.14s	0.88s

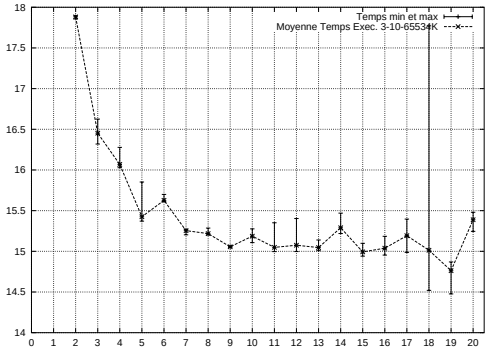
les courbes de temps d'exécutions. Sur les images de dimensions plus grosses (Maison et Rivulet), les courbes se montrent plus régulières. Les images plus réduites en taille (Fleur et Tronc)sont plus problématiques puisque ces courbes présentent des variations plus marquées. Nous croyons que ce genre de variation est due au petit nombre d'appariements de départ dans les deux cas, ce qui créé un déséquilibre de charge beaucoup plus important. De reste, les quatre cas présent une augmentation des temps d'exécution lorsque le nombre de processeurs employés devient trop élevé pour les dimensions des images.



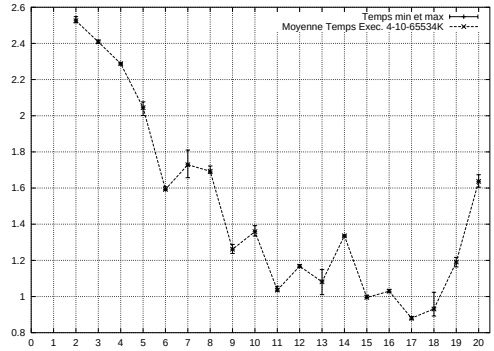
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

FIG. 5.11 – Version Distribuée : Temps d'exécution

5.5.5 Limitations

Le bilan de l'implantation de la version distribuée est très positif malgré le manque de performance dans les résultats obtenus. Nous avons pu réduire le temps d'exécution de l'algorithme pour les quatre paires d'images source (lesquelles présentent des caractéristiques bien différentes) tout en gardant une qualité de l'appariement final supérieure à 95% de la qualité obtenue pour la version séquentielle pour l'ensemble des paires d'images.

Néanmoins le plus important après l'implantation de cette version parallèle de l'algorithme sont les limitations constatées, lesquelles nous seront très utiles pour la suite de nos recherches.

La première observation qui nous semble importante est la constatation que l'altération dans l'ordre d'utilisation des paires des points de départ par l'algorithme de propagation a un coût. Ce coût apparaît sans perte en qualité de l'appariement final par rapport à la qualité de la version séquentielle et il est le résultat de la formation des sous-ensembles des paires de départ envoyées aux processeurs de travail. Nous avons réussi à laisser cette perte dans des niveaux très réduits, mais elle existe toujours.

La deuxième limitation importante est le manque de performance présenté par cette version parallèle. La distribution des paires du tas original par ordre de priorité parmi les processeurs de travail en est la cause directe. Elle a été choisie car nous avons pensé qu'en donnant à tous les processeurs de travail des paires de haute valeur, la charge de travail serait plus équilibrée. Nous avions raison, mais un des effets secondaires de cette approche à été l'énorme redondance de calcul présentée par les processeurs de travail. Nous l'attendions, car nous savions que l'absence de limites de progression sur la surface des images pour l'algorithme de propagation permettrait l'appariement de régions superposées par les différents processeurs de travail. Mais elle s'est montrée beaucoup plus coûteuse. Nous croyons que cette redondance de calcul si importante est causée car plusieurs des paires avec les priorités les plus élevées se trouvent assez souvent très proches. Ainsi, la propagation démarrerait presque au même endroit sur plusieurs processeurs.

5.6 Bilan

Dans ce chapitre nous avons présenté au lecteur nos deux premières versions parallèles pour l'algorithme de propagation proposé dans le travail de [Lhu00] : la première basée sur le découpage des images sources, la deuxième basée dur la distribution des appariements de départ. Notre objectif était de vérifier les avantages et les inconvénients de chacune des approches. Nous n'attendions pas des performances pointues, car le problème est trop complexe et ces deux solutions proposées étaient

plutôt naïves. Néanmoins, l'implantation de ces deux versions parallèles nous a apporté plusieurs informations sur le comportement de l'algorithme. Des informations concernant les limitations déjà citées auparavant et qui contribueront de façon définitive dans la suite du travail. Pour conclure, on termine sur les deux aspects à retenir pour la suite : le découpage en tranches mène à problèmes avec la qualité ; la distribution des appariements de départ présente trop de redondance de calcul.

Chapitre 6

Approche mixte

Sommaire

6.1 Introduction

Dans le chapitre 5, nous avons vu les résultats de l'implantation des premières versions parallèles de l'algorithme de propagation proposé dans le travail de [Lhu00]. D'après ces versions préliminaires, nous avons eu des renseignements intéressants sur le comportement de l'algorithme.

Par exemple, lorsque nous avons essayé le découpage fixe des images, nous avons constaté que l'algorithme en était très sensible et le résultat était une perte rapide de la qualité finale de l'appariement sur trois de quatre paires d'images.

Nous avons alors changé d'approche pour proposer une version sur laquelle notre objectif était d'abord de maîtriser la qualité finale de l'appariement obtenue. Notre idée était basée sur la distinction entre deux types de nœuds : un pour distribuer le travail parmi les autres et recueillir les résultats de leurs calculs, et les autres qui seraient responsables pour calculer la propagation sur un sous-ensemble des appariements de départ.

Cette dernière version nous a permis d'étudier les réactions de l'algorithme de propagation lorsqu'on transformait sa stratégie "meilleur d'abord" globale dans une stratégie "meilleur d'abord" locale. Si d'un côté, nous avons pu constater une légère perte de la qualité de l'appariement finale due certainement à l'altération de l'ordre globale d'utilisation des paires de départ, de l'autre nous avons pu aussi vérifier une réduction (parfois très petite, il est vrai) du temps d'exécution sur les quatre paires exemples. Mais le plus important a été l'identification d'une possible raison pour ces faibles réductions du temps d'exécution : la redondance présente dans les calculs des nœuds de travail.

Nous allons donc proposer dans les sections qui suivent quelques variations sur la version parallèle basée sur la distribution des appariements de départ pour essayer d'en augmenter les performances en réduisant le problème de la redondance de calcul. Pour une meilleure compréhension du texte, nous allons garder la mise en forme utilisée dans le chapitre 5, c'est-à-dire : justifier les choix proposés, présenter les aspects concernant l'implantation, montrer les résultats obtenus et discuter les limitations vérifiées.

6.2 Vers une approche mixte

Dans le chapitre précédent, nous avons vu que chacune des approches de parallélisation proposées a des avantages et des inconvénients. La version des tranches offre un contrôle total de la progression de la parallélisation, mais elle n'assure pas une bonne qualité de l'appariement finale à cause justement des

restrictions imposées par l'existence des tranches. D'un autre côté, la version distribuée est très efficace pour maîtriser la qualité finale de l'appariement des images source, mais n'offre que des temps parallèles très médiocres à cause de sa limitation pour contrôler l'évolution de la propagation sur la surface des deux images.

Les deux approches ne sont donc pas satisfaisantes séparément. En fait, l'idéal serait de trouver un compromis entre les deux propositions de façon à profiter des avantages de chacune tout en supprimant leurs limitations. Si cette conclusion semble évidente, la façon de l'implanter par contre ne l'est guère. Le principal problème pour y arriver est l'élimination (ou presque) de la redondance de calcul en compromettant un minimum le nombre final d'appariements obtenus à la fin de la phase d'application de l'algorithme de propagation. Cela est d'autant plus compliqué si l'on rappelle que la redondance de calcul est justement un des principes de base de la version séquentielle de l'algorithme. Il nous reste donc à chercher un point d'équilibre dans le rapport qualité / redondance dans nos prochaines versions.

6.3 Distribution géographique

La troisième version parallèle de l'algorithme de propagation proposé dans [Lhu00] est notre premier pas dans la direction établie précédemment, c'est-à-dire : créer une version qui mélange les caractéristique positives de deux premières.

L'idée centrale de cette version nous est venue lors d'une analyse faite sur la façon de distribuer les points de départ dans la première version distribuée. La distribution y était faite avec le but de créer de sous-ensembles équilibrés du tas initial. Ainsi, il y avait un ordre établi de formation des sous-tas : la première paire retirée du tas principal était attribuée au premier au tas associé au premier processeur de travail, la deuxième paire était attribuée au deuxième processeur et ainsi de suite jusqu'à faire une première attribution d'une paire du tas principal à tous les processeurs de travail. À ce moment, on recommençait à faire les attributions par le premier processeur de travail jusqu'à l'épuisement du tas principal (figure 5.8).

En effet, les sous-tas crées par cette façon de distribuer les appariements de départ étaient bien équilibrés en termes de mesure ZNCC, par contre il n'y avait aucun souci par rapport à la localité de chaque paire sur la surface des images. Nous nous sommes confrontés alors au problème de la mauvaise distribution géographique des appariements de départ (figure 6.1).

Pour mieux comprendre ce problème, il faut savoir que très souvent il arrive que les premiers paires de points de départ du tas initial se localisent très proches les uns des autres. Cela est parfaitement possible car la mesure ZNCC qui détermine l'ordre de paires dans les tas global est basée sur les caractéristiques de couleur et texture de la surface des images. Ainsi, des paires de points d'intérêt localisées sur

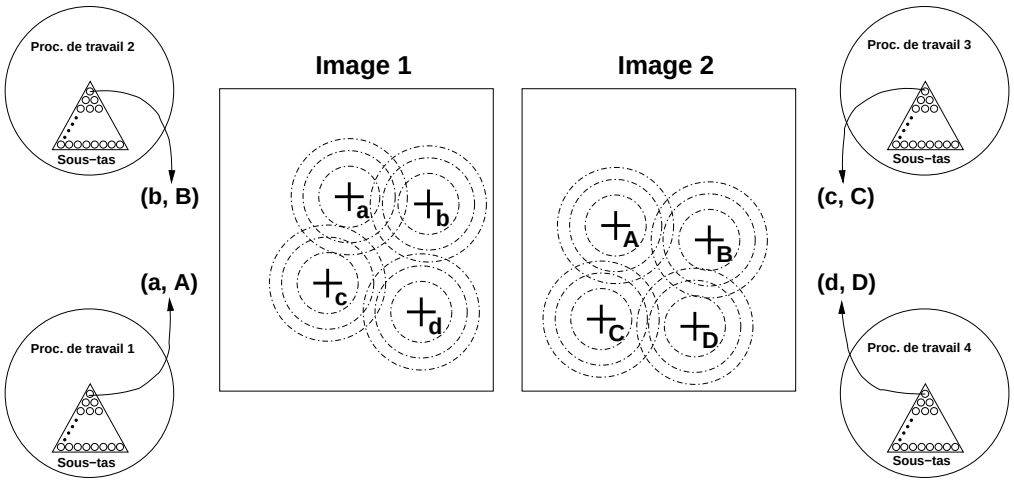


FIG. 6.1 – Problème de la mauvaise distribution des appariements de départ.

une même région des images ont des fortes chances d'avoir une mesure ZNCC très proche. Pour cette raison, une configuration comme celle montrée dans la figure 6.1 n'est pas rare. Dans la représentation schématique de la figure, ou l'exécution en parallèle est faite sur quatre nœuds, les quatre paires de départ de chaque sous-tas sont localisés très à proximité les uns des autres. Cela fait que la propagation de chacun (représenté par des cercles concentriques dans la figure) soit rapidement superposée à la propagation des autres. Nous sommes donc confrontés à une situation où la redondance de calcul des processeurs de travail apparaît très tôt et notre objectif initial est d'empêcher cela.

6.3.1 Implantation

La résolution du problème de la mauvaise distribution des appariements de départ passe donc par une nouvelle stratégie de leur distribution aux processeurs de travail. Il faut la penser de façon à permettre un début de progression de l'algorithme de propagation bien isolé sur chaque processeur de travail comme l'on peut voir sur la figure 6.2.

Pour y arriver, il nous faut donc créer une façon de distribuer les paires de départ basée sur leur localité sur la surface des images. On retombe alors sur le problème du découpage des images en zones.

Nous avons donc utilisé le concept de tranches employé dans la première version parallèle de l'algorithme. Mais cette fois ci, les tranches ne seront employées que dans la phase de distribution des appariements de départ. Au moment de l'exécution de la phase de propagation elle même, les tranches n'empêcheront pas la progression

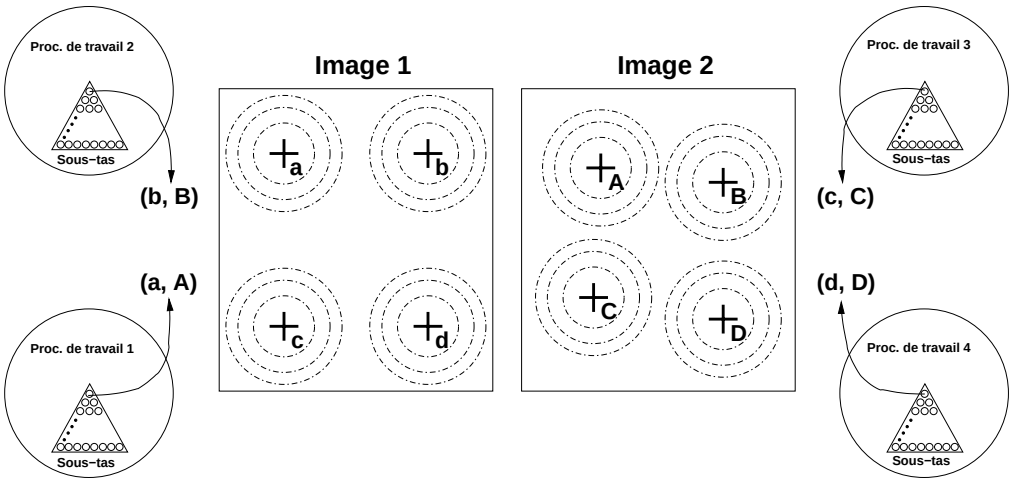


FIG. 6.2 – Solution possible pour le problème de la localité des appariements de départ.

de la propagation en dehors des leurs bordes. C'est ainsi car nous voulons éviter les pertes occasionnées par le problème des bordes déjà vérifiées auparavant.

Dans le nouveau schéma de formation des sous-ensembles du tas de départ, le nombre de tranches sur les images source équivaut au nombre de processeurs de travail existants. Chaque tranche a son propre sous-tas de paires de départ et ils sont formés sur le processeur de distribution de la façon suivante (voir aussi la figure 6.3) :

1. retirer une paire du tas global ;
2. vérifier à quelle tranche elle appartient ;
3. l'associer au tas qui correspond à la tranche sur laquelle la paire se trouve.

Une fois terminée la création des nouveaux sous-tas, ils sont alors envoyés aux processeurs de travail. Le cours d'exécution suit alors le même chemin de la version distribuée présentée au chapitre 5. Les processeurs de travail calculent des appariements jusqu'au remplissage d'un tampon de taille connue pour après l'envoyer au processeur de centralisation. Celui-ci fait un tri pour éliminer les appariements déjà présents dans l'ensemble final de paires appariées. Nous avons gardé ce mécanisme de centralisation car il s'est montré efficace pour résoudre le problème de la qualité de l'appariement final des images source. Ainsi, nous restons pour cette version sur les mêmes pourcentages de la qualité de la version séquentielle obtenus lors de la deuxième version parallèle (première version distribuée).

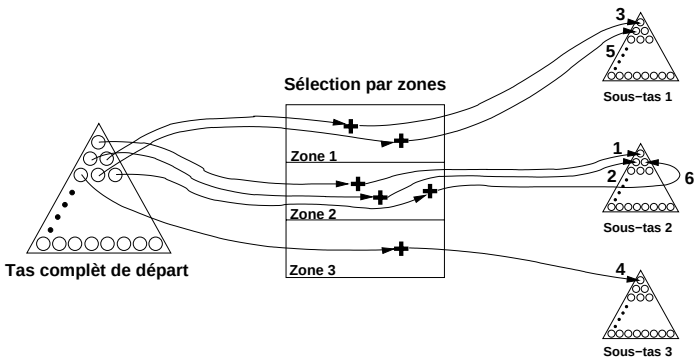


FIG. 6.3 – Division du tas pour la version de la distribution géographique.

6.3.2 Résultats

Dans cette section, nous allons présenter les résultats obtenus par l'exécution de notre troisième version parallèle de l'application développée par [Lhu00]. Notre objectif est d'offrir au lecteur un échantillon des performances de cette version parallèle pour permettre une comparaison aux résultats séquentiels et aux autres versions parallèles déjà proposées ou à proposer.

Pour chaque exemple, nous allons proposer des informations sur l'accélération et le temps d'exécution obtenus. Une comparaison avec la première version distribuée sera aussi présentée. Nous finirons sur un tableau comparatif entre le temps séquentiel et le temps parallèle atteint pour cette version.

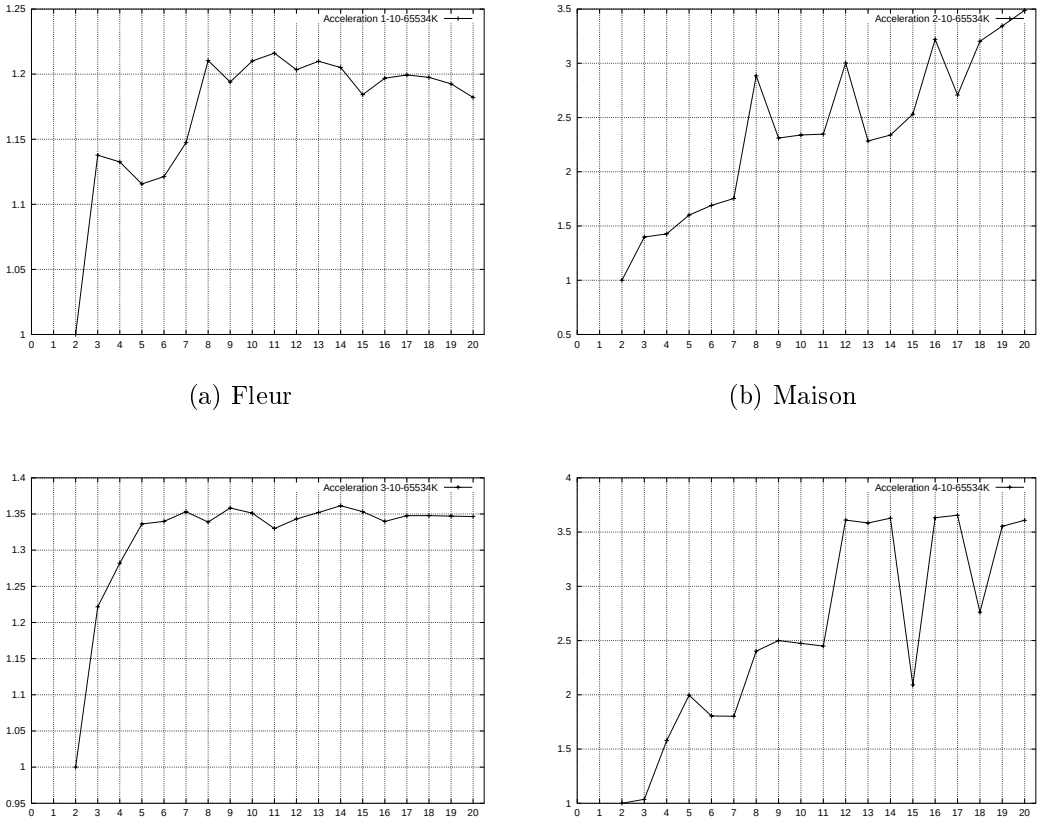
Les résultats présentés dans la suite ont été calculés sur une grappe de 225 processeurs Pentium III 733 Mhz sur Linux connectés via un réseau Ethernet 100. Les nœuds sont distribués en cinq groupes de 43 gérés par un commutateur du type *mesh* de 1 Go (pour plus de détails, regarder dans http ://www-id.imag.fr/Grappes/icluster/UsersGuide.html). Chaque valeur présentée dans la suite est le résultat d'une moyenne arithmétique simple de dix exécutions en éliminant les valeurs extrêmes.

Accélération

Nous commençons l'analyse des résultats obtenus pour la version géographique (version **DadT** - Distribuée avec découpage en Tranches) par les courbes d'accélération. Contre toute attente de notre part, les performances restent très faibles. Pour certains exemples comme Fleur et Rivulet, l'accélération est presque inexistante. Nous avons eu une amélioration de la performance sur la paire d'images Maison et une légère chute sur la paire Tronc (les courbes sont montrées dans la figure 6.4).

De plus, nous pouvons constater de très grosses oscillations dans la courbe d'accélération pour trois de quatre exemples. À notre avis cela ne configure pas des bizarreries dues à une mauvaise implantation car elles sont le résultat d'un ensemble d'exécutions dont les écart-types pour la mesure du temps d'exécution restent dans la normalité.

Les oscillations sont plutôt dues à la stratégie adoptée. Explication : pour chaque nouveau processeur introduit dans la configuration d'exécution de l'application, une nouvelle tranche est créée. L'introduction d'une nouvelle tranche altère carrément la distribution des appariements de départ sur l'ensemble des sous-tas associés à chaque processeur de travail. Il se peut que, dans cette nouvelle distribution, des paires très proches qui avant étaient sur une même tranche, sont désormais localisées sur des tranches voisines. Ainsi, le problème de la mauvaise distribution peut réapparaître en force à chaque nouvelle distribution des appariements de départ et la redondance de calcul est à nouveau présente.



(a) Fleur

(b) Maison

(c) Rivulet

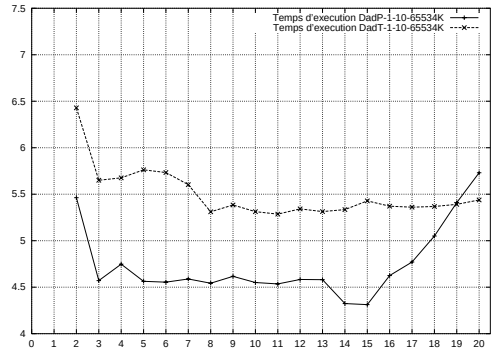
(d) Tronc

FIG. 6.4 – Version Distribution Géographique : Accélération

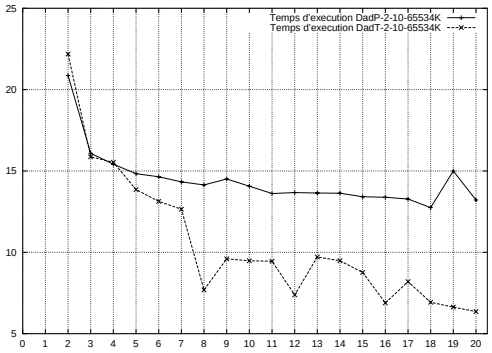
Temps d'exécution

Comme dans la première version distribuée, l'analyse des courbes d'accélération est très décevante. Une constatation s'y impose : l'utilisation des ressources de calcul de la machine est loin d'être optimisée. Encore une fois, pour pouvoir trouver des aspects positifs dans l'implantation de la version distribuée géographique, il faut regarder attentivement les courbes du temps d'exécution.

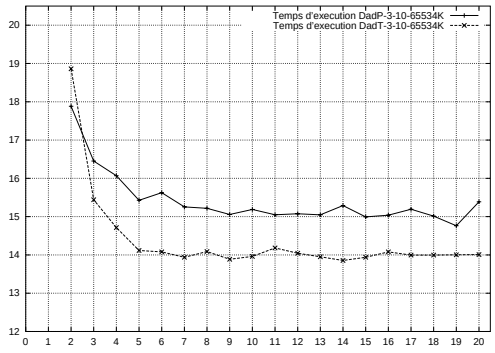
Nous avons choisi de montrer les courbes du temps d'exécution de la version géographique en comparaison avec les courbes de la première version distribuée (figure 6.5). Ainsi, le lecteur pourra visualiser le petit gain en temps d'exécution en faveur de la version géographique existant sur la plupart des configurations de machines employées.



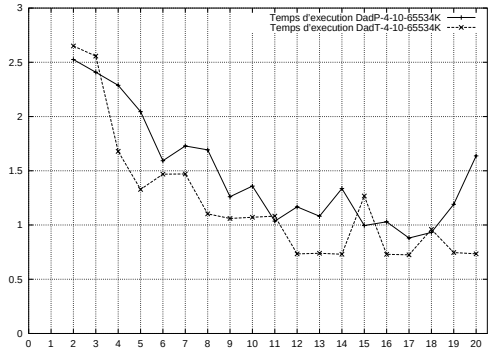
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

FIG. 6.5 – Comparaison du temps d'exécution : version distribuée (DadP) x version distribuée géographique (DadT).

Pour les paires contenant des grosses images (Maison et Rivulet), la constatation d'une réduction du temps d'exécution est nette. La paire Tronc présente une varia-

tion qui pourrait être considérée comme troublante si l'on oublie que les images qui composent la paire sont de petite taille et en plus avec un très bas pourcentage de réussite dans l'appariement. Les temps d'exécution deviennent alors trop petits et par conséquence la moindre variation introduite à cause des modifications dans la distribution des paires de départ amène à des croisements des courbes. Finalement la peur Fleur présente des temps d'exécutions toujours plus lents que les temps obtenus dans la version première version distribuée. Nous n'avons pas la moindre idée de la cause de ce comportement. Nous avons relancé l'exécution de l'application sur cette paire d'images à plusieurs reprises et les résultats ont été toujours les mêmes. De toute façon, les temps parallèles sur cet exemple n'arrivent jamais à être plus bas que le temps séquentiel. Le lecteur pourrait se demander comment est possible une telle situation.

L'explication passe par l'altération ajoutée au code depuis l'implantation de la première version distribuée. Avec la création de deux types de nœuds, l'exécution sur un seul processeur est devenue impossible. Le premier temps mesuré est le temps d'exécution avec un processeur de distribution/centralisation et un processeur de travail. Et comme l'on rajoute plusieurs mécanismes pour organiser l'envoi et la réception des appariements en plus du temps de communication lequel n'existe pas sur la version séquentielle, le temps d'exécution de départ (celui qui sert de référence pour le calcul de l'accélération) croît. Ainsi, nous avons eu une infime accélération par rapport au temps sur deux processeurs dans la paire Fleur lorsque nos exécutons sur trois ou plus processeurs, mais le temps d'exécution reste plus grand que le temps séquentiel car l'accélération n'a pas été suffisante.

Le tableau 6.1 confirme le problème décrit ci-dessus sur la paire Fleur. Il présente une comparaison entre les temps d'exécution obtenus par la version distribuée géographique (DadT), la première version distribuée (DadP) et les temps de l'exécution séquentielle pour les quatre paires d'images. La colonne Nb_{procs} contient le nombre de processeurs sur lequel le temps de la version distribuée géographique (DadT) a été obtenu.

TAB. 6.1 – Comparaison : temps d'exécution version DadT et les autres.

Paire	Nb_{procs}	T_{seq}	$T_{//-DadP}$	$T_{//-DadT}$
Fleur	11	5.11s	4.31s	5.29s
Maison	20	19.75s	12.75s	6.35s
Rivulet	14	17.44s	14.76s	13.85s
Tronc	17	3.14s	0.88s	0.72s

6.3.3 Limitations

La limitation la plus importante de cet nouvelle version parallèle de l'algorithme de propagation est claire : elle n'offre pas des performances convaincantes.

Le manque des performances, malgré une petite amélioration par rapport à la première version distribuée, nous indique que la simple distribution géographique des points de départ n'est pas suffisante pour éliminer le problème de la redondance de calcul. En fait, il nous semble, après l'implantation de la version géographique, qu'il faut trouver un moyen de contrôler les calculs redondants **pendant** l'exécution de l'algorithme de propagation.

De positif, il y a la consolidation du mécanisme garant d'une qualité finale de l'appariement très proche de la qualité séquentielle. Nous avons modifié sensiblement la façon de distribuer les paires de départ parmi les processeurs de travail, ce qui rajoute des altérations importantes dans l'ordre globale de travail avec les paires de départ pour la stratégie "meilleur d'abord", mais la perte de qualité vérifiée est restée dans les mêmes valeurs de la version précédente.

6.4 Distribution avec tranches souples

Basée dans les versions antérieures, nous avons pu constater que tenter de maîtriser la redondance de calcul seulement en jouant avec la distribution des appariements de départ reste une approche assez limitée. Ainsi, nous avons choisi de chercher une solution pour le problème de la redondance de calcul des nœuds de travail pendant l'exécution de l'algorithme de propagation.

Dans cette section nous prétendons proposer une modification dans l'algorithme de propagation avec le but de réduire l'expansion de la croissance des régions sur la surface des images source. Dans la version distribuée géographique, nous avons trouvé une solution pour le problème de la mauvaise distribution des appariements de départ en rajoutant le concept de tranches ouvertes, c'est-à-dire des tranches qui ne pouvaient pas empêcher l'expansion de la propagation en dehors de ces limites. Elles servaient uniquement à délimiter une région à partir de laquelle les appariements de départ étaient distribués aux processeurs de travail. En utilisant cette approche nous avons pu éviter les situations ou plusieurs processeurs de travail démarraient leurs propagations par des paires de départs localisées trop proches sur la surface des images.

Mais nous avons constaté que cette approche présente le problème diamétralement opposé au problème de la version basée uniquement sur les tranches fermées. Dans la première version parallèle basée sur les tranches, l'impossibilité de l'expansion de la progression en dehors des limites des tranches a éliminé

complètement l'existence de redondance de calcul. Par contre, nous avons vu qu'un tel choix compromet l'obtention d'une qualité finale de l'appariement au fur et à mesure que le nombre de processeurs croît. Pour la version géographique, le problème de la qualité n'existe pas, mais l'algorithme de propagation peut avancer sur toute la surface des images. Cette caractéristique a pour résultat l'inévitable redondance de calcul.

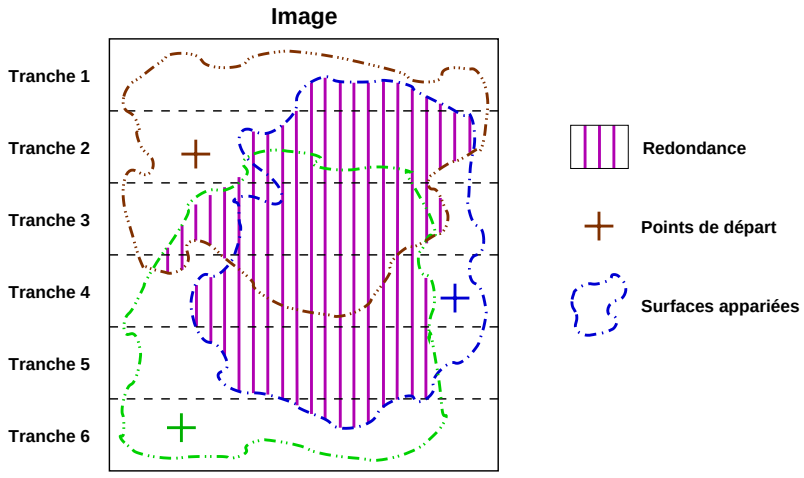


FIG. 6.6 – Redondance : surfaces calculés pour plus d'un processeur

Dans la figure 6.6, nous avons un exemple du problème de la redondance lorsqu'on permet que la propagation avance en dehors des limites des tranches. Sur la figure il n'y a qu'une image pour simplifier, mais dans la réalité la situation est analogue dans l'autre image de la paire source. Les surfaces appariées sont délimitées par les lignes pontillées. La surface rayée correspond aux régions de l'image qui ont été appariées plus d'une fois (redondance de calcul). Nous sommes donc dans une situation dans laquelle nous avons réussi à retarder l'apparition de la redondance de calcul, mais cela n'est pas suffisant, car elle reste encore très présente. Nous allons donc chercher une solution pour ce problème dans la suite.

6.4.1 Implantation

La résolution du problème de la redondance de calcul passe par l'addition d'un mécanisme de contrôle de l'évolution de la propagation sur la surface des images. Il faut que ce mécanisme de contrôle soit actif **pendant** l'exécution de la phase de propagation.

Nous croyons que jusqu'ici les modifications ajoutées à l'algorithme original ont été utiles mais insuffisantes. Les performances restent médiocres et la subtilisation

des ressources trop importante. Par contre les idées développées ont créé une base sur laquelle nous pouvons nous appuyer pour chercher une solution pour la question de la redondance de calcul. Nous allons garder donc le schéma de distribution des points de départ orienté par le critère de localité et nous allons utiliser un moyen terme entre les tranches fermées et les tranches ouvertes : les tranches souples.

L'idée est de garder le découpage en tranches utilisée dans l'attribution des appariements de départ aux processeurs de travail pour délimiter l'avance de la propagation. Mais nous avons déjà vu que stopper la propagation aux limites des tranches résulte dans des pertes dans la qualité finale de l'appariement. Alors, nous allons essayer de permettre l'expansion de la propagation en dehors des limites des tranches, mais seulement un peu. Explication : la propagation pourra avancer sur une surface correspondant à un pourcentage choisi de la surface de la tranche en question. Le pourcentage ne pourra pas être trop important sinon l'on revient au problème des tranches ouvertes, ni trop petit sinon la propagation n'atteindra pas le niveau de qualité désiré.

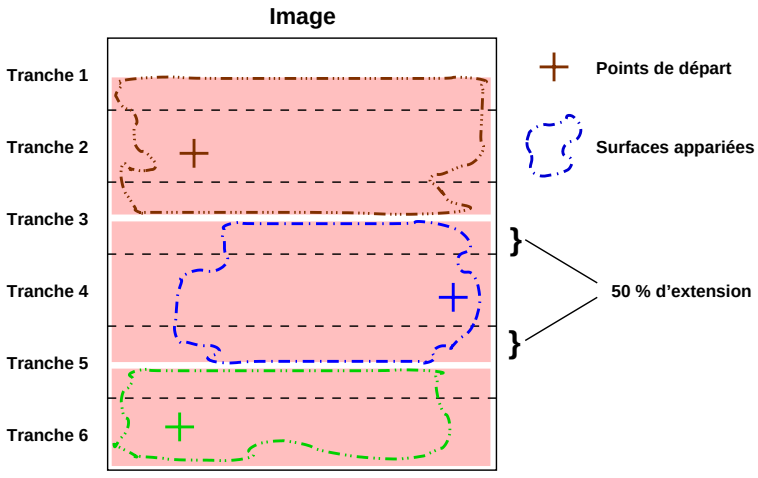


FIG. 6.7 – Avantages dans l'utilisation des tranches souples.

La figure 6.7 montre un exemple où le pourcentage d'extension de la propagation a été définie comme étant 50% de la hauteur d'une tranche donnée. L'extension permet la progression de la propagation dans les parties supérieures et inférieures des tranches sauf lorsqu'une tranche est à l'une des extrémités des images. Dans l'exemple, nous avons mis uniquement l'extension des tranches dans lesquelles il y avait un point de départ (les croix) pour simplifier le dessin. La situation montrée dans l'exemple est idéale, puisque il n'y a pas de calcul redondant, mais cela n'est pas le cas dans la plupart des images car difficilement une tranche ne possède pas un point de départ dans ces limites. Nous n'avons donc pas éliminé la redondance, car des tranches voisines auront toujours une intersection de leurs

zones d'expansion. Nous avons simplement limité son évolution sur la surface des images selon nos besoins. Ainsi certaines régions d'une tranche qui ne peuvent être atteintes par les propagations démarrées avec ses paires de départ, peuvent l'être par des propagations initiées sur une tranche voisine.

6.4.2 Résultats

Dans cette section, nous allons présenter les résultats obtenus par l'exécution de la quatrième version parallèle de l'application développée par [Lhu00]. Notre objectif est d'offrir au lecteur un échantillon des performances de cette version parallèle pour permettre une comparaison aux résultats séquentiels et aux autres versions parallèles déjà proposées.

Pour chaque exemple, nous allons proposer des informations sur l'accélération et le temps d'exécution. Un tableau comparatif entre le temps séquentiel et le temps parallèle sera montré aussi.

Les résultats présentés dans la suite ont été calculés sur une grappe de 225 processeurs Pentium III 733 Mhz sur Linux connectés via un réseau Ethernet 100. Les nœueds sont distribués en cinq groupes de 43 gérés par un commutateur du type *mesh* de 1 Go (pour plus de détails, regarder dans http ://www-id.imag.fr/Grappes/icluster/UsersGuide.html). Chaque valeur présentée dans la suite est le résultat d'une moyenne arithmétique simple de dix exécutions en éliminant les valeurs extrêmes.

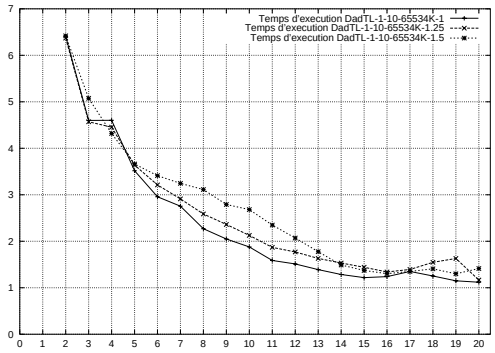
Temps d'exécution

Les temps d'exécution obtenus par la version des tranches souples (version **DadTL** - Distribuée avec découpage en Tranches Limitées) confirment nos suspicions. L'addition du mécanisme de contrôle de la redondance pendant la phase de propagation a payé.

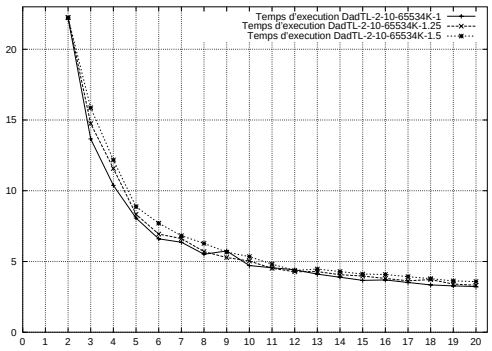
Les courbes des temps d'exécution sont beaucoup plus intéressantes (voir 6.8). À titre de comparaison, nous avons mis trois courbes pour chaque exemple. Chacune des courbes représente l'évolution du temps d'exécution de la phase de propagation de l'application sur un ensemble de vingt nœuds avec une valeur différente pour l'extension permise à chaque tranche. Les valeurs testées sont 100%, 125% et 150%. Nous n'avons pu tester en dessous de 100% car la propagation ne converge plus (c'est-à-dire, elle n'atteint pas la qualité de l'appariement finale préétablie) très vite. Cela arrive parce que nous faisons la distribution des appariements des départ aux processeurs de travail une seule fois. Ainsi, plus on a des tranches, plus elles sont petites et par conséquence moins nous avons des paires de départ sur

certaines tranches. Pour cette raison, si l'on choisit une extension trop petite pour la surface des tranches, la propagation n'atteint pas certaines régions des images et la qualité de l'appariement finale et alors compromise. Ce problème de convergence nous empêche donc d'avoir des meilleurs résultats. Nous en reparlerons dans la suite.

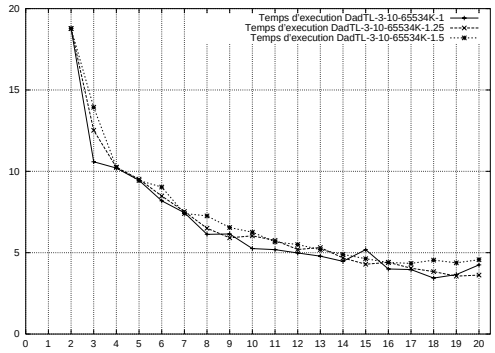
Pour l'instant ce qui est intéressant, c'est la constatation que nous avons trouvé une façon efficace de maîtriser la redondance. Le fait que sur trois des quatre paires d'images exemples de version avec l'extension la plus réduite présente les meilleurs temps d'exécution est une confirmation de cela.



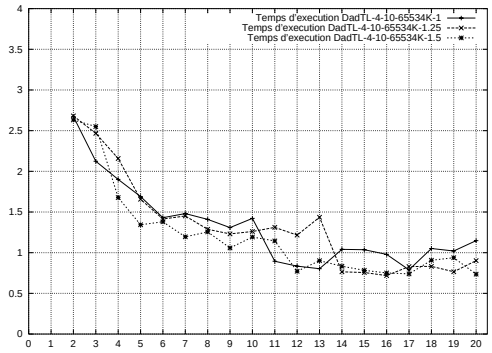
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

FIG. 6.8 – Version Tranches Souples : Comparaison du temps d'exécution par extension

Pour les trois premières paires d'images, le comportement de notre version des tranches souples, malgré l'irrégularité du sujet, est presque régulier. Nous avons des réductions presque toujours constantes de temps d'exécution. Cela confirme que l'algorithme marche très bien pour les grosses images avec un grand pourcentage de leur surfaces appariées. Pour la paire Fleur, nous pouvons noter un essoufflement de l'algorithme à partir de 16 processeurs certainement du à la taille plus petite de l'image.

La paire Tronc, par contre, reste la plus problématique. Sa petite taille et son petit nombre d'appariements possibles pose un problème pour nos versions. Elle est très irrégulière et très vite l'on retombe dans une configuration où plusieurs tranches n'ont pas d'appariements de départ et par conséquence les résultats sont très aléatoires et les courbes se croisent beaucoup. Malgré cela, la réduction du temps d'exécution est présente. Nous ne savons si nous allons réussir à maîtriser ce problème dans les prochaines étapes.

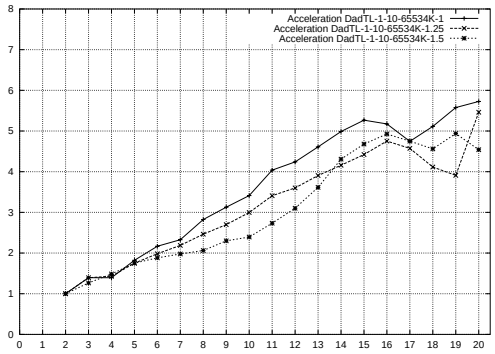
La comparaison avec les autres versions parallèles est montée dans le tableau 7.4. La colonne Nb_{procs} comme d'habitude contient le nombre de processeurs sur lesquels le temps d'exécution a atteint sa valeur minimale. Le lecteur pourra vérifier que la réduction des temps d'exécution est nette pour les paires Fleur, Maison et Rivulet. Par rapport à la paire Tronc, le temps réussi est plus lent que dans la version DadT, mais ils restent très proches. Les temps mis pour la version DadTL ont été obtenus avec l'extension de 100% des tranches.

TAB. 6.2 – Comparaison : temps d'exécution version DadTL et les autres.

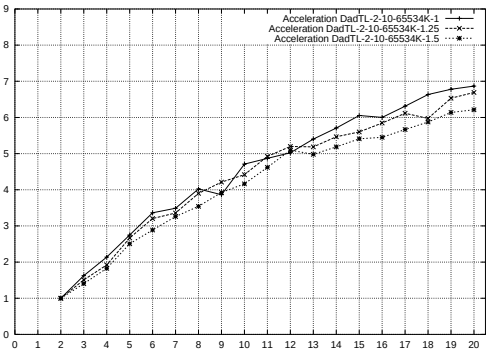
Paire	Nb_{procs}	T_{seq}	$T_{//-DadP}$	$T_{//-DadT}$	$T_{//-DadTL}$
Fleur	20	5.11s	4.31s	5.29s	1.12s
Maison	20	19.75s	12.75s	6.35s	3.23s
Rivulet	18	17.44s	14.76s	13.85s	3.44s
Tronc	17	3.14s	0.88s	0.72s	0.79s

Accélération

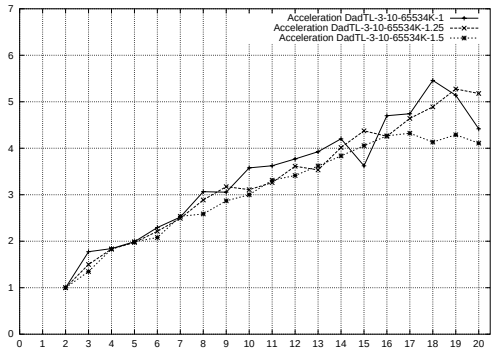
L'analyse des courbes d'accélération pour les quatre paires d'images exemples (figure 6.9) révèle des progrès par rapport aux autres versions, mais montre aussi que les performances obtenues sont très loin encore de l'idéale. Les facteurs d'accélération restent entre 4.5 et 7 pour les paires d'images les plus grosses. Pour la paire Tronc, l'accélération ne dépasse pas un facteur 3.5. La subtilisation de la capacité de calcul des processeurs reste encore trop grande.



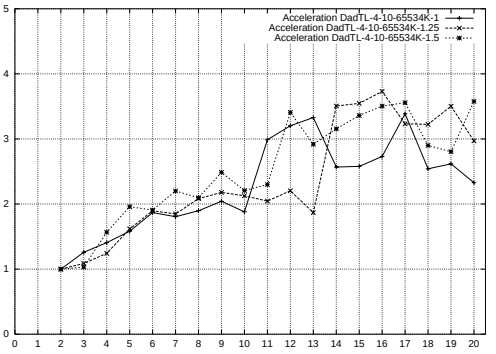
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

FIG. 6.9 – Version Tranches Souples : Comparaison de l'accélération par extension

Les raisons pour une telle mauvaise performance passent certainement par le déséquilibre de charge entre les processeurs de travail. Plus le nombre des tranches augmente, plus grande est la possibilité d'avoir de processeurs qui reçoivent très peu voir aucun appariement de départ. Ainsi, tandis que certains travaillent beaucoup parfois même sur des régions déjà appariées, d'autres ne calculent presque pas. Ce déséquilibre est un facteur qui empêche forcément des meilleures performances et nous allons l'étudier plus en détails dans la suite.

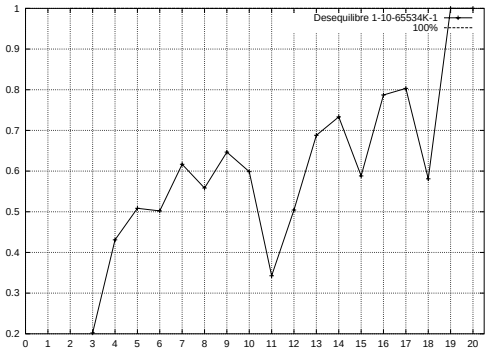
6.4.3 Limitations

L'utilisation des tranches souples (extension des la surface des tranches originales) s'est montrée une évolution importante. Nous avons finalement réussi à établir une façon de contrôler la redondance de calcul sur les processeurs de travail.

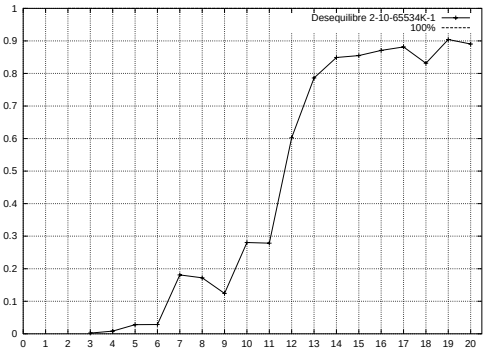
Cela dit, les tranches souples ne peuvent pas résoudre tous les problèmes de parallélisation de l'algorithme de propagation toutes seules. L'utilisation d'un

mécanisme de distribution de travail employé seulement avant le départ de la phase de propagation est un facteur qui empêche une performance plus pointue.

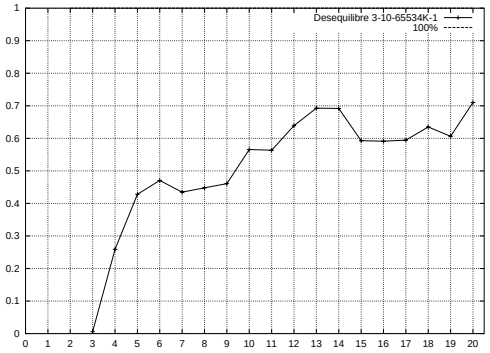
Nous avons pu mesurer les temps de calcul de chacun de processeurs de travail. Après nous avons calculé la différence entre le nœud le plus lent et le plus rapide pour chaque configuration de processeurs. Le résultat est montré dans la figure 6.10.



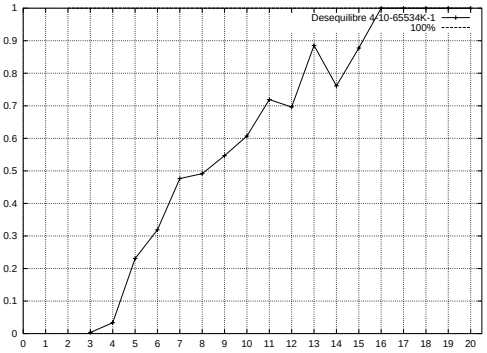
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

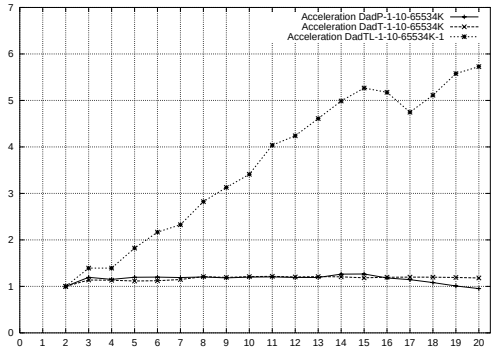
FIG. 6.10 – Version Tranches Souples : Déséquilibre de charge (extension égal à 1)

Nous pouvons constater que le déséquilibre de charge entre le processeur le plus lent et le plus rapide est très vite plus grand que 50% sur les quatre exemples. Pour les paires d'images plus petites, le déséquilibre atteint 100% (c'est-à-dire qu'il y a au moins un processeur qui ne travaille pas). Ces courbes confirment nos spéculations et montrent que nous devons ajouter une façon d'équilibrer la distribution de la charge de travail parmi les processeurs puissions avoir des facteur de performance plus raisonnables.

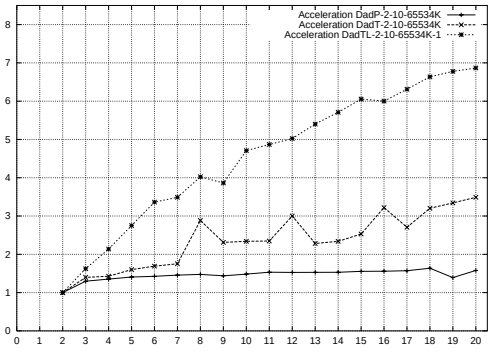
6.5 Bilan

L'idée de base de ce chapitre était la création d'une version parallèle pour l'algorithme de propagation qui profiterait des avantages des deux approches proposées dans le chapitre précédent. Pour y arriver nous avons combiné le mécanisme de contrôle de qualité basée sur la distribution des paires de départ sur un groupe de processeurs de travail avec le découpage en tranches des images pour limiter l'évolution de la propagation sur la surface des images.

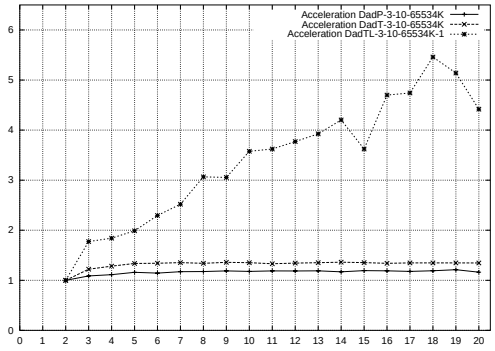
Les résultats obtenus avec la première tentative, la version géographique, ont été décevants. Néanmoins, les observations faites lors de l'analyse de courbes nous ont permis de comprendre une question fondamentale : il fallait trouver une façon de maîtriser la redondance de calcul sur les processeurs de travail pendant l'exécution de l'algorithme de propagation et non seulement lors du choix de la stratégie des distribution des appariements de départ.



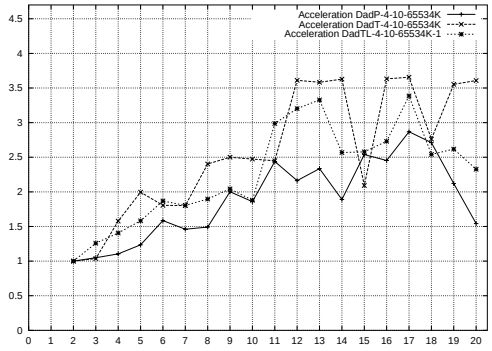
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

FIG. 6.11 – Comparaison de l'accélération pour les trois versions distribuées

Dans la deuxième partie de ce chapitre, nous avons donc proposé une nouvelle

version parallèle pour l'algorithme de propagation qui a finalement conduit à des résultats plus performants. La solution pour le problème de la propagation a été trouvée dans l'idée des tranches souples.

Dans la figure 6.11, on peut voir une comparaison des courbes d'accélération obtenues avec les deux premières versions parallèles (DadP et DadT) et les courbes de la version basée sur le concept des tranches souples (DadTL) pour les quatre paires d'images exemple.

Malgré l'évolution claire dans les performances emmenée par l'utilisation des tranches souples, il reste encore des points à améliorer. Notamment dans la régulation de la charge de travail parmi les nœuds. Nous espérons, avec l'addition d'un mécanisme de distribution équilibrée du travail, pouvoir utiliser de façon plus performante les ressources de calcul et aussi résoudre le problème de convergence de la qualité de l'appariement final.

Chapitre 7

Améliorations

Sommaire

7.1 Introduction

Le sujet principal du chapitre 6 a été la quête d'une solution pour le problème de la redondance de calcul dans le processeurs de travail. Nous avons gardé le schéma distribué proposé dans la deuxième moitié du chapitre 5 et rajouté des modifications dans la façon de distribuer les appariements de départ parmi les processeurs de travail et dans les limites de la progression de la propagation sur la surface des images.

Ces modifications ont apporté un gain de performance considérable par rapport aux premières versions parallèles surtout si l'on se souvient qu'un mécanisme de contrôle de la qualité de l'appariement final a été aussi implanté. Cependant, nous avons détecté des points faibles sur notre version basée sur les tranches souples qui peuvent encore être améliorés.

Les points faibles de la version distribuée basée sur l'idée des tranches souples sont doubles :

- la distribution inégale de la charge de calcul parmi les processeurs de travail ;
- le goulot d'étranglement représenté par le processeur de centralisation des résultats (le même qui fait la distribution des paires de départ).

Dans ce chapitre, nous allons étudier et proposer des solutions pour ces deux items. L'idée est d'abord d'adapter un mécanisme de régulation de charge pour tester son efficacité dans le schéma distribué proposé, pour après étudier une façon de régler la surcharge de travail du processeur de centralisation.

7.2 Régulation de charge

Dans cette section, nous allons présenter la stratégie de régulation de charge que nous avons adoptée, les raisons pour lesquelles nous l'avons choisie et les résultats obtenus en l'utilisant.

La première question à analyser concerne le type de régulation de charge que nous voulons utiliser : dynamique ou centralisée ? En fait, la réponse à cette question a été vite trouvée. Nous avons choisi d'employer une stratégie centralisée car elle s'impose naturellement si nous voulons garder un mécanisme de contrôle de la qualité de l'appariement final. Sur une stratégie dynamique, la résolution du problème de formation d'un tas final unique et sans répétitions de paires appariées ne nous semble pas simple. En réalité, nous croyons qu'une telle approche entraînerait un coût de calcul trop important pour être justifiée.

Une stratégie centralisée par contre possède beaucoup d'inconvénients, mais est la plus adaptée pour résoudre la question de la qualité de l'appariement final. Il s'agit de garder un nœud de centralisation responsable pour faire le tri de toutes les paires qui arrivent des processeurs de travail jusqu'à ce que le niveau de qualité exigé soit atteint.

La deuxième question à étudier afin de choisir l'implantation concerne l'utilisation ou non d'une politique de localité lorsque plus de travail doit être envoyé à un processeur de travail. Est-ce que le processeur de distribution doit prendre en compte le travail réalisé auparavant par le processeur de travail qui demande un nouveau groupe de paires de départ ? L'idée semble intéressante, mais encore une fois il faut analyser le coût de l'opération par rapport au gain qu'elle engendre. Pour adopter une telle stratégie, les nœuds de travail doivent garder une image de toutes les paires qu'ils ont trouvées pour que, lors de la réception d'une nouvelle charge de travail, ils puissent vérifier si les calculs à faire n'ont pas été déjà faits. Cela implique un coût en mémoire et aussi un coût en temps pour la vérification sur le tas de la pertinence de chaque nouvelle paire. L'importance du coût prend des proportions plus graves lorsqu'on considère que la politique de localité serait rarement employée car il n'y a pas moyen d'assurer un ordre de fin de calcul pour les processeurs de travail. Ainsi, il est impossible de prévoir quels processeurs finiront d'abord leurs calculs pour réserver des charges voisines de travail. Pour cette raison, nous avons opté pour ne pas utiliser des politiques de localité dans l'algorithme de régulation de charge qui sera employé.

7.2.1 Implantation

Pour cette version parallèle avec régulation de charge, nous allons garder certaines caractéristiques des versions précédentes avec les adaptations nécessaires pour permettre l'ajout d'un mécanisme de régulation de la charge de travail. Les caractéristiques préservées sont donc :

- contrôle de la qualité de l'appariement final fait par un processeur de centralisation ;
- distribution initiale des appariements de départ basée sur le découpage des images sources en tranches ;
- contrôle de la redondance de calcul fait pendant la phase d'exécution de l'algorithme de propagation à travers le mécanisme des tranches souples.

En préservant ces caractéristiques, nous avons opté pour un schéma du type maître-esclave pour notre nouvelle version parallèle. Il y aura un processeur (le maître) responsable de la distribution de la charge de travail parmi les processeurs esclaves. Les processeurs esclaves feront leur travail et à la fin ils enverront une demande pour plus de travail si il en reste à faire. L'unité de base de travail sera

l'ensemble de paires de départ associé à chaque tranche d'images. Les images sources seront découpées en un nombre de tranches déterminé en fonction du nombre de processeurs esclaves (exemple : deux tranches par processeur esclave). Le schéma de base de la version avec régulation de charge est montré dans la figure 7.1.

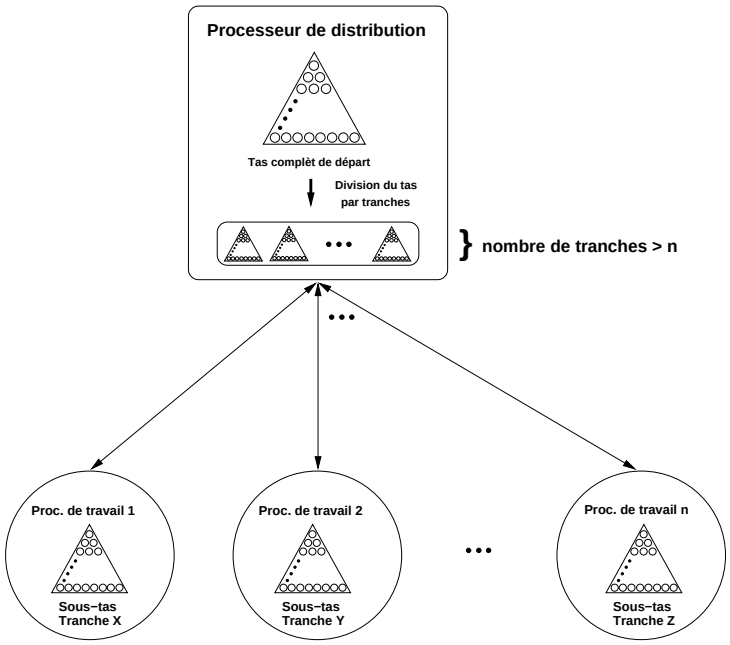


FIG. 7.1 – Schéma avec régulation de charge.

Les communications du maître vers les esclaves seront faites uniquement pour envoyer des ensembles des paires de départ. Dans l'autre direction, les esclaves pourront envoyer, toujours en utilisant les tampons décrits dans le chapitre 5 (section 5.5.3), deux types de messages au maître :

- tampon plein - le nombre d'appariements nécessaire pour remplir le tampon de communication a été atteint. Le tampon est donc envoyé au maître pour qu'il fasse la centralisation sur le tas final ;
- fin de travail - le tampon d'appariements est envoyé par l'esclave au maître incomplet, c'est-à-dire que l'esclave n'a plus de travail à faire sur la tranche qui lui a été attribuée.

Le maître, selon le type de message envoyé par l'esclave, peut simplement recevoir le paquet et continuer son travail de triage ou bien renvoyer une nouvelle charge de travail à l'esclave demandeur. Lorsque les images sont découpées en un grand nombre de tranches, il arrive très souvent que certaines tranches n'aient pas de paires de départ sur sa surface. Dans ce cas, ces tranches ne sont pas envoyées

aux esclaves. Leur surface sera appariée par d'autres tranches grâce à l'extension permise à l'algorithme de propagation en dehors des tranches.

Pourquoi l'algorithme glouton ?

La description faite auparavant du schéma de régulation de charge que nous allons adopter est celle de l'algorithme glouton, c'est-à-dire : **la prochaine tâche de travail à être calculée est attribuée au premier nœud qui finit son calcul et en demande plus**. Nous allons expliquer dans la suite pourquoi nous avons choisi cet algorithme pour implanter un mécanisme de régulation de charge dans notre version parallèle de l'algorithme de propagation.

D'abord, nous supposons que dans notre version parallèle de l'algorithme de propagation, le temps des communications peut être envisagé comme étant négligeable par rapport au temps de calcul. Ensuite, nous imaginons une version parallèle de l'algorithme de propagation avec régulation de charge basée sur **n'importe quel algorithme d'ordonnancement de type liste** exécutée sur p processeurs. Dans un tel cas, le temps total d'inactivité (T_{inac}) des processeurs serait donné par $p \times T_{//} - T_{seq}$ (voir exemple dans la figure 7.2).

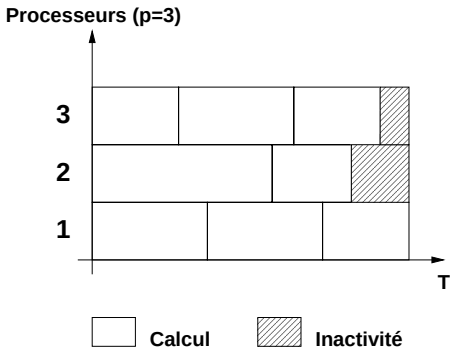


FIG. 7.2 – Régulation de charge : exemple d'activité normale.

Dans le même contexte, nous pouvons aussi imaginer l'occurrence du cas critique de cet algorithme lorsqu'une tâche est beaucoup plus longue que les autres. Tous les autres processeurs qui ne seraient pas en train de l'exécuter resteraient inactifs (voir figure 7.3). Le temps d'exécution de cette tâche serait donc le temps minimal d'exécution de l'algorithme sur un nombre infini de processeurs, on l'appelle T_{min}. Ainsi, le temps d'inactivité T_{inac} serait égal à $(p-1) \times T_{min}$.

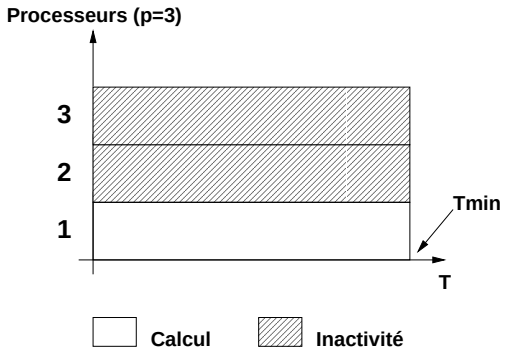


FIG. 7.3 – Régulation de charge : cas critique.

Alors, dans la plupart des cas, $p \times T_{//} - T_{seq} \leq (p-1) \times T_{min}$. D'où :

$$T_{//} \leq \frac{T_{seq}}{p} + (1 - \frac{1}{p}) \times T_{min}$$

Si l'on considère que le temps d'ordonnancement optimal ($T^*_{//}$) sera toujours supérieur ou égal à T_{min}, et que T_{seq} sera toujours inférieur ou égal à $p \times T^*_{//}$, on peut remplacer dans la formule ci-dessus pour obtenir : $T_{//} \leq (2 - \frac{1}{p}) \times T^*_{//}$. On arrive donc à un ratio de performance borné par $(2 - \frac{1}{p})$.

Le raisonnement ci-dessus est basé sur la preuve de la borne de Graham [Gra66]. Dans sa version originale, on montre également qu'ajouter des informations sur les tâches, comme la durée de leur exécution, ne permet pas d'améliorer le ratio de performance sauf à restreindre le type de graphe de flux de données traités.

En ce que nous concerne, la conclusion est que, pour n'importe quel algorithme d'ordonnancement de type liste, si l'on travaille sur un nombre important de nœuds identiques et la longueur de la tâche la plus grosse n'est pas beaucoup plus grande que (T_{seq}/p), l'algorithme glouton est tout aussi efficace que d'autres plus lourds et compliqués.

7.2.2 Résultats

Dans cette section, nous allons présenter les résultats obtenus par l'exécution de notre version parallèle avec régulation de charge de l'application développée par [Lhu00]. Notre objectif est d'offrir au lecteur un échantillon des performances de cette version parallèle pour permettre une comparaison aux résultats séquentiels et aux autres versions parallèles déjà proposées ou à proposer.

Pour chaque exemple, nous allons proposer des informations sur l'accélération et le temps d'exécution obtenus. Le déséquilibre de charge sera aussi discuté. Une comparaison avec les autres versions parallèles déjà présentées sera faite. Nous finirons sur un tableau comparatif entre le temps séquentiel et le temps parallèle atteint pour cette version.

Les résultats présentés dans la suite ont été calculés sur une grappe de 225 processeurs Pentium III 733 Mhz sur Linux connectés via un réseau Ethernet 100. Les nœueds sont distribués en cinq groupes de 43 gérés par un commutateur du type *mesh* de 1 Go (pour plus de détails, regarder dans http ://www-id.imag.fr/Grappes/icluster/UsersGuide.html). Chaque valeur présentée dans la suite est le résultat d'une moyenne arithmétique simple de dix exécutions en éliminant les valeurs extrêmes.

Accélération

Les courbes d'accélération pour la version de l'algorithme de propagation avec régulation de charge (version **Glouton**) sont montrées dans la figure 7.4. Pour générer ces courbes nous avons fait varier deux paramètres :

1. **l'extension** : nous avons utilisé des valeurs entre 100% et 30% (1.0 et 0.3) de la surface des tranches ;
2. **le nombre de tranches par processeur** : les valeurs ont varié de 1.5, 2.0 et 3.0 tranches par processeur ;

Par rapport à l'extension, le lecteur se rappellera que dans la version **DadTL** nous n'avons pas pu utiliser une valeur d'extension en dehors des tranches inférieures à 100% (extension = 1.0) de la surface d'une tranche. Avec l'ajout du mécanisme de régulation de charge décrit dans la section précédente, cette restriction a été réduite. Nous avons pu aller jusqu'à une valeur de 30% pour l'extension (extension = 0.3). Cela a été possible grâce à l'élimination des attributions des tranches sans appariements de départ aux processeurs de travail.

Dans la version DadTL, la distribution des appariements de départ était faite d'une seule fois automatiquement en fonction de la division en tranches des images laquelle était déterminée par le nombre de processeurs utilisés. Ainsi, un processeur de travail recevait parfois une tranche sans aucun appariement de départ sur sa surface. Pour compenser cela, il fallait une extension de la propagation en dehors des tranches plus importante. Les zones de la surface des images correspondant à des tranches sans paires de départ pouvaient alors être appariées.

Dans la version avec régulation de charge, les processeurs ne reçoivent jamais des tranches sans appariements de départ. Lors de la formation de sous-tas de

paires de départ associés à chaque tranche, les tranches ayant des sous-tas vides sont éliminées. En plus il y a plus d'une tranche par processeur et les tranches sont donc plus fines par rapport au même nombre de processeurs pour la version DadTL. Ainsi, les zones que ne sont pas appariées à cause du manque de paires de départ ont une surface plus réduite et peuvent alors être atteintes par des extensions plus courtes venant d'autres tranches. Bien sûr, cet argument ne vaut que jusqu'à un certain point puisque en dessous de 30 % d'extension l'algorithme de propagation n'arrive plus a converger.

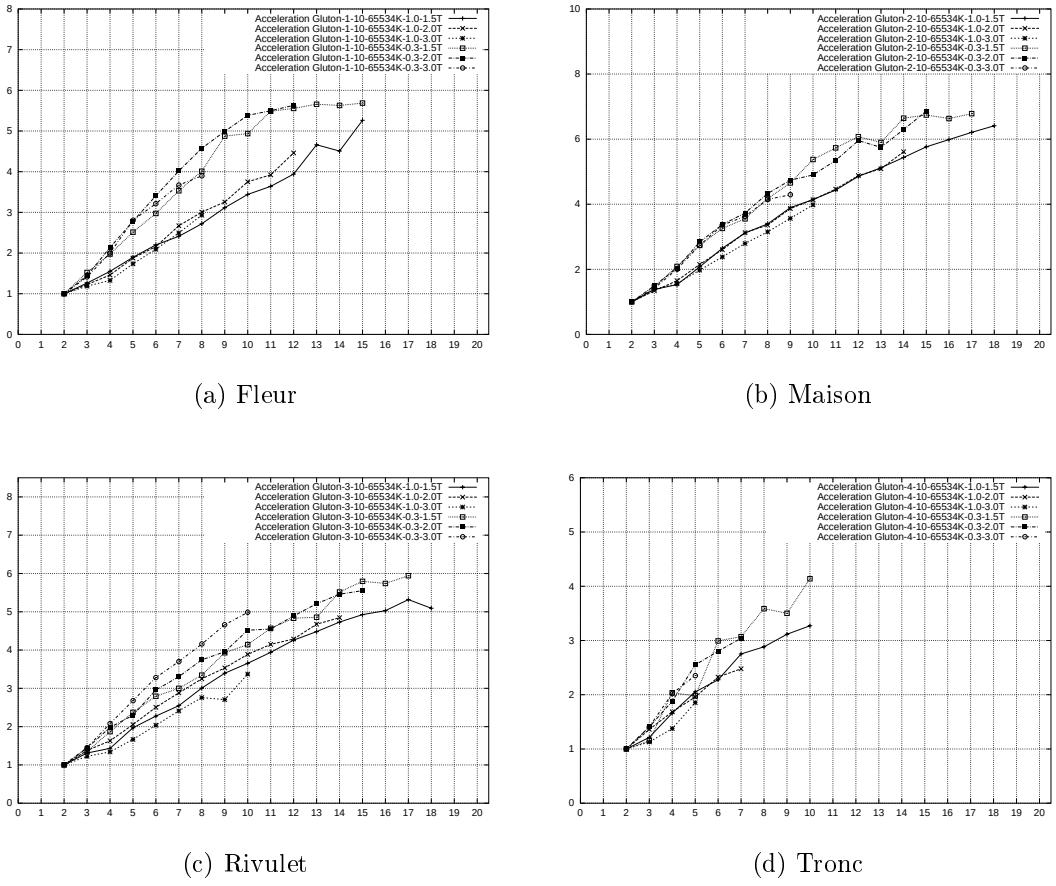
L'autre paramètre, le nombre de tranches par processeur, nous a permis d'étudier l'influence du découpage des images dans la performance de la version avec régulation de charge. Bien évidemment, plus on a des tranches par processeur, plus petit est le nombre maximum de processeurs qui peut être utilisé car chaque image peut être découpée en un nombre limité de tranches. Le nombre maximum de processeurs qui peut être employé pour chaque paire d'images exemple est montré dans le tableau 7.1.

TAB. 7.1 – Nombre maximum de processeurs par tranche pour chaque image (extension entre parenthèses).

Paire	1.5 (1.0)	2.0 (1.0)	3.0 (1.0)	1.5 (0.3)	2.0 (0.3)	3.0 (0.3)
Fleur	15	12	8	15	12	8
Maison	18	14	10	17	15	9
Rivulet	18	14	10	17	15	10
Tronc	10	7	5	10	7	5

Le nombre maximum de tranches en lequel une image peut être découpée varie selon les dimensions des images et l'extension employée. Encore une fois, nous rappelons au lecteur qu'à chaque fois que nous altérons le découpage des images, nous modifions aussi les sous-tas de paires de points de départ qui leur sont associés et par conséquent nous changeons aussi la façon d'évoluer de la stratégie "meilleur d'abord" locale à chaque tranche. Ainsi, il est normal que pour la même paire d'images, le nombre maximum de tranches permis ne soit pas le même si l'on varie l'extension adoptée, par exemple.

L'analyse comparative des courbes d'accélération dans la figure 7.4 permet de confirmer encore une fois que la limitation de l'extension a des reflets directs sur la réduction de la redondance de calcul. Les courbes dont la valeur de l'extension est de 0.3 sont nettement supérieures à celles avec une extension égale à 1.0. Cela se vérifie sur les quatre paires d'images. D'un autre côté, la variation du nombre de tranches par processeurs ne résulte pas en de grosses différences de performance. Les courbes présentent à peu près la même pente, laquelle diminue lorsque le nombre limite de tranches par images devient proche.



(a) Fleur (b) Maison

(c) Rivulet (d) Tronc

FIG. 7.4 – Comparaison de l'accélération en fonction du nombre de tranches par processeur (extension = 0.3 et 1.0)

En regardant ces courbes, le lecteur peut constater que l'addition du mécanisme de régulation de charge basé sur l'algorithme glouton a permis une meilleure utilisation des ressources à disposition car les courbes d'accélération montent plus vite que dans les versions précédentes. Une comparaison des courbes sera faite à la fin de ce chapitre.

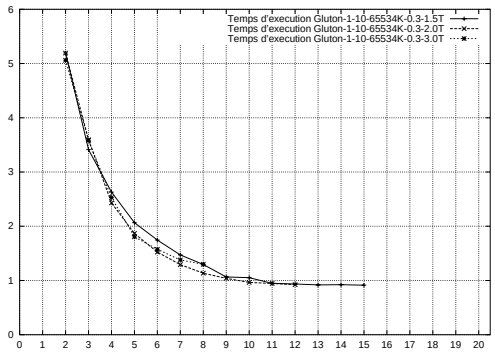
Temps d'exécution

Nous allons maintenant présenter les temps d'exécution obtenus avec la nouvelle version parallèle de l'algorithme de propagation laquelle est basée sur un mécanisme de régulation de charge du type glouton. Les courbes pour les quatre paires d'images source sont montrées dans la figure 7.5.

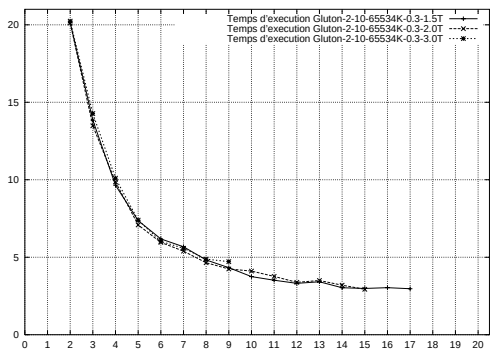
Les résultats présentés dans les courbes correspondent à des exécutions avec

une extension de la propagation en dehors de tranches équivalente à 30% (0.3). C'est avec cette valeur d'extension que nous avons obtenu les meilleurs courbes d'accélération. Les temps pour l'exécution avec la zone d'extension en dehors des tranches équivalente à 100% de la surface de celles-ci seront discutés plus tard. Pour chaque exemple nous montrons les courbes des temps pour les trois rapports tranches / processeurs (1.5, 2.0 et 3.0).

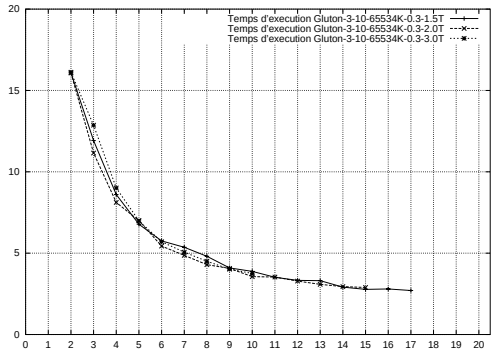
Une analyse rapide nous permet de visualiser que le comportement de la version avec régulation de charge est encore plus régulier que celui de la version précédente (DadTL). La réduction des temps d'exécution suit presque la même pente pour les trois rapports nombre de tranches / processeur testés dans les trois premiers exemples (Fleur, Maison et Rivulet). Encore une fois c'est l'exemple Tronc qui présente le plus de variations, sans doute à cause de son nombre réduit d'appariements de départ et de sa petite surface finale appariée.



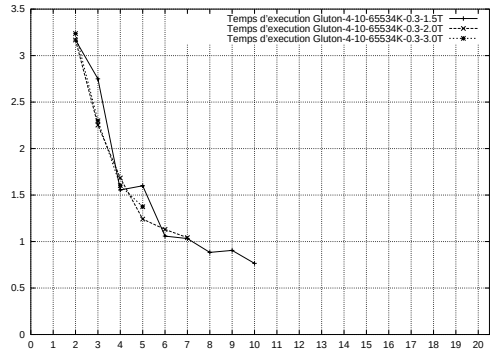
(a) Fleur



(b) Maison



(c) Rivulet



(d) Tronc

FIG. 7.5 – Comparaison du temps d'exécution en fonction du nombre de tranches par processeur (extension = 0.3)

Néanmoins, l'analyse des courbes ne nous permet pas d'identifier les meilleurs

temps d'exécution pour chaque configuration. Nous avons donc élaboré le tableau 7.2 pour permettre une comparaison entre les résultats obtenus avec la variation de deux paramètres : l'extension (0.3 et 1.0) et le rapport nombre de tranches / processeur (1.5, 2.0 et 3.0). Sur le tableau l'on peut voir les temps d'exécution pour les six combinaisons possibles des valeurs des paramètres. Pour chacune des combinaisons, le nombre de processeurs sur lequel le meilleur temps a été obtenu est mis entre parenthèses.

TAB. 7.2 – Temps d'exécution pour la version Glouton (extension - tranches / processeur)

Paire	$T_{//-1.0-1.5}$	$T_{//-1.0-2.0}$	$T_{//-1.0-3.0}$	$T_{//-0.3-1.5}$	$T_{//-0.3-2.0}$	$T_{//-0.3-3.0}$
Fleur	0.98s (15)	1.15s (12)	1.80s (18)	0.91s (15)	0.92s (12)	1.30s (8)
Maison	3.42s (18)	3.91s (14)	5.52s (10)	2.97s (17)	2.93s (15)	4.72s (9)
Rivulet	3.67s (17)	4.03s (14)	5.40s (10)	2.71s (17)	2.89s (15)	3.75s (10)
Tronc	0.93s (10)	1.23s (7)	1.76s (5)	0.77s (10)	1.04s (7)	1.37s (5)

Les meilleurs résultats pour les quatre paires d'images exemples ont été obtenus en utilisant une extension de 30% et un rapport nombre de tranches par processeur égal à 1.5. L'utilisation d'un nombre plus élevé de tranches par processeur n'est pas une bonne stratégie car la convergence de qualité devient plus difficile à atteindre. Les tranches plus fines ont moins de paires de départ et ainsi il y a beaucoup de propagations qui démarrent avec une baisse des valeurs de mesure ZNCC. Ces propagations ne vont pas très loin et, pour compenser, il faut calculer plus.

TAB. 7.3 – Comparaison : temps d'exécution version Glouton x version DadTL.

Paire	$T_{//-DadTL}$	$T_{//-1.0-1.5}$	$T_{//-0.3-1.5}$
Fleur	1.12s (20)	0.98s (15)	0.91s (15)
Maison	3.23s (20)	3.42s (18)	2.97s (17)
Rivulet	3.44s (18)	3.67s (17)	2.71s (17)
Tronc	0.79s (17)	0.93s (10)	0.77s (10)

Pour permettre au lecteur une comparaison entre la version DadTL avec extension égale à 100% et la version avec régulation de charge nous avons élaboré le tableau 7.3. Le lecteur peut constater que entre la version DadTL et la version Glouton avec extension égale à 100% les temps sont très proches (avec un léger avantage pour la version DadTL). À l'exception de la paire Fleur, les autres ont un petit retard de 2 dixièmes de seconde. L'avantage de la version avec régulation de charge est qu'elle nous permet d'avoir presque les mêmes temps d'exécution sur un nombre plus réduit de nœuds. Le gain est encore plus important lorsqu'on fait la comparaison entre la version DadTL et la version Glouton avec une extension de

30%. Dans ce cas, le résultat est clair : la version Glouton offre de meilleurs temps d'exécution sur un nombre plus petit de processeurs. Il faut rappeler que, sans le mécanisme de régulation de charge, la valeur de l'extension ne pouvait pas être inférieure à 100%.

Finalement, nous présentons sur le tableau 7.4 les meilleurs résultats obtenus avec chacune des version parallèles qui présentent un mécanisme de contrôle de la qualité de l'appariement final. La colonne Nb_{procs} contient le nombre de processeurs sur lequel le meilleur temps d'exécution a été trouvé par la dernière version rajoutée au tableau ($T_{//-Glt.}$). Pour les paires Fleur, Maison et Rivulet, les temps d'exécution ont encore été améliorés. La paire Tronc semble avoir atteint un seuil difficile à dépasser, mais avec la version Glouton le temps a été obtenu sur dix processeurs seulement.

TAB. 7.4 – Comparaison : temps d'exécution version Glouton et les autres.

Paire	Nb_{procs}	T_{seq}	$T_{//-DadP}$	$T_{//-DadT}$	$T_{//-DadTL}$	$T_{//-Glt.}$
Fleur	15	5.11s	4.31s	5.29s	1.12s	0.91s
Maison	17	19.75s	12.75s	6.35s	3.23s	2.97s
Rivulet	17	17.44s	14.76s	13.85s	3.44s	2.71s
Tronc	10	3.14s	0.88s	0.72s	0.79s	0.77s

7.2.3 Limitations

L'adoption d'un mécanisme de régulation de charge pour la version distribuée basée sur les tranches souples nous a permis d'atteindre des temps d'exécution plus rapides sur un nombre plus réduit de processeurs. Malgré cela, le gain obtenu n'a pas été vraiment important par rapport à la version antérieure (DadTL).

Nous croyons que désormais, après la maîtrise de la redondance de calcul et du problème de déséquilibre de charge entre les nœuds, le plus grand goulot d'étranglement pour une meilleure performance réside dans l'unification des paires appariées faite par le processeurs de centralisation (le maître de la dernière version). Il nous semble qu'il est surchargé, puisqu'il doit distribuer le travail, recevoir les appariements résultats et en plus les trier pour éviter la répétition dans le tas final. Une idée déjà discutée auparavant est la possibilité d'utiliser un mécanisme de régulation de charge dynamique où il n'y a pas un nœud central responsable par le contrôle de la charge de travail de tous les nœuds de travail. Cela résout la question de la distribution du travail, mais nous resterons obligés d'avoir un processeur pour centraliser les appariements faits pour chaque processeurs car c'est le seul moyen d'assurer la qualité de l'appariement final.

Il nous semble donc plus important de concentrer nos efforts sur l'élaboration d'un moyen de soulager le travail de triage fait par le processeur de centralisation plutôt que de rajouter un mécanisme pour permettre une distribution dynamique de la charge de travail.

7.3 Hiérarchisation du maître

La version basée sur l'algorithme glouton a permis un gain de performance par rapport aux autres versions, mais ce gain n'a pas été à la hauteur de nos espérances. Les améliorations apportées dans les dernières versions concernent :

- la maîtrise de la qualité désirée pour l'appariement final ;
- le contrôle de la redondance de calcul sur les processeurs de travail ;
- la régulation de la charge de travail parmi les nœuds de calcul.

L'addition de chacun de ces mécanismes a prouvé son efficacité. Les temps d'exécution ont été réduits sur les quatre paires d'images exemples testées. Dans les meilleurs cas, le temps d'exécution a baissé de presque 85% (Maison - 84.96%, Rivulet - 84.46%, et Fleur - 82.19%) par rapport au temps d'exécution séquentiel. La réduction la moins importante est vérifiée sur la paire Tronc avec une diminution de presque 70% (69.71%) de son temps d'exécution. Donc, en terme de temps d'exécution, les résultats sont assez satisfaisants.

Mais nous croyons qu'il reste encore des possibilités d'amélioration de la performance de notre version parallèle de l'algorithme de propagation. Une des possibilités à notre avis passe certainement par le travail du processeur maître. Ce processeur nous semble surchargé après l'addition du mécanisme de régulation de charge car désormais il est en charge de la gestion de quatre tâches :

- l'envoi des appariements de départ aux processeurs de travail ;
- la réception des tampons envoyés par les processeurs de travail avec les appariements définitifs ;
- le tri des appariements reçus car la redondance de calcul des processeurs de travail ne peut pas être totalement éliminée ;
- la vérification à chaque nouvelle addition à l'ensemble définitif d'appariements si la qualité désirée a été atteinte.

Dans cet ensemble de tâches attribuées au processeur maître, la plus coûteuse est le tri. Il consiste en une vérification dans le tas final d'appariements si chaque nouveau appariement reçu par le maître n'y est pas déjà. Le tableau 7.5 présente un exemple qui renforce cette suspicion. Il s'agit du nombre de paires calculées par chaque processeur, hormis le maître, pour chacune de nos paires d'images source

en utilisant la dernière version parallèle (version glouton) sur quatre nœuds (un maître et 3 esclaves) avec 1.5 tranches par processeur et une expansion de 0.3 (30%).

TAB. 7.5 – Comparaison : nombre d'appariements calculés x nécessaires.

Nombre d'apps.	esclave 1	esclave 2	esclave 3	Calculé	Nécessaire
Fleur	63965	38397	67744	170106	105000
Maison	113325	143094	96132	352551	225000
Rivulet	144327	140264	79572	364163	230000
Tronc	15788	15059	11557	42404	28000

Le lecteur peut constater que le nombre de paires envoyées au maître est toujours supérieur de au moins 60% au nombre de paires nécessaires (l'explication sur l'estimation du nombre d'appariements nécessaires se retrouve dans la section 5.5.2). Pour chaque paire gérée et envoyée par les processeurs de travail au processeur de centralisation, il faut vérifier s'il n'est déjà pas dans le tas du maître lequel contient le nombre final d'appariements.

Dans cette section, nous voulons investiguer la pertinence d'une modification du schéma de triage du maître. La solution que nous proposons dans la suite est la hiérarchisation du travail de triage du processeur maître.

7.3.1 Implantation

Dans cette version parallèle de l'algorithme de propagation, nous allons proposer un mécanisme pour soulager la charge de travail du processeur maître. Les autres aspects rajoutés jusqu'ici aux dernières versions parallèles seront conservés. Nous rappelons que l'existence d'un processeur de centralisation reste inévitable, car c'est le seul moyen efficace pour contrôler la qualité de l'appariement final.

Nous allons donc implanter une version avec une hiérarchisation du processeur maître. Pour cela, nous allons ajouter au schéma de parallélisation un nouvel élément : le sous-maître. Le sous-maître est un type de processeur qui sera placé entre le vrai maître et les processeurs de travail. Il sera en communication avec un sous-ensemble des processeurs de travail et sera responsable de la jonction dans un tas intermédiaire des appariements envoyés par ses esclaves. Le nouveau schéma peut être visualisé dans la figure 7.6.

Le mécanisme de distributions des appariements de départ a été un petit peu modifié. À chaque fois qu'un processeur finit le calcul sur une tranche qui lui a été attribuée, il envoi un message au maître pour l'avertir qu'il attend plus de travail. Sinon, tous les tampons intermédiaires contenant les appariements calculés par

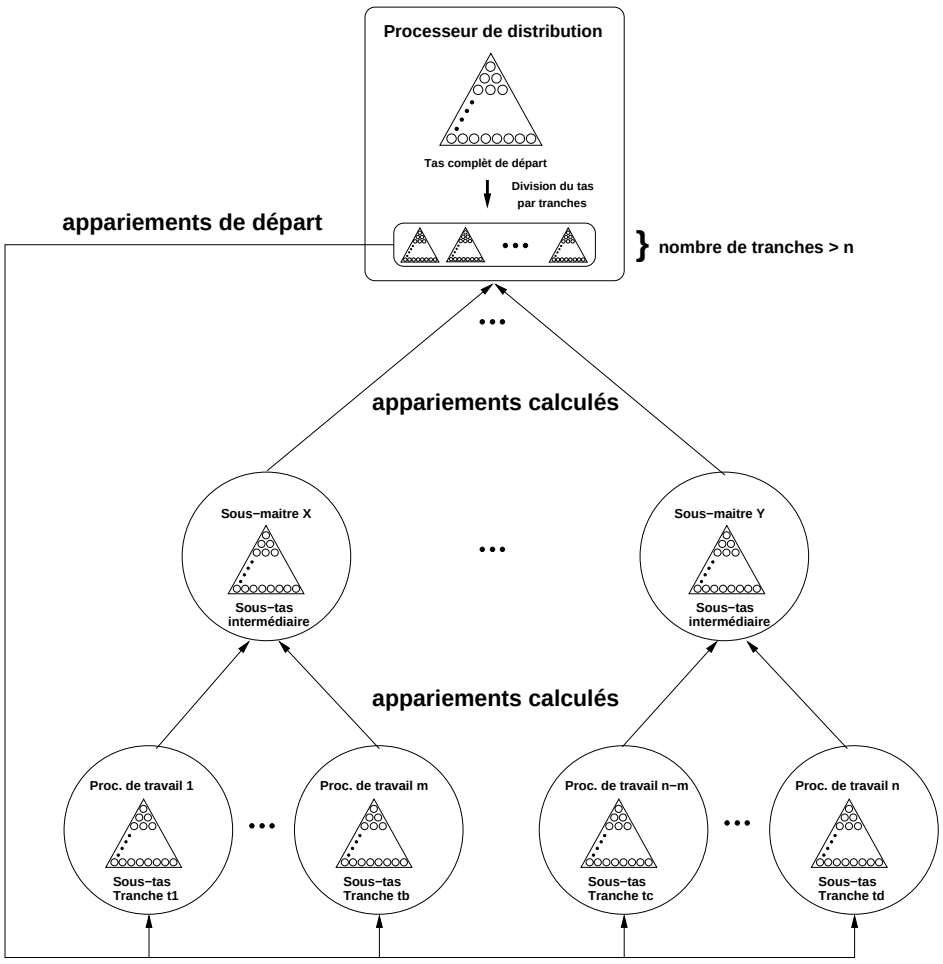


FIG. 7.6 – Schéma hiérarchique.

le processeur esclave sont envoyés à un sous-maître. Chaque processeur esclave a uniquement un processeur sous-maître. Ce sous-maître possède un nombre défini d'esclaves et son travail consiste à faire le tri des appariements envoyés par ses esclaves. Lorsque le tas d'appariements du sous-maître atteint un nombre de paires préétabli, ces paires sont alors envoyées au maître. Le maître, à son tour, reçoit les paires déjà partiellement triées et fait son tri à lui. Ce schéma possède donc deux paramètres : le nombre d'esclaves par sous-maître et la taille du tampon d'appariements à envoyer des sous-maîtres au maître.

Le nombre d'esclaves par sous-maître ne doit pas être trop petit car sinon il n'y a pas de raison pour en avoir un. D'un autre côté, il ne doit pas non plus être trop élevé car sinon le problème du goulot d'étranglement est simplement transféré du maître aux sous-maîtres. Pour notre version, nous avons établi un nombre maximum de quatre esclaves par sous-maître, mais chaque sous-maître peut travailler avec un, deux, trois ou quatre esclaves. Lorsque nous avons besoin d'un cinquième esclave,

un nouveau sous-maître est créé.

Si l'on dispose, par exemple, de 5 processeurs pour faire tourner l'application, nous aurons un maître et un sous-maître avec trois esclaves. Cela veut dire que, pour un nombre petit de nœuds à disposition, les performances de cette version seront inférieures à celles de la version précédente. Cette version devient intéressante à partir d'une configuration où le nombre d'esclaves à disposition est assez grand pour que plus d'un sous-maître soit utilisé, car ainsi ils travailleront en parallèle entre eux et en *pipeline* avec les esclaves et le maître.

Cette version deviendra intéressante lorsque nous disposerons donc d'au moins huit processeurs (un maître, deux sous-maîtres, le premier avec quatre esclaves et le deuxième avec seulement un esclave). Les performances idéales doivent être atteintes lorsque nous utilisons un multiple de quatre pour le nombre d'esclaves car ainsi chaque sous-maître travaille au maximum de sa charge.

Par rapport à la taille du tampon de communication entre les sous-maîtres et le maître, nous avons gardé la taille utilisée auparavant lorsque la communication était faite directement des esclaves au maître. Ainsi les communications restent homogènes et le système est plus simple à gérer.

7.3.2 Résultats

Dans cette section, nous allons présenter les résultats obtenus par l'exécution de notre version parallèle hiérarchique de l'application développée par [Lhu00]. Notre objectif est d'offrir au lecteur un échantillon des performances de cette version parallèle pour permettre une comparaison aux résultats séquentiels et aux autres versions parallèles déjà proposées ou à proposer. Néanmoins nous voulons prévenir le lecteur que, au contraire des autres versions, les résultats de celle-ci seront présentés seulement sur une configuration de paramètres.

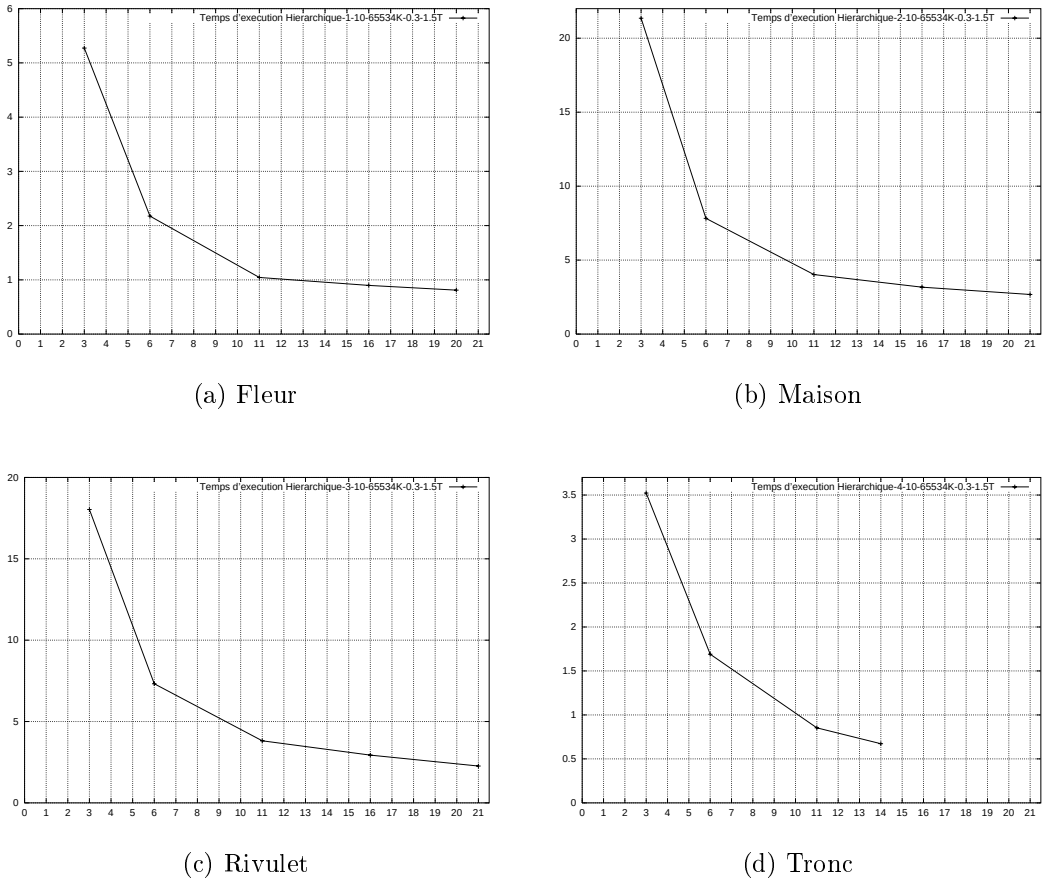
Pour chaque exemple, nous allons proposer des informations sur l'accélération et le temps d'exécution obtenus. Nous finirons sur un tableau comparatif entre le temps séquentiel et le temps parallèle atteint pour cette version. Pour que les courbes soient plus lisibles, nous allons mesurer les temps d'exécution **uniquement sur les configurations idéales**, c'est-à-dire avec un nombre d'esclaves multiple de quatre, jusqu'à ce que le nombre maximum de processeurs supportés pour chaque paire d'images exemple soit atteint.

Les résultats présentés dans la suite ont été calculés sur une grappe de 225 processeurs Pentium III 733 Mhz sur Linux connectés via un réseau Ethernet 100. Les nœueds sont distribués en cinq groupes de 43 gérés par un commutateur du type *mesh* de 1 Go (pour plus de détails, regarder dans http ://www-id.imag.fr/Grappes/icluster/UsersGuide.html). Chaque valeur présentée dans la

suite est le résultat d'une moyenne arithmétique simple de dix exécutions en éliminant les valeurs extrêmes.

Temps d'exécution

Nous allons maintenant présenter les temps d'exécution obtenus avec la nouvelle version parallèle de l'algorithme de propagation laquelle est basée sur un mécanisme de régulation de charge du type glouton plus le schéma hiérarchique pour diminuer la charge de travail du processeur maître. Les courbes pour les quatre paires d'images source sont montrées dans la figure 7.7.



(a) Fleur (b) Maison

(c) Rivulet (d) Tronc

FIG. 7.7 – Version Hiérarchique : Temps d'exécution

Les résultats présentés dans les courbes correspondent à l'exécution avec une extension de la propagation en dehors de tranches équivalente à 30% (0.3) un rapports tranches / processeurs de 1.5. Cette configuration a été la plus performante sur la version antérieure et nous l'avons gardé pour nos tests sur la version hiérarchique. Les temps n'ont été mesurés que sur les configurations où le nombre d'esclaves est

optimal (multiple de quatre) jusqu'à ce que le nombre maximal de nœuds soit atteint.

Le comportement des courbes pour les quatre paires d'images exemples est assez régulier. La paire Tronc qui était la plus problématique sur les versions précédentes, cette fois semble bien se comporter. Mais cela peut être simplement le résultat d'un nombre plus réduit d'exécutions.

Nous présentons dans le tableau 7.6 les meilleurs résultats obtenus avec chacune des version parallèles qui présentent un mécanisme de contrôle de la qualité de l'appariement final. La colonne Nb_{procs} contient le nombre de processeurs sur lequel le meilleur temps d'exécution a été trouvé par la dernière version rajoutée au tableau ($T_{//-Hch.}$). Pour les paires Fleur, Maison et Rivulet, les temps d'exécution ont encore été améliorés. La paire Tronc a montré une petite évolution du temps d'exécution, mais il a fallu rajouter quatre processeurs en plus par rapport à la version précédente. Nous rappelons encore une fois au lecteur que les résultats de la version Hiérarchique doivent être pris avec précaution puisque pour cette version le nombre de tests a été plus réduit que pour les autres versions.

TAB. 7.6 – Comparaison : temps d'exécution version Hiérarchique et les autres.

Paire	Nb_{procs}	T_{seq}	$T_{//-DadP}$	$T_{//-DadT}$	$T_{//-DadTL}$	$T_{//-Glt.}$	$T_{//-Hch.}$
Fleur	20	5.11s	4.31s	5.29s	1.12s	0.91s	0.81
Maison	21	19.75s	12.75s	6.35s	3.23s	2.97s	2.68
Rivulet	21	17.44s	14.76s	13.85s	3.44s	2.71s	2.27
Tronc	14	3.14s	0.88s	0.72s	0.79s	0.77s	0.67

7.3.3 Limitations

Nous venons de présenter notre dernière version parallèle pour l'algorithme de propagation basée sur la stratégie "meilleur d'abord". L'implantation de cette version est encore préliminaire, il n'y a pas assez de raffinements. Notre objectif était plutôt de vérifier si le travail de triage des appariements fait par le maître était vraiment un goulot d'étranglement. Les résultats obtenus sur une configuration de paramètres nous indiquent que la réponse à cette question est positive.

Nous croyons qu'avec une étude plus approfondie, cette version hiérarchique peut être encore améliorée car dans son état actuel elle présente beaucoup de points faibles. Une des possibilités à étudier est l'implantation de sous-maîtres à travers les facilités offertes par l'environnement Athapascan pour l'utilisation de processus légers (*threads*). D'autres optimisations peuvent être proposées et ajoutées en analysant plus finement un ensemble plus large des résultats.

Les résultats que nous avons présentés pour la version Hiérarchique doivent être pris uniquement comme une indication du fait qu'il y a encore d'autres aspects à travailler pour l'obtention d'une performance plus pointue pour l'algorithme de propagation basée sur un schéma hiérarchique.

7.4 Bilan

L'idée initiale de ce chapitre était de chercher des moyens d'optimiser les résultats obtenus avec la version DadTL. Là-bas, les temps d'exécution ont atteint des niveaux très satisfaisants mais nous avons vérifié que les ressources pouvaient être mieux utilisées. Pour essayer de résoudre ce problème, nous avons ajouté deux améliorations à la version DadTL, à savoir : régulation de charge et un mécanisme de hiérarchisation du travail de triage du processeur maître.

Les résultats obtenus par la version avec régulation de charge basée sur l'algorithme glouton ont été assez stimulants. L'algorithme a pu aller plus vite grâce à la meilleure utilisation des processeurs et, plus important, il a gagné en plus en performance grâce à la possibilité de réduire l'extension des tranches souples.

Finalement, la version hiérarchique nous a permis, malgré les limitations déjà citées, de confirmer nos suspicions par rapport à la surcharge du processeur maître en ce que concerne le triage des appariements envoyés par les processeurs de travail. Cette version nous porte à croire que des améliorations sont encore possibles qui permettraient des temps d'exécution un peu plus courts. Cela dit, nous croyons que, pour les paires d'images exemples des dimensions plus réduites, nous sommes très proches des limites. La faible évolution des temps pour les paires Fleur et Tronc depuis la version DadTL nous conduisent à penser ainsi.

Chapitre 8

Conclusions et Perspectives

Sommaire

La problématique centrale de cette thèse est la mise en parallèle d'un algorithme de mise en correspondance appliqué à la synthèse de nouvelles vues d'une scène 3D à partir de photographies prises en des points différents. L'algorithme cherche à apparier le plus grand nombre possible de pixels des images source. Pour cela, une nouvelle technique de croissance de régions appelée propagation est employée. La propagation est une technique qui permet un appariement quasi-dense des images source, c'est-à-dire : tous les pixels sont mis en correspondance, sauf ceux où la *luminance* varie peu. L'évolution de la propagation est basée sur une stratégie d'utilisation des meilleures paires de départ d'abord. Ces paires sont obtenues à partir d'une mise en correspondance de points d'intérêt (cette dernière étant réputée la plus fiable qu'on puisse obtenir). Les résultats obtenus dans la thèse de Maxime Lhuillier [Lhu00] prouvent que ce nouvel algorithme est robuste et capable de traiter des images difficiles à traiter auparavant par d'autres méthodes de mise en correspondance. Évidemment, il y a des limitations à son utilisation, comme une trop grande distorsion de la texture entre les images à apparier, distorsion due au changement de point de vue et la trop grande présence de *textures* répétitives.

L'algorithme de propagation que nous venons de décrire s'insère dans la catégorie des problèmes irréguliers adaptatifs. Cette catégorie de problèmes présente une évolution des données de départ au fur et à mesure que le calcul est réalisé. Ce genre de problème possède un comportement dynamique lequel commence par avoir un haut degré de parallélisme qui diminue très rapidement.

Le contexte de parallélisation de l'algorithme de propagation est important aussi, car il cadre le type de solution que nous avons proposée. Nous avons décidé de faire nos études de parallélisation en ayant pour cible les grappes de processeurs. Nous sommes donc dans un contexte de machine à mémoire distribuée où les communications entre les nœuds sont faites par échange de messages. Ainsi, la stratégie d'utilisation des meilleures paires comme graines de l'algorithme de propagation, laquelle passe forcément par une recherche centralisée sur une structure centralisée de stockage des paires, est devenue notre plus grand problème. Il nous a donc fallu proposer une version distribuée d'un tel algorithme de façon à préserver au maximum les attributs de la version séquentielle.

Dans les deux sections qui suivent, nous présentons un bilan des contributions apportées par notre étude vers une version distribuée de l'algorithme de propagation, puis un bilan concernant les limitations observées dans notre proposition. Ensuite, une dernière section sera consacrée à l'énumération des possibles évolutions que nous envisageons pour les différents aspects abordés dans cette thèse.

8.1 Bilan des Contributions

Après une période d'analyse des caractéristiques fonctionnelles de l'algorithme de propagation dans le contexte de la synthèse d'images, nous avons démarré l'élaboration d'une solution parallèle pour le problème. Nous avons adopté une approche incrémentale, c'est-à-dire : les modifications sur l'algorithme originel ont été faites par étapes. Dans chaque nouvelle étape, une implantation sur la grappe a été réalisée et des résultats ont été générés. Nous avons orienté nos recherches par l'analyse des résultats, ainsi les points faibles de chaque solution proposée étaient détectés et les bases pour une nouvelle version parallèle étaient acquises. À travers l'application de cette méthode nous avons pu formuler des solutions nouvelles pour la parallélisation d'un algorithme récent d'appariement d'images (la propagation). Les principales contributions de notre étude sont énumérées dans la suite :

1. un mécanisme de contrôle de la qualité de l'appariement final pour les versions parallèles. Le maintien de la qualité du résultat final était une barrière importante à la mise en parallèle, car dans l'algorithme séquentiel elle était assurée par la stratégie "meilleur d'abord" lorsque une nouvelle paire de points était prise pour servir de graine à l'algorithme de propagation. Or, lorsqu'on distribue les appariements de départ à travers un nombre n de nœuds, une telle stratégie devient impraticable. Ainsi, nous avons proposé une solution basée sur une estimation du pourcentage de l'image qui devrait être appariée pour forcer l'algorithme à avancer jusqu'à ce que le niveau souhaité de qualité soit atteint. Pour réaliser ce contrôle, nous avons du centraliser sur une structure unique les appariements calculés par les nœuds de travail. Avec cette approche nous avons réussi des appariements finaux avec une qualité très proche de la version séquentielle (toujours au delà de 90%) ;

2. le contrôle de la redondance de calcul réalisé par les nœuds de travail. En fait, la redondance de calcul a été une de plus grosses difficultés que nous avons eu pour pouvoir paralléliser l'algorithme de propagation. Les calculs redondants existent lorsqu'une région qui a déjà été appariée l'est à nouveau. Cela se produit dans les versions parallèles, car un nœud de travail ne sait pas ce que les autres nœuds ont déjà fait. Le problème n'existe pas dans la version séquentielle car l'état du système est connu par l'algorithme de propagation à tout moment. Dans une version parallèle, où plusieurs sous-ensembles de la propagation évoluent indépendamment, les nœuds de travail ne connaissent pas les appariements des autres. Une solution partielle à ce problème est notre idée d'utilisation du concept de tranches souples proposée au chapitre 6 ;

3. les deux contributions précédentes ont permis finalement l'élaboration d'une version parallèle capable de traiter le problème de l'adaptation d'un algorithme séquentiel qui présente une évolution déterminée par un ordre global

préétabli aux contraintes d'une version parallèle distribuée où l'ordre global est remplacé par un ordre local distribué parmi les nœuds. Sur cette version parallèle de l'algorithme nous avons pu ajouter des améliorations comme la régulation de charge de travail et la hiérarchisation du nœud de centralisation.

4. une contribution d'ordre plus pragmatique de notre travail est l'effective réduction du temps d'exécution par rapport à la version séquentielle pour les quatre exemples testés.

8.2 Bilan des Limitations

Malgré les résultats satisfaisants présentés au long des chapitres 6 et 7, certains aspects de notre version parallèle de l'algorithme de propagation indiquent que des solutions encore plus performantes pourraient être atteintes. Nous considérons donc ces aspects comme étant des limitations de la solution proposée dans notre travail.

1. la solution que nous proposons n'a pas été testée sur un grand nombre d'exemples. Cela est une limitation car nous ne savons pas si notre algorithme parallèle supporte des images de taille plus conséquente ;

2. le problème de la redondance de calcul a été maîtrisé, mais pas complètement éliminé. Cela empêche des performances plus pointues, car nous restons dans le meilleur de cas avec une redondance de calcul supérieure à 30% de la surface des tranches attribuées aux nœuds de travail. Nous ne savons pas s'il existe une solution pour l'algorithme parallèle de propagation où la redondance de calcul n'existe pas ;

3. pour contrôler la qualité de l'appariement final, nous avons adopté une stratégie de centralisation des appariements calculés par les nœuds de travail, lesquels sont envoyés à un nœud maître. Cette approche reste un goulot d'étranglement de l'algorithme. L'amélioration que nous avons ajoutée avec la hiérarchisation du nœud de centralisation n'a pas apporté beaucoup de performance. Nous croyons que cette stratégie peut certainement être perfectionnée pour soulager encore plus le travail de triage du nœud maître.

4. Finalement, malgré l'ajout d'un mécanisme de régulation de charge, nous avons constaté que l'irrégularité du problème n'a pas été entièrement dominée. Il reste encore des inégalités dans la charge de travail des nœuds. Nous croyons qu'une étude plus détaillée dans ce domaine peut conduire à une amélioration sensible dans l'utilisation des ressources de calcul.

8.3 Perspectives

Nous croyons que les performances obtenues jusqu'ici valident le modèle parallèle que nous avons choisi ainsi que l'algorithme parallèle de propagation proposée. La réduction des temps d'exécution a été significative et les ressources employées pour l'obtenir ne sont pas énormes. Il nous semble donc raisonnable de continuer nos recherches sur ce thème car nous croyons être très proches des performances qui permettront l'emploi de l'algorithme de propagation dans des applications réelles.

Pour avancer, il faut consacrer un peu plus de temps de travail pour éliminer certains aspects de notre solution parallèle qui constituent des limitations aux performances obtenues. Un chemin que nous semble possible est l'élaboration d'un schéma de hiérarchisation du nœud maître utilisant les facilités de programmation légère offertes par Athapascan. Il nous semble aussi qu'il y a des gains de performance à obtenir avec l'amélioration de la régulation de la charge de travail des nœuds de calcul.

Quatrième partie

Annexe

Glossaire Imagerie

Contrainte épipolaire : restriction qui oblige que les correspondants possibles d'un point sur une deuxième image soient situés sur une même droite appelée alors épipolaire. L'utilisation de la contrainte épipolaire est possible lors de la mise en correspondance si l'on suppose que la scène est rigide. Ainsi, seuls deux points qui se ressemblent et qui sont situés sur la même droite épipolaire peuvent poser un problème à un algorithme d'appariement.

Contraste : qualité d'une image liée à la possibilité de distinguer les parties sombres des parties claires. Le contraste d'une image peut être défini comme un rapport entre les zones correspondant à une intensité maximale et minimale, par exemple $c = (I_{max} - I_{min})/(I_{max} + I_{min})$ où I_{max} et I_{min} sont respectivement les intensités maximale et minimale.

Courbure (d'une surface) : grandeur liée, pour une courbe, au changement de direction de la tangente à cette courbe lorsque l'on se déplace sur celle-ci. Pour une surface, la courbure se définit en chaque point, pour chaque direction, à partir d'un vecteur tangent à la surface.

Lissage : procédé ayant pour but de retrouver les valeurs d'une fonction $f(x)$ dont les observations sont bruitées par des erreurs d'observation. Dans le cadre de l'imagerie, il s'agit de la normalisation (régularisation) de la *texture* globale de la surface d'une vue.

Luminance : grandeur mesurant la brillance par unité de surface d'une source lumineuse. La luminance est égale à l'intensité du rayonnement émis par une source dans une direction donnée lors de la projection de la surface de cette source sur un plan perpendiculaire à la direction d'observation.

Luminescence : dans le cadre de l'imagerie, il s'agit de l'ensemble de luminances de la surface (ou d'une région de la surface) d'une source lumineuse.

Réflexions spéculaires : type de réflexion renvoyant la lumière principalement dans une direction après l'avoir reçue d'une seule direction selon les principes d'optique géométrique. Le résultat est semblable au reflet produit par un miroir ou à un

reflet brillant. La réflexion spéculaire est dépendante du point de vue.

Solution la plus lisse : solution optimale d'un procédé de lissage sur la surface d'une image donnée.

Texture : répétition spatiale d'un même motif dans différentes directions de l'espace. La texture traduit un aspect homogène de surface d'un objet sur une image (par exemple : le bois d'une table. Elle se manifeste donc par une information visuelle qui permet de la décrire qualitativement à l'aide d'adjectifs comme : grossière, fine, lisse, tachetée, granuleuse, marbrée, régulière, irrégulière, etc.

Zones occultées : zones d'une vue formées par des pixels qui se trouvent cachés des rayons lumineux originaires des points de vues par des objets opaques présents au premier plan. Ce phénomène est plus ou moins courant selon le type de scène observée. Dans une scène d'extérieur, des occultations sont presque toujours présentes sous la forme de personnages, arbres, bâtiments devant un paysage. Elles sont moins fréquentes dans les scènes d'intérieur et si les points de vue sont suffisamment proches.

Cinquième partie

Bibliographie

Bibliographie

[ACGK88] S. Ahuja, N. Carriero, D. Gelertner, and V. Krishnaswamy. Matching language and hardware for parallel computation in the linda machine. *IEEE Transactions on Computers*, 37(8) :pp. 921–929, 1988.

[Agh85] Gul A. Agha. *ACTORS : a model of concurrent computation in distributed systems*. PhD thesis, University of Michigan, Computer and Communication Science, USA, 1985.

[Agh90] G. Agha. *ACTORS : a model of concurrent computations in distributed systems*. MIT Press, 1990.

[AP92] H. Alnuweiri and V. Prasanna. Parallel architectures and algorithms for image component labeling. *IEEE Trans. Patt. Anal. Machine Intell.*, 14 :pp. 1014–1034, 1992.

[BB] D. Ballard and C. Brown. *Computer Vision*. Prentice Hall, Englewood Cliffs.

[BDB94] J. Barron, Fleet D., and S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12(1) :pp. 43–77, 1994.

[Ber97] Pierr-Eric Bernard. *Parallélisation et multiprogrammation pour une application irregulière de dynamique moléculaire opérationelle*. PhD thesis, Institut National Polytechnique de Grenoble, France, 1997.

[BJK+95] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leiserson, K. Randall, Y. Zhou, and K. Randall. Cilk : an efficient multithreaded runtime system. *ACM SIGPLAN Notices*, 30(8) :pp. 207–216, 1995.

[Bla98] Jérôme Blanc. *Sunthèse de nouvelles vues d'une scène 3D à partir d'images existantes*. Thèse de doctorat en informatique, Institut National Polytechnique de Grenoble, France, 1998.

[BM97] J. Blanc and R. Mohr. Towards fast ans realistic images synthesis from real views. In *Proceedings of the 10th Scandinavian Conference on Image Analysis*, pages pp. 455–461, 1997.

[BN84] A. Birrel and B. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1) :pp. 39–59, 1984.

[BVZ98] Y. Boykov, O. Veksler, and R. Zabih. Disparity component matching for visual correspondence. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages pp. 470–475, 1998.

[CR87] M. Cosnard and Y. Robert. Algorithmique parallèle : une étude de complexité. *Techniques et Science Informatiques*, 6 :pp. 115–125, 1987.

[CRFS] N. Copty, S. Ranka, G. Fox, and R. Shankar. Solving the region growing problem on the connection machine. In *Proceedings of the 22th International Conference on Parallel Processing.*

[CRFS94] N. Copty, S. Ranka, G. Fox, and R. Shankar. A data parallel algorithm for solving the region growing problem on the connection machine. *Journal of Parallel and Distributed Computing*, 21(1) :pp. 160–168, 1994.

[DDP97] F. Durand, G. Drettakis, and C. Puech. The visibility skeleton : a powerful and efficient multi-purpose global visibility tool. In *Proceedings of the SIGGRAPH 97*, pages pp. 89–100, 1997.

[DJ89] U.R. Dhond and Aggarwal J.K. Structure from stereo - a review. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6) :pp. 1489–1510, 1989.

[Fal94] L. Falkenhagen. Depth estimation from stereoscopic image pairs assuming piecewise continuous surface. In *Proceedings of the Workshop on Image Processing for Broadcast and Video Production*, pages 115–127, Hamburg, Germany, 1994.

[FD90] B. Fagin and A. Despain. The performance of parallel prolog programs. *IEEE Transactions on Computers*, 39(12) :pp. 1434–1445, 1990.

[FL94] P. Fua and Y. Leclerc. Using 3-dimensional meshes to combine image-based and geometry-based constraints. *Lectures Notes in Computer Science*, 800 :281–294, 194.

[FLR+98] O. Faugeras, S Laveau, L. Robert, G. Csurka, and C. Zeller. Complete dense stereovision using level set methods. In *Proceedings of the 5th European Conference on Computer Vision*, pages pp. 379–393, 1998.

[Fly72] M. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, C-21(9) :pp. 948–960, 1972.

[FMD01] L.G. Fernandes, N. Maillard, and Y. Denneulin. Parallelizing a dense matching region growing algorithm for an image interpolation application. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages pp. 491–495, 2001.

[Fos95] I. Foster. *Designing and building parallel programs.* Addison-Wesley, 1995.

[Fua91] P. Fua. Combining stereo and monocular information to compute dense depth maps that preserve discontinuities. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages pp. 1292–1298, 1991.

[FvDFH91] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes. *Computer graphics : principle and practice.* Addison-Wesley, 1991.

[Gra66] R.L. Graham. Bounds for certain multiprocessors anomalies. *Bell System Technologie Journal*, 45 :pp. 1563–1581, 1966.

[GRCD98] F. Galilé, J-L. Roch, G. Cavalheiro, and M. Doreille. Athapascan-1 : on-line building data flow graph in a parallel language. In *Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques*, pages pp. 88–95, 1998.

[GRV95] T. Gautier, J-L. Roch, and G. Villard. Regular versus irregular problems and algorithms. In *Proceedings of the 1995 Workshop on Parallel Algorithms for Irregularly Structured Problems*, pages pp. 1–25, 1995.

[GSB97] M.A. Garcia, A.D. Sappa, and L. Basafie. Efficient aproximation of range images through data-dependent adaptative triangulation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages pp. 628–633, 1997.

[Gui95] F. Guinand. *Ordonnancement avec communications pour architectures multiprocesseurs dans diver s modèles d'exécution*. PhD thesis, Institut National Polytechnique de Grenoble, France, 1995.

[Hal86] R.H. Halstead. Parallel symbolic computing. *Computer*, 19(8) :pp. 35–43, 1986.

[HP74] S.L. Horowitz and T. Pavlidis. Picture segmentation by a directed split-and-merge procedure. In *Proceedings of the 2nd International Joint Conference on Pattern Recognition*, pages pp. 424–433, 1974.

[HS81] K.P. Horn and G. Scunck. Determining optimal flow. *Artificial Intelligence*, (17) :pp. 185–203, 1981.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of the Alvey Vision Conference*, pages pp. 147–151, 1988.

[IB94] S.S. Intille and A.F.. Bobick. Disparity-space images and large occlusion stereo. In *Proceedings of the 3rd European Conference on Computer Vision*, pages pp. 179–186, 1994.

[JBHD95] J. Jájá, D. Bader, D. Harwood, and L. Davis. Parallel algorithms for image enhancement and segmentation by region growing with an experimental study. Technical report, Institute for Advanced Computer Studies, University of Maryland, 1995.

[Kos93] A. Koschan. What is new in computational stereo since 1989 : a survey on stereo papers. Technical report, University of Berlin, 1993.

[Lhu00] Maxime Lhuillier. *Modélisation pour la synthèse d'images à partir d'images*. Thèse de doctorat en informatique, Institut National Polytechnique de Grenoble, France, 2000.

[LQ99a] M. Lhuillier and L. Quan. Image interpolation by joint view triangulation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, volume 2, pages pp. 139–145, 1999.

[LQ99b] M. Lhuillier and L. Quan. Joint view triangulation for two views. In *Proceedings of the 12th Conference on Vision Interface*, 1999.

[LQ00a] M. Lhuillier and L. Quan. Edge-constrained joint view triangulation for image interpolation. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, volume 2, pages pp. 218–224, 2000.

[LQ00b] M. Lhuillier and L. Quan. Robust dense matching using local and global geometric constraints. In *Proceedings of the 15th International Conference on Pattern Recognition*, volume 1, pages pp. 968–972, 2000.

[MGG99] M. Montoya, C. Gil, and I. Garcia. Load balancing for a class of irregular and dynamics problems : region growing image segmentation algorithms. In *Proceedings of the 7th Euromicro Workshop on Parallel and Distributed Processing*, pages pp. 163–169, 1999.

[Mon87] O. Monga. An optimal region growing algorithm for image segmentation. *International Journal of Pattern Recognition and Artificial Intelligence*, 1(3) :pp. 351–375, 1987.

[Mor77] H.P. Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, page pp. 584, 1977.

[ORF92] C.-W. Ou, S. Ranka, and G. Fox. Fast and parallel mapping algorithms for irregular problems. Technical report, Northeast Parallel Architectures Center, Siracuse University, 1992.

[ORF93] C.-W. Ou, S. Ranka, and G. Fox. Fast mapping and remapping algorithm for irregular and adaptative problems. In *Proceedings of the 1993 International Conference on Parallel and Distributed Systems*, pages pp. 279–283, 1993.

[PJF85] S.B. Pollard, Mayhew J.E.W., and J.P. Frisby. Pmf : a stereo correspondence algorithm using a disparity gradient constraint. *Perception*, (14) :pp. 449–470, 1985.

[PS85] F. Preparata and M.I. Shamos. *Computational geometry, an introduction*. Springer-Verlag, Berlin, Germany, 1985.

[Ran93] A. Ranade. A framework for analyzing locality and portability issues in parallel computing. *Lecture Notes in Computer Science*, 678 :pp. 185–193, 1993.

[Rip90] D. Rippa. Minimal roughness property of the delauney triangulation. *Computer Aided Geometric Design*, (7) :pp. 489–497, 1990.

[Sch96] D. Scharstein. Stereo vision for view synthesis. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages pp. 852–858, 1996.

[SD96] S. Seitz and C. Dyer. View morphing. In *Proceedings of the SIGGRAPH*, pages pp. 21–30, 1996.

[Ski95] D. Skillicorn. *Foundations of parallel programming*, volume Number 6 in International series on parallel computation. Cambridge University Press, 1995.

[SMB98] C. Schimid, R. Mohr, and C. Bauckhage. Comparing and evaluating interest points. In *Proceedings of the 6th International Conference on Computer Vision*, pages pp. 230–235, 1998.

[SOHL+96] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI : the complete reference*. MIT Press, 1996.

[ST94] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages pp. 593–600, 1994.

[ST98] D. Skillicorn and D. Talia. Models and languages for parallel computation. *ACM Computing Surveys*, 30(2) :pp. 123–169, 1998.

[SZ99] R. Szeliski and R. Zabih. An experimental comparison of stereo algorithms. In *Vision algorithms : theory and practice (workshop helded during ICCV'99)*, pages pp. 59–66, 1999.

[Tan92] A. Tanembaum. *Modern Operating Systems*. Prentice Hall, 1992.

[Til88] J. Tilton. Image segmentation by iterative parallel region growing with applications to data compression and image analysis. In *Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation*, pages pp. 357–360, 1988.

[vdSD96] A. van der Steen and J. Dongarra. Overview of recent supercomputers. Technical report, UT-CS-96-325, Department of Computer Science, University of Tennessee, 1996.

[WLR90] M. Willebeck-LeMair and A. Reeves. Solving non-uniform problems on simd computers : case study on region growing. *Journal of Paralle and Distributed Computing*, 8 :pp. 135–149, 1990.

[ZFC99] A. Zisserman, A. Fitzgibbon, and G. Cross. Vhs to vrml : 3d graphical models from video sequences. In *IEEE International Conference on Multimedia and Systems*, pages pp. 51–57, 1999.

[Zuc76] S. Zucker. Region growing : childhood and adolescence. *Computer Graphics and Image Processing*, (5) :pp. 382–399, 1976.

Résumé : Depuis quelques années, la complexité croissante de besoins informatiques dans tous les domaines de recherche scientifique et technique exige une puissance de calcul sans cesse plus importante. Dans ce contexte, le calcul parallèle apparaît comme un outil vital qui permet d'exploiter la capacité de calcul de plusieurs microprocesseurs travaillant ensemble. Le domaine auquel le calcul parallèle sera appliqué dans cette thèse est celui de l'imagerie, plus spécifiquement la synthèse d'images à partir d'images réelles. Pour permettre la création de nouvelles vues virtuelles à partir de scènes réelles, une des étapes les plus coûteuses en temps de calcul est la phase d'appariement des images. Il s'agit de la formation de paires de pixels qui représentent le même point sur les surfaces de deux images source. Le problème principal que nous cherchons à résoudre dans cette thèse est la proposition d'une version parallèle pour l'algorithme d'appariement quasi-dense (aussi appelé algorithme de propagation) d'images qui est basé sur une stratégie adaptative globale. Cette version doit être capable de préserver la qualité du résultat final obtenu par la version séquentielle de l'algorithme de propagation tout en réduisant le temps d'exécution dans un contexte de programmation orienté vers les grappes de processeurs.

Mots Clés : Calcul parallèle. Appariement d'images. Algorithme d'appariement quasi-dense. Propagation. Grappes de processeurs. Ordonnancement.

Abstract : In every field of scientific and industrial research, the extension of the use of Computer Science has resulted in an increasing need for computing power. Within this context, parallel computing appears as an important tool in order to allow the exploitation of many microprocessors working together. The field to which parallel computing will be applied in this thesis is computer graphics, more preciselly in synthesis of virtual scenes from real images. In this process, the pixel matching step is the most computing expensive task. It deals with creating couples of pixels which represent the same point over two source scenes surface. The main proposal of this thesis is a new version of the "quasi-dense" pixel matching algorithm (also called propagation algorithm). This new algorithm is a parallel version that relies on a global adaptative strategy. This version must be able to preserve the final result quality obtained by the sequential version. Furthermore, it must also reduce its execution time in a cluster-oriented programming context.

Keywords : Parallel computing. Pixel matching. "Quasi-dense" algorithm. Propagation. Clusters. Scheduling.